Firebase: trabajar en la nube

Jesús Tomás
Vicente Carbonell
Jordi Bataller
Jaime Lloret

Firebase Authentication, Cloud Firestore, Cloud Functions, Cloud Storage, Analytics, Crashlytics, Test Lab, Redes sociales y mucho más.

Alfaomega
Marcombo

Firebase: trabajar en la nube

Firebase: trabajar en la nube

Jesús Tomás
Vicente Carbonell
Jordi Bataller
Jaime Lloret

Alfaomega

Marcombo

Diseño de la cubierta: ENEDENÚ DISEÑO GRÁFICO

Datos catalográficos
Tomás, Jesús; Carbonell, Vicente; Bataller, Jordi; Lloret, Jaime
Firebase: trabajar en la nube
Primera Edición

Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 978-607-538-415-3
Formato: 17 x 23 cm Páginas: 412

Firebase: trabajar en la nube
Jesús Tomás, Vicente Carbonell, Jordi Bataller y Jaime Lloret
ISBN: 978-84-267-2660-5 de la edición publicada por MARCOMBO, S.A., Barcelona, España

Primera edición: Alfaomega Grupo Editor, México, enero 2019

Pitágoras 1139, Col. Del Valle, 03100, México D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Pág. Web: **http://www.alfaomega.com.mx**
E-mail: **atencionalcliente@alfaomega.com.mx**

ISBN: 978-607-538-415-3

Nota importante:
La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro y en el material de apoyo en la web, ni por la utilización indebida que pudiera dársele.

Edición autorizada para venta en México y todo el continente americano.

Impreso en México. Printed in Mexico.

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. – Dr. Isidoro Olvera (Eje 2 sur) No. 74, Col. Doctores, Ciudad de México – C.P. 06720. Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396 – E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. – Calle 62 No. 20-46, Barrio San Luis, Bogotá, Colombia, Tels.: (57-1) 746 0102 / 210 0415 – E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Av. Providencia 1443. Oficina 24, Santiago, Chile
Tel.: (56-2) 2235-4248 – Fax: (56-2) 2235-5786 – E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. – Av. Córdoba 1215, piso 10, CP: 1055, Buenos Aires, Argentina, – Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

Para Erica, con el amor de su padre

JESÚS TOMÁS

Para Maite, Inés y Carles, con todo el cariño, por su paciencia y comprensión

VICENTE CARBONELL

Para Mar, Ferran y Cristina

JORDI BATALLER

Para Cristina y Claudia, por el tiempo que no he pasado junto a vosotras

JAIME LLORET

Índice general

¿Cómo leer este libro?

Este libro se ha estructurado en varios capítulos que abordan principalmente el uso de Firebase para el desarrollo de aplicaciones en Android. No es precisa una lectura secuencial, el lector puede ir directamente al capítulo que le interese.

En este libro se da por supuesto que el lector tiene experiencia en programación sobre Android. No se tratan temas relativos al desarrollo básico como los componentes o estructura de las aplicaciones. Si el lector está interesado en un texto que aborde la programación en Android desde el principio, le recomendamos El gran libro de Android publicado en esta misma editorial. También puede ser de interés la lectura de otros libros de esta colección como El gran libro de Android Avanzado, Plataformas Android: Wear, TV, Auto y Google Play Games o Android Things y Visión Artificial.

El libro que tienes entre las manos no ha sido concebido solo para ser leído. Es más bien una guía estructurada que te irá proponiendo una serie de ejercicios, actividades, vídeos explicativos, test de autoevaluación, etc. Parte de este material y algunos recursos adicionales están disponibles en la web www.androidcurso.com. En ella se publicarán las novedades, erratas e información complementaria relativas a este libro. Además encontrarás:

- **Material adicional sobre Android:** Encontrarás nuevos tutoriales, vídeos, referencias, etc., no incluidos en el libro.
- **Código abierto de proyectos Android:** Muchos alumnos que han realizado cursos con nosotros han tenido la generosidad de compartir sus proyectos. Te recomendamos que consultes la lista de proyectos disponibles de código abierto: puedes aprender mucho estudiando su código.
- **Cursos online:** Si te interesa ampliar tu formación, puedes matricularte en cursos sobre Android impartidos por la Universidad Politécnica de Valencia. Incluso puedes obtener un título de Especialización o de Máster de forma 100 % online.

A lo largo del libro se utilizan los siguientes iconos para indicar distintos tipos de recursos:

Objetivos: Antes de empezar cada capítulo lee con detenimiento la introducción y los objetivos.

Ejercicio: La mejor forma de aprender es hacer ejercicios. No tendrás más que ir siguiendo los pasos uno tras otro para descubrir cómo se resuelve el ejercicio propuesto. Para que no se te haga pesado teclear todo el código, te sugerimos que lo copies y pegues desde la página web del curso.

Práctica: Este será el momento de que tomes la iniciativa y trates de resolver el problema que se propone. Recuerda que para aprender, hay que practicar.

Solución: Te será de ayuda si tienes dificultades al resolver una práctica o simplemente quieres comparar tu solución con otra diferente.

Vídeo [Tutorial]: Vídeos grabados por los autores donde se exponen de forma didáctica los aspectos clave de los temas tratados. Se utiliza una moderna herramienta desarrollada en la Universidad Politécnica de Valencia que te permitirá ver simultáneamente las presentaciones y al profesor.

Enlaces de interés: Internet te será de gran ayuda para completar la información necesaria para programar en Android. Te proponemos las páginas más interesantes de cada apartado.

Preguntas de repaso: ¿Has comprendido correctamente los aspectos clave? Sal de dudas haciendo los test de autoevaluación.

Trivial programación Android: Instálate esta app y mide tus conocimientos jugando en red contra otros oponentes.

CAPÍTULO 1.

Introducción a Firebase y Autentificación

JESÚS TOMÁS

Firebase es la nueva plataforma de Google con la que conseguiremos que nuestras aplicaciones trabajen en la nube. Toda la problemática que supone trabajar en Internet (almacenamiento, registros de usuarios, gestión del backend…) la tenemos resuelta de una forma sencilla e integrada. Ahorraremos cientos de horas de implementación y, además, al estar basado en la infraestructura de Google, obtendremos resultados altamente fiables, seguros y escalables. Entre los servicios que incorpora podemos destacar:

- Identificación de usuario
- Bases de datos NoSQL y de tiempo real
- Mensajería en la nube
- Notificaciones push
- Almacenamiento de ficheros y hosting
- Análisis en tiempo real de lo que hacen los usuarios
- Herramientas de monetización
- Y mucho más

Se trata de una plataforma disponible para diferentes sistemas (Android, iOS, Web, C++ e Unity). Además, el coste del servicio es gratuito para un uso moderado; solo tendremos que pagar cuando nuestra aplicación comience a tener éxito.

Google quiere centralizar en Firebase todas plataformas para la gestión de apps (Analytics, Admod, Parse…). Desde una misma consola tendremos acceso a todos los datos de nuestra aplicación.

Firebase puede englobarse como un servicio del tipo Mobile Backend as a Service (MBaaS). Empezaremos este capítulo describiendo algunas alternativas que ofrecen servicios similares.

Continuaremos con una introducción a Firebase, resaltaremos sus ventajas e inconvenientes. Pasaremos a realizar una descripción de cada uno de los servicios que integra la plataforma.

En la última parte del capítulo, aprenderemos a utilizar los servicios de autentificación de Firebase. Estos incluyen la posibilidad de pedir al usuario unas credenciales basadas en correo y contraseña o utilizar las credenciales que ya disponen en redes sociales.

Objetivos

- Comparar Firebase con otros servicios del tipo MBaaS.
- Enumerar los servicios integrados en Firebase.
- Describir el proceso de autentificación que proporciona Firebase.
- Implementar un proceso de autentificación automático con FirebaseUI.
- Implementar procesos de autentificación personalizados basados en correo y contraseña, Google, Facebook, Twitter y anónima.
- Describir en qué consiste la unificación de cuentas.

1.1. Mobile Backend as a Service (MBaaS)

Firebase puede ser catalogado como una plataforma de tipo Mobile Backend as a Service (MBaaS), por lo que puede ser interesante describir primero este tipo de plataformas y qué alternativas existen en la actualidad.

Un MBaaS proporciona una serie de servicios en la nube (autentificación, almacenamiento, notificaciones, análisis…) para el desarrollo de aplicaciones móviles y Web. La principal finalidad es aumentar la productividad en el desarrollo de apps, dado que el programador no ha de desarrollar los diferentes servicios en la nube que requiere su aplicación. Además, la plataforma nos proporciona todas las infraestructuras de acceso, almacenamiento y seguridad. Otra ventaja es que podemos gestionar todas nuestras aplicaciones desde una consola Web unificada (el Backend).

Aunque la mayoría de los MBaaS se enfocan al desarrollo de aplicaciones móviles, realmente, estos servicios también pueden ser usados para el desarrollo de aplicaciones Web o de escritorio. Por esta razón, también se conocen simplemente como Backend as a Service (BaaS), sin el término Mobile en el nombre.

Los MBaaS suelen estar formados por un SDK para diferentes plataformas o lenguajes y un Backend (normalmente Web) para la gestión de proyectos. El acceso a la plataforma desde los diferentes terminales suele realizarse mediante servicios Web tipo REST.

En la actualidad existen muchos MBaaS. De hecho, las principales empresas tecnológicas como Google, Apple, Amazon o Microsoft han sacado recientemente una plataforma con este propósito. Veamos algunos de ellos:

AWS Mobile (Amazon)

Amazon es una de las empresas pioneras en ofrecer servicios Web. Recientemente ha sacado una plataforma específica para apps: AWS Mobile. Entre sus servicios destacamos:

- DynamoDB: bases de datos no SQL
- Cognito: Autentificación de usuarios
- Notificaciones push
- Almacenamiento
- Lambda: funciones ejecutadas en servidor
- Kinesis: procesamiento de datos en tiempo real

Azure (Microsoft)

Dentro de los servicios de computación en la nube ofrecidos por Azure, se ha desarrollado una línea específica para dar servicio a aplicaciones móviles (Mobile Apps).

- Dispone de SDK para iOS, Android, Windows o Mac.
- Ofrece servicio de autentificación basado en cuentas de Microsoft, Google o las principales redes sociales.
- Permite enviar notificaciones push a cualquier plataforma.
- Se puede programar la lógica de la aplicación en el backend utilizando C# o Node.js.

CloudKit (Apple)

Esta plataforma de Apple para potenciar los servicios de iCloud en iOS y macOS.

Ofrece servicios de autentificación, una base de datos privada, una base de datos pública y servicios de almacenamiento de datos. Para el desarrollo en Android no sería una alternativa viable.

Parse (Facebook)

Esta plataforma fue potenciada inicialmente por Facebook, pero en la actualidad han abandonado el proyecto y han pasado el testigo a una comunidad de código abierto. De hecho, una de las principales ventajas de esta plataforma es su versión en código abierto, tanto para el servidor como para SDK de decenas de plataformas (http://parseplatform.org/). En este caso ha de ser el desarrollador quien proporcione el servidor, pero, claro, conseguimos un servicio ilimitado sin coste por uso. Integra: MongoDB como base de datos, Amazon S3 bucket para almacenar ficheros y OneSignal para las notificaciones.

La siguiente tabla muestra una comparativa de estas plataformas:

	Firebase	AWS	CloudKit	Azure	Parse
Empresa	Google	Amazon	Apple	Microsoft	Facebook
Código abierto	no	no	no	no	sí
API nativo	Android iOS Unity JavaScript REST C	REST (Java JavaScript Perl PHP Python Ruby)	iOS Mac	Android iOS Windows Mac	Android iOS Unity JavaScript REST C Arduino *.net* PHP
Bases de datos	NoSQL	NoSQL /SQL	¿?	NoSQL /SQL	¿?

La siguiente tabla muestra una comparativa de los servicios ofrecidos sin coste:

	Firebase	AWS*	Azure**	Parse
Autentificación	-	-	500 K	-
Notificaciones	-	1 M /m 5 K usuarios/m	1 M	-
Cloud Messaging	-	1M mensajes/m	-	-
Almacenamiento	5 GB 1 GB/día	10 GB	1 GB	-
Hosting	1 GB 10 GB/m (dominio, SSL)	10 GB	1 GB (PHP)	-
Base de datos	1 GB 10 GB/m 100 conexiones	25 GB 200M solicit/m (SQL, NoSQL)	20 MB* (SQL, NoSQL)	-
Funciones	125K solicit/m 40K GB-seg/m	1M solicit/m 400K GB-seg/m	1M solicit 400K GB-seg	-
Test dispositivo	5 test/d real 10 test/d virtual	250 min	no	no

(-) servicio ilimitado

(*) El servicio gratuito solo se ofrece durante el primer año.

Amazon limita el uso de máquinas virtuales a 750 horas el primer año.

(**) Un máximo de 60 minutos de CPU/día y 1 GB de RAM.

Enlaces de interés

- http://searchmicroservices.techtarget.com/tip/Firebase-AWS-and-more-A-comparison-of-five-leading-MBaaS-providers

- https://www.infoworld.com/article/2842791/application-development/mbaas-shoot-out-5-cloud-platforms-for-building-mobile-apps.html
- https://db-engines.com/en/system/Amazon+DynamoDB%3BFirebase+Realtime+Database

1.2. Introducción a Firebase

Firebase empieza como una empresa, fundada en el año 2011, en San Francisco. Su finalidad era dar servicios de backend en el desarrollo de aplicaciones. Su producto inicial fue la base de datos en tiempo real, aunque luego se añadieron nuevos servicios. En el año 2014 fue comprada por Google y fue lanzada en el Google I/O del 2016, donde se integraron nuevos servicios con otros que antes ofrecía Google de forma independiente (Analytics, AdMob…).

Trabajar con Firebase nos aporta gran cantidad de ventajas:

- Una sola herramienta nos resuelve los principales problemas de trabajo en la nube: bases de datos, autentificación, almacenamiento, análisis detallados de nuestra app, etc.
- Sencillo y rápido de utilizar.
- Podemos utilizarlo desde múltiples plataformas. Tiene soporte para Android, iOS, Web (JavaScript), C++ e Unity, aunque podemos usarla desde cualquier sistema gracias a su API REST.
- Gratuito para un volumen de uso moderado.
- Firebase utiliza la infraestructura de Google, una de las más extensas y fiables de la actualidad. La responsabilidad de la seguridad y fiabilidad, recae ahora en Google.
- Escalable Google dispone de una extensa red de servicios en la nube.

También podemos destacar varias desventajas:

- La versión gratuita permite acceder a todas las funciones de Firebase, pero con algunas limitaciones. Por ejemplo, estamos limitados a 100 conexiones simultáneas en las bases de datos y a la autentificación de 10 K usuarios al mes. Estas limitaciones nos permiten trabajar con muchos tipos de aplicaciones. Como alternativas al plan gratuito (Spark) disponemos de dos planes: Flame (25 $/mes) y Blaze (pago por uso)[1].
- Al tratarse de un producto comercial (no es de código abierto) estamos expuestos a posibles cambios en las condiciones de comercialización.
- Las bases de datos NoSQL tienen sus ventajas, pero también presentan inconvenientes. El principal es que nos obliga a cambiar la forma de

[1] https://firebase.google.com/pricing/?hl=es-419

concebir las estructuras de datos. Además, presenta limitaciones a la hora de realizar búsquedas.

1.2.1. Los servicios de Firebase

Firebase integra diferentes servicios en una misma plataforma. En este apartado describiremos brevemente cada uno de ellos:

1.2.1.1. Herramientas de desarrollo

Authentication: Resuelve el problema de la autentificación de tus usuarios de una forma sencilla y sin riesgo de que su seguridad quede comprometida. Google se encarga de todo.

Puedes usar una autentificación basada en correo y contraseña o a través del número de teléfono. También puedes usar las principales redes sociales, incluidas Google, Facebook y Twitter y GitHub. Puedes gestionar tus usuarios de forma muy sencilla desde el backend.

Si utilizas una interfaz de usuario prediseñado (Firebase UI), puedes resolver la autentificación de tus usuarios en cuestión de minutos.

Realtime Database: Ya no necesitas crear un servidor con las bases de datos y acceder a él mediante servicios Web. Firebase nos ofrece una solución muy sencilla, fundada en bases de datos NoSQL, en las que todo se almacena en una estructura similar a JSON. Además, son bases de datos en tiempo real. Cualquier cambio en un nodo de la base de datos puede ser automáticamente sincronizado a nuestra aplicación.

Si un usuario pierde la conexión a Internet, podrá usar la caché local para publicar y almacenar cambios. Cuando el dispositivo se vuelve a conectar, los datos de la caché son almacenados.

Cloud Firestore: Una alternativa a Realtime Database, todavía en fase beta. Es muy parecida, sigue siendo NoSQL, de tiempo real y con un API parecido, aunque incorpora un nuevo modelo de datos más intuitivo. Cloud Firestore también cuenta con consultas más ricas y rápidas y resulta más escalable. Como único inconveniente encontramos una mayor latencia.

Cloud Functions: Puedes ejecutar código en el lado de servidor cuando se produzcan ciertos eventos. Por ejemplo, un cambio en la base de da-

tos, la autentificación de un usuario o un evento en Analytics. Este código ha de ser escrito en forma de función de JavaScript y será ejecutado en un entorno de Node.js seguro y administrado. Desde Cloud Functions tenemos acceso a todos los servicios de Firebase: bases de datos, notificaciones…

Cloud Storage: Almacena ficheros en la nube con el máximo nivel de disponibilidad. Las transferencias de ficheros pueden ser detenidas o reanudadas según la conectividad del dispositivo. Aprovecha las ventajas de la autentificación de usuario para dar los permisos adecuados a cada fichero. Se realiza redundancia multirregional, lo que garantiza una mínima latencia, aunque se dispare el número de acceso incluso en regiones muy distantes.

Hosting: Crea tu sitio o aplicación Web de forma sencilla. No importa dónde estén tus usuarios, gracias al almacenamiento con redundancia multirregional (CDN) la descarga será rápida en todo el mundo. Utiliza certificados SSL gratuitos para tus propios dominios.

ML Kit: Saca provecho de la experiencia de Google en aprendizaje automático para aplicarlo en la resolución de los problemas planteados en tu aplicación. Si no tienes experiencia en este campo, puedes implementar la funcionalidad que necesitas con solo unas pocas líneas de código. Si eres expertos en redes neuronales u optimización de modelos utiliza TensorFlow Lite para crear tus propias soluciones en tu aplicación móvil.

1.2.1.2. Herramientas de comprobación

Crashlytics: Descubre problemas de tu app para distintos dispositivos y así poderlos corregir lo antes posible. Resulta muy fácil de integrar, tanto en iOS como en Android. Desde el panel de control, podrás ver estos errores agrupados y por orden de importancia. Soluciona los problemas a partir de los potentes informes de fallos en tiempo real.

Crash Reporting: Ha sido reemplazado por Crashlytics.

Test Lab para Android: Para asegurarte de que tu aplicación funciona correctamente, gracias a este servicio, podrás probarla tanto en dispositivos

reales como virtuales. Puedes usarlo desde Android Studio, la consola Web o un entorno de integración continua. Aunque no hayas escrito pruebas de comprobación, puedes usar el rastreador, Robo, para que navegue por tu app de forma automática y detecte errores potenciales.

Performance Monitoring: Descubre si tus usuarios perciben que la aplicación funciona de forma fluida. Obtén estadísticas de rendimiento, tanto personalizadas, como automáticas. Conoce cómo el acceso a Internet afecta a los usuarios. Desde la consola podrás realizar el análisis de los resultados y detectar el origen de los problemas.

1.2.1.3. Herramientas de análisis e interacción con los usuarios

Google Analytics: Analiza toda la información sobre tu app (quién la instala, cómo la utiliza, qué notificaciones ha pulsado…) para así poder tomar las mejores decisiones. Descubre aspectos clave como quién, dónde o cuándo, para así poder averiguar el porqué. Analytics integra información del resto de herramientas como Invites, AdWords, AdMob… Se capturan automáticamente ciertos eventos sobre el uso de tu app, pero también puedes crear tus propios eventos personalizados.

Predictions: Utilizando herramientas de aprendizaje automático, Firebase crea automáticamente grupos de usuarios según las predicciones de su comportamiento. Con estas predicciones puedes mejorar la experiencia de usuario, aumentar los ingresos o la retención. Existen predicciones predeterminadas, como “harán compras” o “se mantendrán en la app”, pero también puedes crear tus propias predicciones personalizadas, como “pasará el primer nivel antes de una semana”.

Cloud Messaging: Envía mensajes entre aplicaciones (FCM) de forma gratuita y entre diferentes plataformas. También puedes enviar notificaciones a tus usuarios directamente desde la consola. Estas pueden contener datos personalizados o una hora de activación adaptada al usuario. Además, puedes analizar los resultados de conversiones desde Analytics.

Dynamic Links: Cuando te pasan un enlace a un vídeo de YouTube, la forma habitual de abrirlo es a través del navegador Web. Sin embargo, es posible que tengas instalada la app de YouTube, que te aportaría una mejor experiencia de usuario. Gracias a los Dynamics Links vas a poder definir URL asociados a

tu aplicación, de forma que el usuario pueda abrir el contenido desde esta. En caso de no tenerla instalada, se le puede sugerir que la instale, o si no está interesado o estás en un sistema no Android, abrir el contenido desde la Web.

Invites: Aumenta la visibilidad de tu aplicación haciendo que tus usuarios envíen invitaciones a sus amigos. Puedes promover este comportamiento con algún tipo de regalo. Las invitaciones pueden ser enviadas por correo electrónico o SMS. Al basarse en Dynamic Links, cuando un usuario reciba la invitación, si tiene instalada la app, se abrirá la actividad correspondiente; en caso contrario, se solicitará la instalación.

Remote Config: Cambia el comportamiento y aspecto de tu app desde la consola, sin tener que lanzar una nueva actualización. Puedes lanzar nuevo contenido, personalizar por segmentos de usuarios o realizar pruebas para validar mejoras.

App Indexing: Consigue que la búsqueda de Google muestre enlaces a tu app cuando los usuarios buscan contenido de tu app. Además, se mejorará el posicionamiento conseguido con tu app.

AdMob: Añade anuncios en tus apps para conseguir ingresos. La plataforma de publicidad móvil de Google ahora está integrada dentro de Firebase.

AdWords: Crea campañas de anuncios en el buscador de Google para dar a conocer tu aplicación y aumentar el número de instalaciones.

1.2.2. Agregar Firebase en un proyecto

En este curso vamos a continuar trabajando con el proyecto “Mis Lugares”, el cual solo permitía almacenar la información en local (en una lista en memoria o en una base de datos SQLite). A lo largo del curso vamos a hacer que esta aplicación trabaje con una base de datos Firebase almacenada en la nube. Además, integraremos muchos de los servicios disponibles en Firebase.

El primer paso va a ser crear un nuevo proyecto en la consola de Firebase y añadir en la aplicación todas las librerías necesarias.

Existen dos formas de agregar un proyecto a Firebase, manual y automática. Para aprender correctamente dónde se encuentra la información relevante reco-

mendamos realizar la manual. Una vez que controlemos el proceso, la automática resulta más rápida.

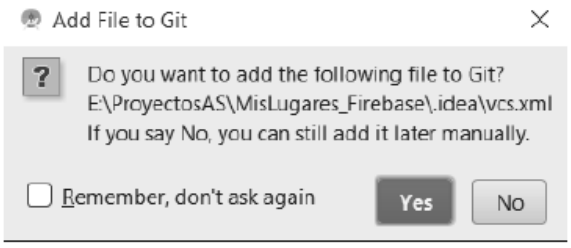
Ejercicio: Añadir Firebase a "Mis Lugares" de forma manual

1. Si has realizado el curso de EdX "Android: Introducción a la Programación", puedes utilizar el proyecto que creaste. En caso contrario, o si realizaste muchos cambios, desde Android Studio accede a File / New / Project from Version Control / GitHub. Indica el siguiente URL:

   https://github.com/jesus-tomas-girones/MisLugares_base.git

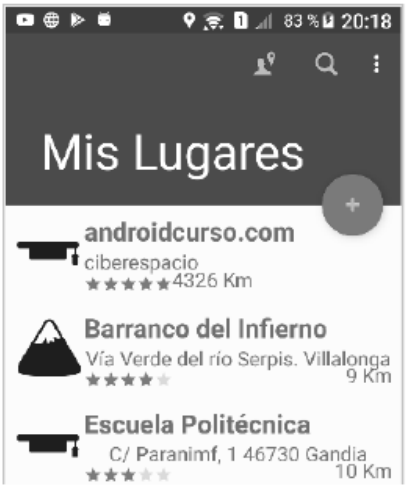
   Puedes crear el proyecto en el directorio que prefieras. Pulsa en **Clone**.

2. Aparecerá el siguiente cuadro de dialogo:



   Cada vez que crees un nuevo fichero en el proyecto, te preguntará si quieres añadirlo al repositorio Git. No tiene sentido que escribas en este Git, por lo que has de indicar **No**. Puede ser una buena idea recordar esta acción, para que no te vuelva a preguntar.

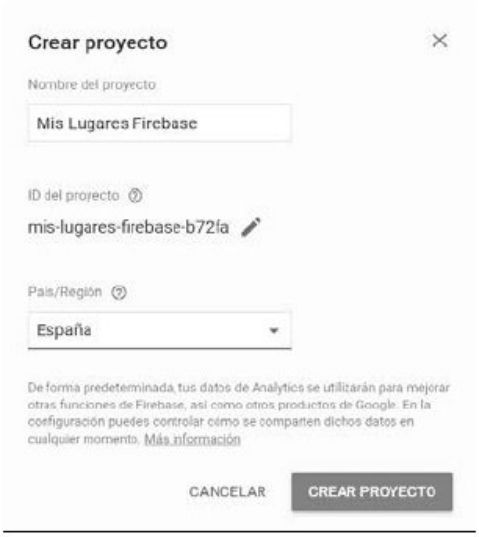
3. Ejecuta el proyecto y verifica que funciona correctamente:



4. Accede a la consola de Firebase (https://console.firebase.google.com). Necesitarás una cuenta de Google.

5. Pulsa en **Agregar proyecto**. Un proyecto puede dar soporte a varias aplicaciones en distintas plataformas como Android, iOS y Web. Toda aplicación que acceda a Firebase ha de registrarse en un proyecto.

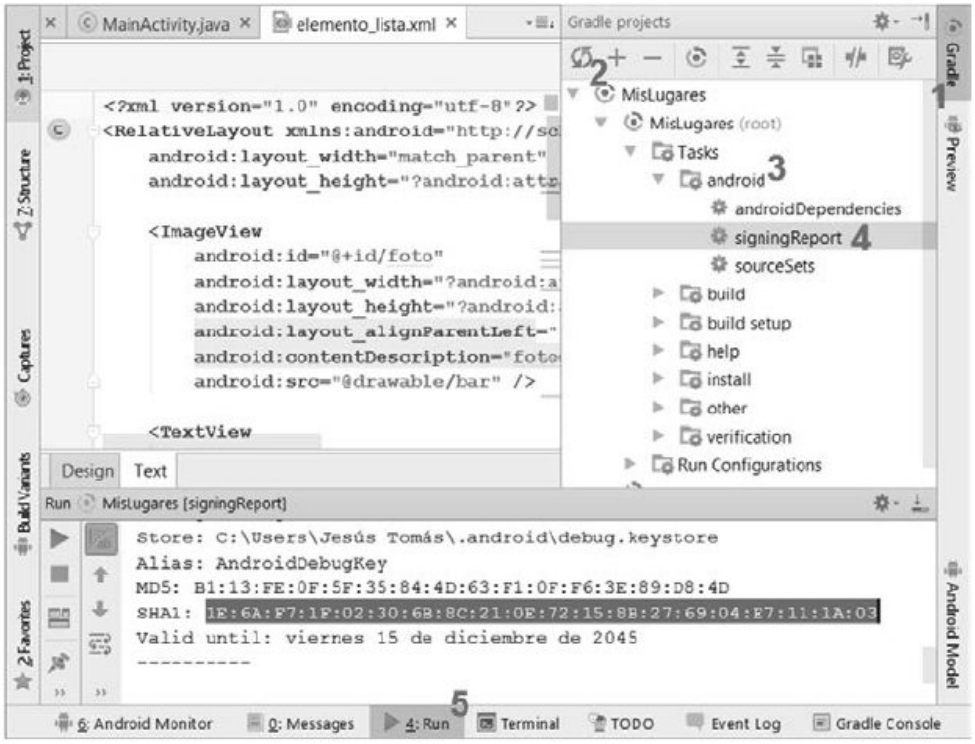
6. Introduce como nombre de proyecto, “Mis Lugares”, y un país de referencia. Esta última información se usa principalmente para determinar la moneda en la que se mostrarán los ingresos:



7. Accederás a la consola del proyecto:

8. Pulsa el botón **Agrega Firebase a tu aplicación de Android**.

Añade el nombre del paquete de la aplicación. El apodo, o nickname, es opcional. Permite tener varias aplicaciones con un mismo nombre de paquete. En nuestro proyecto no va a ser necesario. La firma SHA1, también es opcional. La necesitaremos para trabajar con autentificación con Google y teléfono, links dinámicos e Invites. Para obtenerla puedes seguir las instrucciones que se muestran al pulsar en el icono de interrogación. Otra alternativa es obtener esta firma directamente desde Android Studio. Si prefieres esta alternativa, sigue los siguientes 5 pasos:

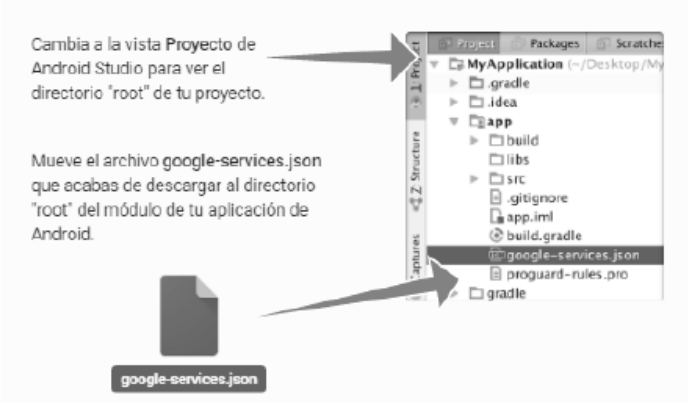


9. Desde Android Studio, pulsa en el botón **Gradle** del panel de la derecha.
10. Una vez abierta la ventana GRADLE, pulsa el botón **Refresh**.
11. En el desplegable abre la ruta <Nombre del proyecto> / Task / android.
12. Haz doble clic en **SigningReport**.
13. En la ventana RUN aparecerá la firma digital SHA1 (si es necesario, pulsa el botón). Cópiala al portapapeles, pégala en la consola de Firebase y pulsa en **Registrar app**.

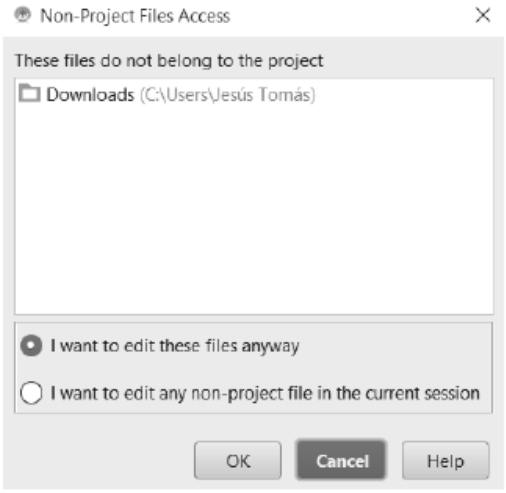
    NOTA: La firma que has pegado es de debug. Cuando firmes la aplicación para su publicación con la firma definitiva, tendrás que volver a la consola de Firebase y cambiar esta firma.

    NOTA: La firma de debug es diferente en cada ordenador donde hayas instalado Android Studio. Si cambias de ordenador, tendrás que volver a la consola de Firebase y cambiar esta firma.

14. El segundo paso te permite descargar el fichero google-services.json. Consiste en un archivo de configuración que hemos de copiar en la carpeta del módulo de nuestra aplicación. Este módulo suele llamarse app.



Si aparece una ventana similar a la siguiente pulsa en **OK**.



15. El tercer y último paso consiste en añadir algunas configuraciones en el Gradle del proyecto:

El complemento de los servicios de Google para Gradle carga el archivo `google-services.json` que acabas de descargar. Para poder usar el complemento, debes modificar los archivos build.gradle.

1. **build.gradle de proyecto** (`<project>/build.gradle`):

```
buildscript {
  dependencies {
    // Add this line
    classpath 'com.google.gms:google-services:3.1.0'
  }
}
```

2. **build.gradle de aplicación** (`<project>/<app-module>/build.gradle`):

```
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
```

incluye Analytics en la configuración predeterminada

3. Por último, presiona **Sincronizar ahora** en la barra que aparece en el entorno IDE:

Gradle files have changed sir | Sync now

16. En build.gradle (Project:Audiolibros) añade:

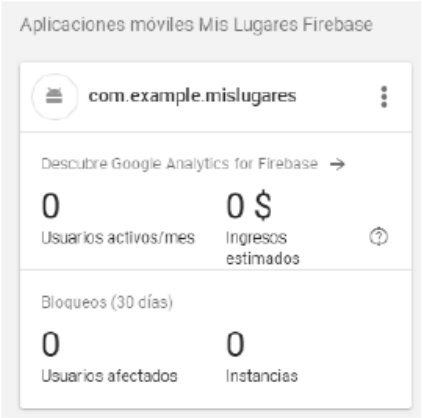
```
buildscript {
   …
   dependencies {
      …
      classpath 'com.google.gms:google-services:3.1.0'
   }
}
…
```

17. En build.gradle (Module:app) añade al final:

```
apply plugin: 'com.google.gms.google-services'
```

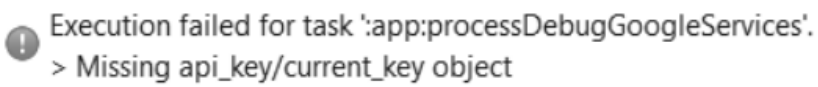
18. Cuando cambias los ficheros de Gradle has de sincronizar el proyecto. Te aparecerá un mensaje similar al siguiente. Pulsa en **Sync Now**.

19. En la consola de Firebase pulsa en el botón **Finalizar**. Aparecerá la siguiente información:



20. Ejecuta el proyecto para verificar que todo funciona correctamente.

NOTA: Si te aparece el siguiente error:

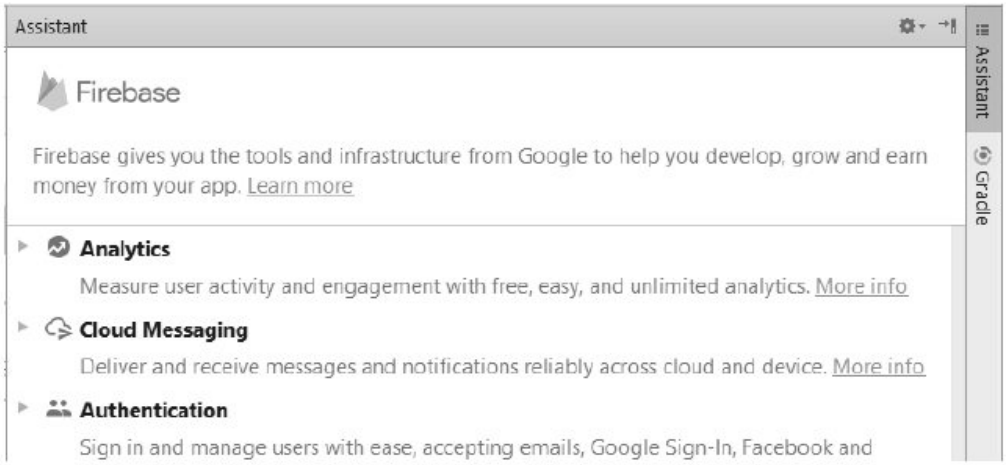


descarga de nuevo el fichero google-services.json. Para descargarlo pulsa en los tres puntos que aparecen junto a com.example.audiolibros de la captura anterior, selecciona **Configuración** y busca el nombre del fichero.

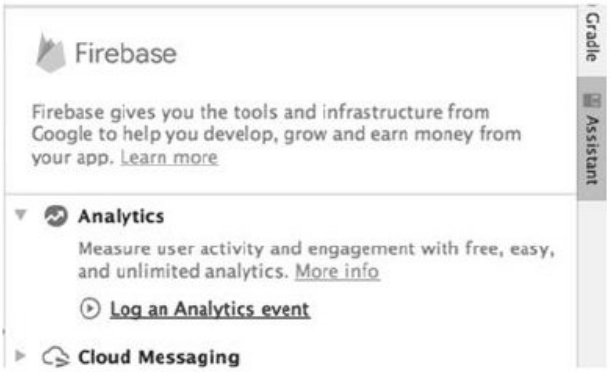
Ejercicio: Añadir Firebase a un proyecto a de forma automática

1. Accede a la consola de Firebase (https://console.firebase.google.com) y crea una cuenta, si no tienes una creada.

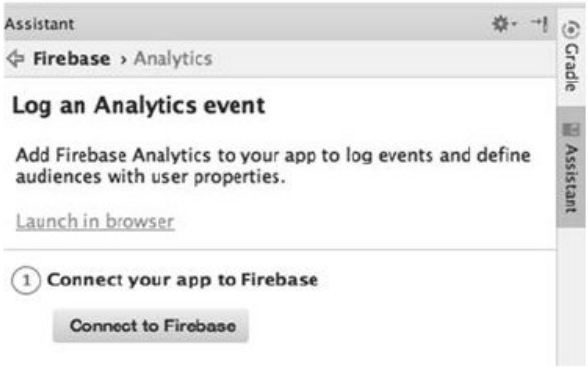
2. Abre o crea el proyecto al que quieras añadir Firebase.
3. Selecciona el menú de Tools / Firebase. Aparecerá un asistente que te permitirá integrar rápidamente la mayoría de servicios de Firebase:



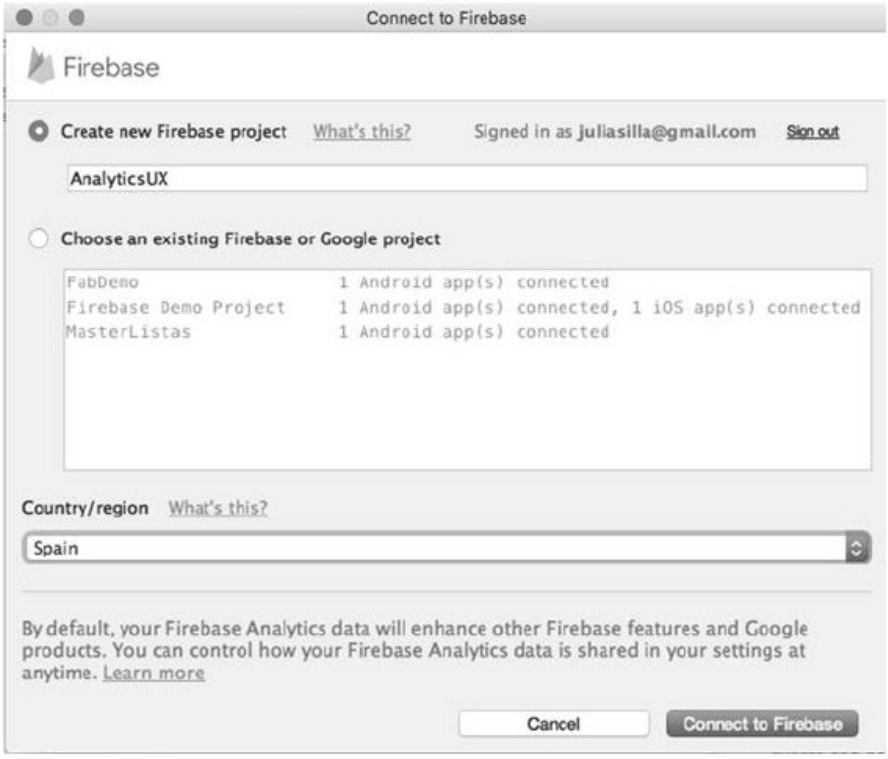
4. A modo de ejemplo probaremos el primero, Analytics. Selecciona Analytics / Log an Analytics event:



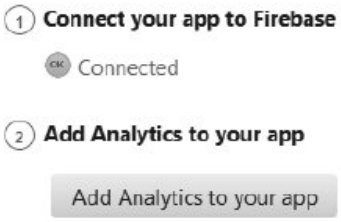
5. Verás que aparecen una serie de pasos que vamos a seguir. Pulsa el botón **Connect to Firebase** e indica tus credenciales:



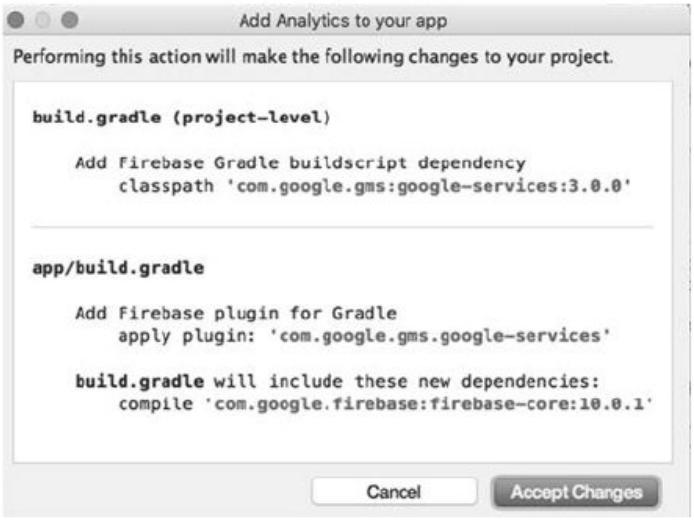
6. Se creará automáticamente un proyecto en Firebase. Acepta el nombre de la aplicación y la región.



7. Pasamos al segundo punto. Pulsa en **Add Analytics to your app**:



8. Acepta los cambios para añadir las dependencias automáticamente:



9. El siguiente paso de la lista es para generar eventos Log. De momento puedes saltarlo, ya que lo explicaremos más adelante.

10. Si accedes a la consola de Firebase, nuestro proyecto aparecerá en breve:

1.3. Autentificación con Firebase

Realizar un proceso de autentificación a través de Internet es extremadamente costoso. La trasferencia ha de realizarse de manera encriptada a través de https. El almacenamiento de las credenciales ha de cumplir todas las medidas de seguridad. Se ha de implementar un sistema de caché por si los usuarios pierden el acceso a Internet. Se necesita un backend, para gestionar los usuarios, envío de correos y la restauración de contraseñas. Además, resulta interesante implementar pasarelas para delegar la autentificación usando usuarios de redes sociales. Con Firebase no es necesario enfrentarse a todos estos problemas, ya que nos proporciona una solución integrada y totalmente automática.

Otro aspecto a destacar es que la responsabilidad de almacenar las credenciales es de Google. Por lo tanto, no hemos de preocuparnos de posibles fallos de seguridad en la custodia de estas credenciales.

Finalmente, hay que indicar que disponemos de gran variedad de sistemas de autentificación. Además de la autentificación con correo y contraseña, podemos autentificar utilizando el número de teléfono del usuario, realizar una autentificación personalizada, conectándonos a un sistema de autentificación propio, o usuario anónimo, muy interesante si el usuario no quiere darnos sus credenciales hasta estar seguro que va a utilizar la aplicación.

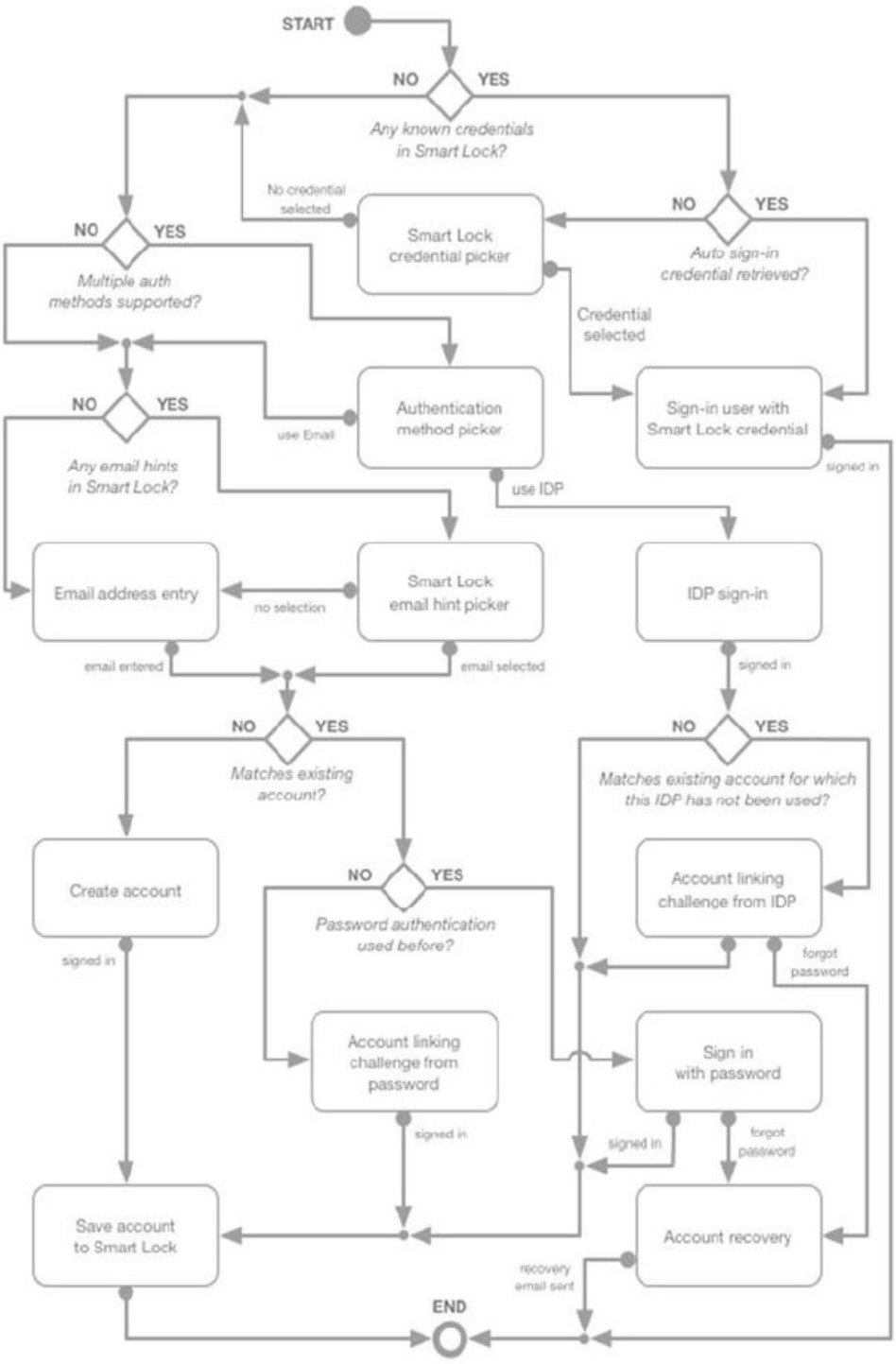
No obstante, la alternativa preferida por muchos usuarios es utilizar el mismo usuario que tienen en redes sociales, y así no tener que memorizar más contraseñas. Vamos a tener soporte para Google, Facebook, Twitter y GitHub.

1.3.1. Autentificación con FirebaseUI

FirebaseUI complementa al SDK, es una biblioteca de código abierto que incluye elementos de interfaz gráfica que podemos usar para realizar acciones frecuentes con autentificación, bases de datos o almacenamiento.

Resulta especialmente útil para autentificación. La librería se encarga de todo, incluso permite al usuario escoger el proveedor de autentificación. No obstante, cuando se presenten problemas va a resultar interesante saber cómo se realiza el proceso de autentificación. También cuando queramos una autentificación personalizada a nuestras necesidades.

Este proceso realizado por FirebaseUI se explica en el siguiente diagrama:

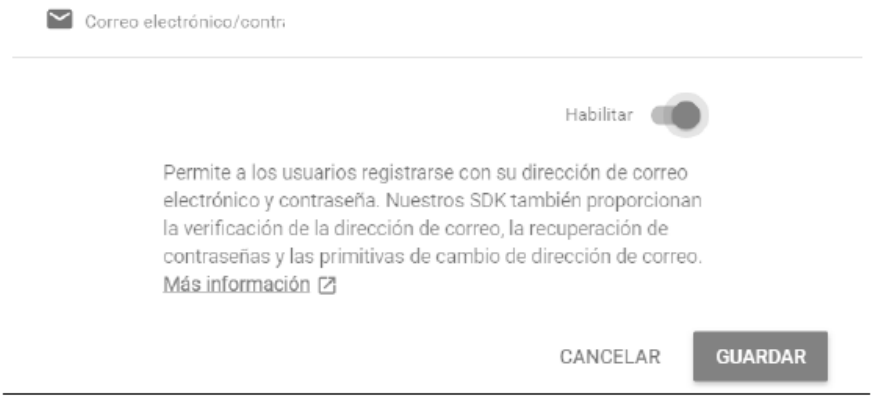


1.3.1.1. Autentificación por correo y Google

Vamos a realizar dos métodos de identificación en un mismo ejercicio, dado que el trabajo a realizar es similar. Si deseas solo uno de estos métodos de identificación, puedes anular el que no te interese.

Ejercicio: Autentificación por correo y Google

1. Accede a la consola de Firebase y en el proyecto Mis Lugares / Authentication / Método de acceso, selecciona Correo electrónico/contraseña, pulsa en **Habilitar** y luego en **Guardar**:



2. Habilita también como método de acceso las cuentas de Google. No es necesario que actives ninguna opción avanzada. Recuerda que es necesario haber introducido la firma SHA1 en los datos de la aplicación.
3. Añade en build.gradle (Project) la línea subrayada:

```
allprojects {
    repositories {
        jcenter()
        maven { url 'https://maven.google.com' }
    }
}
```

FirebaseUI necesita tener acceso a la librería Fabric al ser utilizada por Facebook y Twitter.

4. En build.gradle (Module: app) asegúrate de que la versión mínima es 16 o más e incluye la siguiente dependencia:

```
android {
   defaultConfig {
      minSdkVersion 16
      …
dependencies {
   implementation 'com.firebaseui:firebase-ui-auth:3.1.3'
}
```

5. Añade en AndroidManifest.xml las líneas subrayadas:

```
<manifest …>
   <application
      android:supportsRtl="true" …>
      …
```

```
      <activity android:name=".LoginActivity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      …
```

Para que la librería que acabamos de añadir funcione correctamente, tenemos que eliminar el atributo `suportRtl`. Además, registramos una nueva actividad (`LoginActivity`) que será lanzada al iniciar la aplicación.

6. Elimina el `<inter-filter>` de la actividad `MainActivity` que anteriormente se lanzaba al iniciar la aplicación.

   NOTA: En caso de tener una actividad de splash, es posible que quieras lanzar la actividad para el Login tras esta. En tal caso, modifica estos pasos para adaptarlos a tu aplicación.

7. Crea la clase `LoginActivity` con el siguiente código:

```
public class LoginActivity extends AppCompatActivity {
   private static final int RC_SIGN_IN = 123;

   @Override protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      login();
   }

   private void login() {
      FirebaseUser usuario = FirebaseAuth.getInstance().getCurrentUser();
      if (usuario != null) {
         Toast.makeText(this, "inicia sesión: " +
            usuario.getDisplayName()+" - "+ usuario.getEmail()+" - "+
            usuario.getProviders().get(0),Toast.LENGTH_LONG).show();
         Intent i = new Intent(this, MainActivity.class);
         i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                  | Intent.FLAG_ACTIVITY_NEW_TASK
                  | Intent.FLAG_ACTIVITY_CLEAR_TASK);
         startActivity(i);
      } else {
         startActivityForResult(AuthUI.getInstance()
          .createSignInIntentBuilder()
          .setAvailableProviders(Arrays.asList(
            new AuthUI.IdpConfig.Builder(AuthUI.EMAIL_PROVIDER).build(),
            new AuthUI.IdpConfig.Builder(AuthUI.GOOGLE_PROVIDER).build()))
          .setIsSmartLockEnabled(false)
          .build(), RC_SIGN_IN);
        }
    }
```

Lo primero que llama la atención de esta actividad es que no se llama al método `setContentView()`. La vista asociada a la actividad se creará solo si es necesario, además va a ser FirebaseUI quien cree la vista asociada a la

actividad. En el método `login()` verificamos si Firebase ya tiene un usuario validado utilizando `getCurrentUser()`. En caso afirmativo, mostramos en un `Toast` los datos del usuario y arrancamos la actividad `MainActivity`.

En caso de no existir el usuario, vas a lanzar una actividad para que el usuario seleccione el método de autentificación. El método `createSignInIntentBuilder()` va a crear la intención adecuada. Podemos configurarla con los proveedores que nos interese. En el ejemplo, correo electrónico y Google.

8. Añade el siguiente método:

```
@Override
public void onActivityResult(int requestCode, int resultCode,Intent data){
   super.onActivityResult(requestCode, resultCode, data);
   if (requestCode == RC_SIGN_IN) {
      if (resultCode == ResultCodes.OK) {
         login();
         finish();
      } else {
         IdpResponse response = IdpResponse.fromResultIntent(data);
         if (response == null) {
            Toast.makeText(this,"Cancelado",Toast.LENGTH_LONG).show();
            return;
         } else if (response.getErrorCode() == ErrorCodes.NO_NETWORK) {
            Toast.makeText(this,"Sin conexión a Internet",
                                                Toast.LENGTH_LONG).show();
            return;
         } else if (response.getErrorCode() == ErrorCodes.UNKNOWN_ERROR) {
            Toast.makeText(this,"Error desconocido",
                                                Toast.LENGTH_LONG).show();
            return;
         }
      }
   }
}
```

Este método será llamado.

9. Ejecuta la aplicación. El resultado ha de ser similar al siguiente:

10. Tras acceder con varios usuarios, entra en la consola de Firebase. En la pestaña **Usuarios** encontrarás la siguiente información:

11. Cuando en un mismo dispositivo entres por segunda vez ya no te pedirá las credenciales de acceso. Como todavía no hemos implementado la opción de logout, si quieres hacer más pruebas, ves a la consola de Firebase y elimina el usuario creado.

12. Cambia en el método `doLogin()` el valor de `setIsSmartLockEnabled(false)` a `true`. Verifica que el sistema recordará los usuarios que han accedido a Mis Lugares y sus contraseñas. Esto permite un cambio de usuario mucho más rápido.

Práctica: Verificación de correo

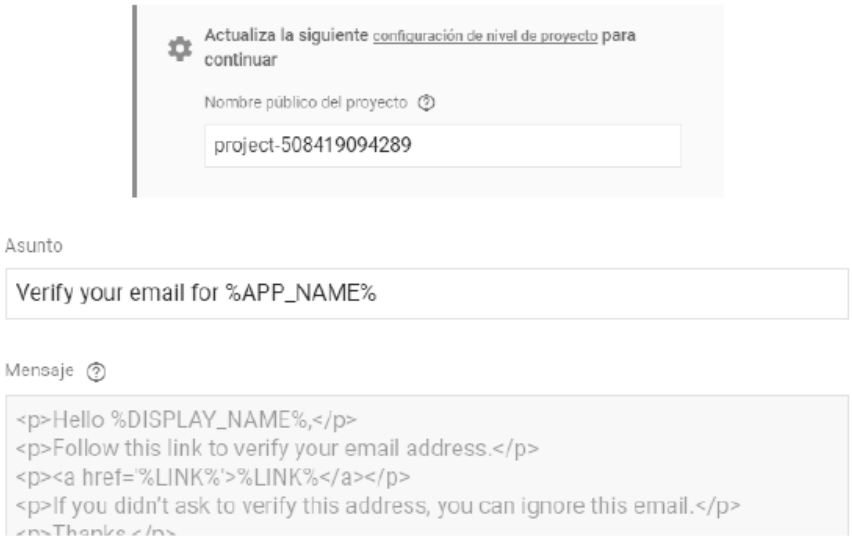
Como has podido observar, si creas una cuenta con correo y contraseña, no es necesario verificar si es válida. Esto significa que algunas personas podrán usar una dirección de correo que no sea suya o que ni siquiera exista. Para evitar esto, Firebase pone a nuestra disposición el envío de correos de confirmación. Esta técnica consiste en el envío de un correo a la dirección que indicó el usuario. El usuario ha de abrir este correo y pulsar en un link para validar su cuenta.

Mediante el método `FirebaseUser.isEmailVerified()` podemos averiguar si ha verificado su dirección de correo. Para que se le envíe un correo de verificación, tenemos el método `FirebaseUser.sendEmailVerification()`.

Con estos dos métodos, haz que la aplicación no deje entrar a los usuarios si no han verificado su cuenta de correo. Para ello, verifica en `login()`, antes de lanzar la actividad principal si el usuario ya está verificado, y en caso negativo envíale un correo de verificación y avísalo de que ha de abrir el correo. Realiza

estas acciones solo si el proveedor de autentificación es correo. Para el resto de proveedores, estas acciones no tienen sentido.

Accede a la consola de Firebase y selecciona Authentication / Plantillas / Verificación de dirección de correo electrónico. Modifica el nombre público del proyecto para que sea fácilmente identificable. Modifica el asunto de la plantilla para que el texto esté en castellano. Si tratas de modificar el cuerpo del mensaje, comprobarás que no te lo permite. Aparecerá el siguiente error: "Para evitar el envío de spam, no se puede editar el mensaje en esta plantilla de correo".



1.3.1.2. Obtener datos del usuario y cerrar sesión

Obtener los datos del perfil resulta muy sencillo:

```
FirebaseUser usuario = FirebaseAuth.getInstance().getCurrentUser();

String nombre = usuario.getDisplayName();
String correo = usuario.getEmail();
String telefono = usuario.getPhoneNumber();
Uri urlFoto = usuario.getPhotoUrl()
String uid = usuario.getUid();
String proveedor = usuario.getProviderId();
```

No siempre se dispone de toda la información, esto depende del proveedor de autentificación.

NOTA: Los métodos para acceder o modificar el perfil de usuario y cerrar sesión pertenecen al SDK de Firebase. Se han añadido dentro del apartado de FirebaseUI para secuenciar mejor el aprendizaje.

Ejercicio: Visualizar los datos del usuario

1. Añade el siguiente ítem de menú en res / menu / menu_main.xml:

```
<item android:id="@+id/menu_usuario"
    android:title="Perfil de Usuario"
    android:icon="@android:drawable/ic_menu_info_details"
    android:orderInCategory="1"
    app:showAsAction="always|withText"/>
```

2. En `MainActivity` añade dentro de `onOptionsItemSelected()` el siguiente código:

```
if (id == R.id.menu_usuario) {
   Intent intent = new Intent(this, UsuarioActivity.class);
   startActivity(intent);
}
```

3. Crea `UsuarioActivity` con el siguiente código:

```
public class UsuarioActivity  extends AppCompatActivity {
   @Override protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_usuario);
   }
}
```

4. Crea res/layout/activity_usuario.xml con el siguiente código:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <fragment
        android:id="@+id/usuario_fragment"
        android:name="com.example.mislugares.UsuarioFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:clickable="true"/>
</LinearLayout>
```

5. Crea `UsuarioFragment` con el siguiente código:

```
public class UsuarioFragment extends Fragment {
   @Override public View onCreateView(LayoutInflater inflador,
                        ViewGroup contenedor, Bundle savedInstanceState) {
      View vista = inflador.inflate(R.layout.fragment_usuario,
                         contenedor, false);
      FirebaseUser usuario = FirebaseAuth.getInstance().getCurrentUser();
      TextView nombre = (TextView) vista.findViewById(R.id.nombre);
      nombre.setText(usuario.getDisplayName());
      return vista;
   }
}
```

6. Crea res/layout/fragment_usuario.xml con el siguiente código:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/scrollView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal" >
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Nombre de usuario: " />
            <TextView
                android:id="@+id/nombre"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        </LinearLayout>
    </LinearLayout>
</ScrollView>
```

7. Registra la actividad que acabas de crear en AndroidManifest.xml:

```
<activity android:name=".UsuarioActivity"
          android:label="Información sobre el usuario" />
```

8. Verifica el resultado.
9. Visualiza en la actividad anterior otros datos del usuario utilizando los métodos `getEmail()`, `getProviders()`, `getPhoneNumber()` y `getUid()`. El método `getProviders()` ofrece una lista de proveedores, para visualizarlos, utiliza `toString()`:

```
TextView proveedores = (TextView) vista.findViewById(R.id.proveedores);
proveedores.setText(usuario.getProviders().toString());
```

El resultado ha de ser similar a:

Ejercicio: Cerrar sesión

1. Añade un botón para cerrar sesión en fragment_usuario.xml:

```
<Button
    android:id="@+id/btn_cerrar_sesion"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Cerrar sesión"
    android:padding="8dp"
    android:layout_marginRight="16dp"
    android:layout_marginTop="16dp"
    android:background="#d0021b"
    android:textColor="#fff"/>
```

2. En `UsuarioFragment`, añade en `onCreateView()` el siguiente código:

```
Button cerrarSesion =(Button) vista.findViewById(R.id.btn_cerrar_sesion);
cerrarSesion.setOnClickListener(new View.OnClickListener() {
   public void onClick(View view) {
      AuthUI.getInstance().signOut(getActivity())
         .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
               Intent i = new Intent(getActivity(),LoginActivity.class);
               i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                           | Intent.FLAG_ACTIVITY_NEW_TASK
                           | Intent.FLAG_ACTIVITY_CLEAR_TASK);
               startActivity(i);
               getActivity().finish();
            }
         });
   }
});
```

3. Verifica el resultado.

Ejercicio: Añadir fotografía de usuario

Las cuentas de Google, Facebook y Twitter suelen tener una fotografía de usuario asociada. En este ejercicio vamos a ver cómo podemos acceder a esta fotografía.

1. Añade la siguiente dependencia en build.gradle (Module: app):

```
compile 'com.android.volley:volley:1.0.0'
```

2. Añade en fragment_usuario.xml, al principio del `LinearLayout`, el código:

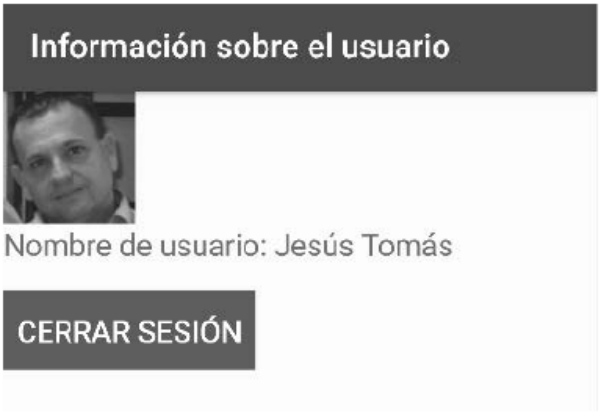
```
<com.android.volley.toolbox.NetworkImageView
    android:id="@+id/imagen"
    android:layout_width="120dp"
    android:layout_height="120dp"
    android:paddingTop="0dp"
    android:src="@android:drawable/sym_def_app_icon" />
```

Vamos a cargar la imagen a un tamaño fijo. Además, cargaremos la imagen de Internet usando un `NetworkImageView`.

3. En `UsuarioFragment`, añade en `onCreateView()`,el siguiente código:

```
// Inicialización Volley  (Hacer solo una vez en Singleton o Applicaction)
RequestQueue colaPeticiones = Volley.newRequestQueue(getActivity()
                                                .getApplicationContext());
ImageLoader lectorImagenes = new ImageLoader(colaPeticiones,
        new ImageLoader.ImageCache() {
            private final LruCache<String, Bitmap> cache =
                    new LruCache<String, Bitmap>(10);
            public void putBitmap(String url, Bitmap bitmap) {
                cache.put(url, bitmap);
            }
            public Bitmap getBitmap(String url) {
                return cache.get(url);
            }
        });

// Foto de usuario
Uri urlImagen = usuario.getPhotoUrl();
if (urlImagen != null) {
    NetworkImageView fotoUsuario = (NetworkImageView)
            vista.findViewById(R.id.imagen);
    fotoUsuario.setImageUrl(urlImagen.toString(), lectorImagenes);
}
```

4. Verifica que el resultado es similar al siguiente:



1.3.1.3. Métodos para cambiar el perfil de usuario

Cambiar la información de perfil de un usuario resulta sencillo. Mira este ejemplo:

```
FirebaseUser usuario = FirebaseAuth.getInstance().getCurrentUser();
UserProfileChangeRequest perfil = new UserProfileChangeRequest.Builder()
     .setDisplayName("Nuevo nombre usuario")
     .setPhotoUri(Uri.parse("https://www.ejemplo.com/usuario/foto.jpg"))
     .build();
usuario.updateProfile(perfil);
usuario.updateEmail("nuevo.correo@ejemplo.com");
```

```
usuario.updatePassword("nueva cotraseña")
        .addOnCompleteListener(new OnCompleteListener<Void>() {
           @Override
           public void onComplete(@NonNull Task<Void> task) {
              if (!task.isSuccessful()) {
                 Log.e("MisLugares", "Acción incorrecta");
              }
           }
        });
```

Los tres métodos `update` pueden añadir un escuchador `OnCompleteListener` para asegurarte que ha funcionado correctamente. Para mayor claridad, solo se ha añadido en el último.

Por razones de seguridad cambiar el correo, la contraseña o borrar la cuenta (`.delete()`) no pueden realizarse si el usuario no ha accedido recientemente. En caso contrario, se creará una `FirebaseAuthRecentLoginRequiredException`. Cuando esto suceda, debes volver a autentificar al usuario. Para ello, pide de nuevo al usuario las credenciales de acceso y pásalas a `reauthenticate`:

```
AuthCredential credential = EmailAuthProvider
        .getCredential("viejo.correo@ejemplo.com", "vieja contraseña");
usuario.reauthenticate(credential);
```

En este ejemplo se muestra cómo obtener las credenciales para correo y contraseña. A lo largo del capítulo se mostrará cómo hacerlo para otros proveedores. Es interesante que añadas el escuchador al método `reauthenticate` para informar al usuario si el cambio ha sido satisfactorio.

Desafío: Modificar datos de perfil

Utilizando estos métodos, modifica `UsuarioFragment` para permitir al usuario cambiar todos o algunos de los datos de su perfil. Puedes utilizar el planteamiento que prefieras. Puedes realizarlo para el proveedor correo y contraseña o también para otros proveedores. Es importante que informes al usuario si el cambio se ha realizado con éxito o incluso, en función del proveedor, si se va a poder realizar el cambio. Es recomendable realizar este desafío al final de la unidad. Se puede utilizar el esquema de verificación de formularios propuesto en el siguiente apartado.

1.3.1.4. Autentificación por Facebook y Twitter

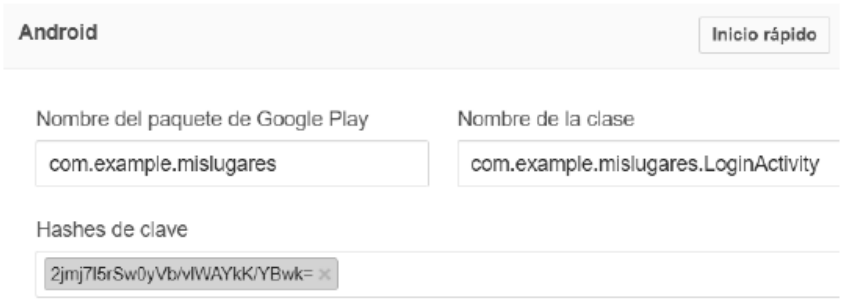
Ejercicio: Autentificación con Facebook

Para poder validar usuarios con Facebook en tu aplicación vas a tener que darla de alta en la consola de Facebook. Has de tener en cuenta que mientras tu aplicación esté en fase de desarrollo, no podrás probarla con cualquier usuario de Facebook. Solo podrás hacerlo con los que hayas dado de alta como evaluadores.

1. Entra en el sitio de Facebook para desarrolladores (https://developers.facebook.com/?locale=es_ES). Pulsa el en el botón **Entrar** e introduce o crea un usuario de Facebook.
2. Si no estás registrado como desarrollador de Facebook, pulsa en el botón **Regístrate**.
3. Pulsa en el botón **Crear aplicación** (si no aparece, usa Mis aplicaciones / Añadir una nueva aplicación) y rellena los siguientes datos:

4. Pulsa en **Crear identificador de la aplicación** y sigue las instrucciones.
5. Entrarás en la consola de la aplicación. En el menú de la izquierda, selecciona Configuración / Básica:

6. Pulsa en **Agregar plataforma**, selecciona **Android** e introduce los datos:



7. Para obtener la clave Hashes, utiliza en Windows el siguiente comando:

```
keytool -exportcert –alias androiddebugkey -keystore
"%HOMEPATH%\.android\debug.keystore" | openssl sha1 -binary | openssl base64
```

NOTA: El comando *keytool* lo encontrarás donde se instala la distribución de Java (ej. C:\Archivos de programa\Java\jre7\bin). El comando *openssl* es más complicado de encontrar. En http://www.dcomg.upv.es/~jtomas/Android/openssl.exe lo tienes.

NOTA: Si pide una contraseña has de introducir "android".

Y en iOS:

```
keytool -exportcert -alias androiddebugkey -keystore
~/.android/debug.keystore | openssl sha1 -binary | openssl base64
```

Otra alternativa que funciona en cualquier SO, sin instalar software, consiste en ejecutar desde una aplicación Android:

```
try {
    PackageInfo info = this.getPackageManager().getPackageInfo(
            "com.example.audiolibros", PackageManager.GET_SIGNATURES);
    for (Signature signature : info.signatures) {
        MessageDigest md = MessageDigest.getInstance("SHA");
        md.update(signature.toByteArray());
        Log.d("Audiolibros", "KeyHash:"+Base64.encodeToString(md.digest(),
                Base64.DEFAULT));
    }
} catch (PackageManager.NameNotFoundException e) {
} catch (NoSuchAlgorithmException e) {
}
```

El resultado se muestra en el log.

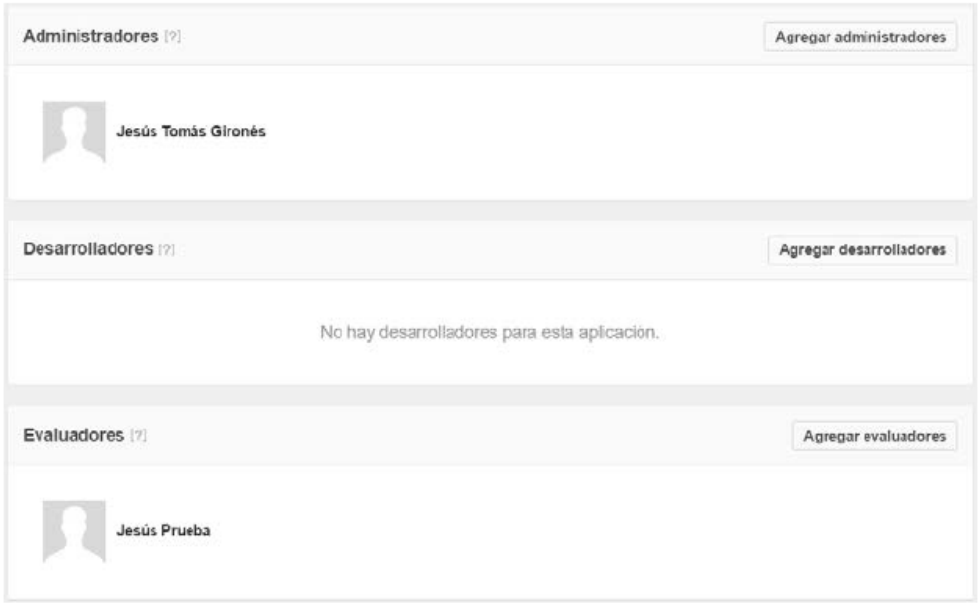
8. Pulsa el botón **Guardar cambios**. Dado que la aplicación no está publicada en Google Play, aparecerá la siguiente alerta:

Selecciona **Usar el nombre de este paquete**. La autentificación de usuario funcionará igualmente.

9. En otro navegador accede a la consola de Firebase. En la pestaña **Método de acceso**, habilita la autentificación con Facebook.

10. El ID de la app y el Secreto de app lo puedes obtener en la consola de Facebook Configuración / Básica (ver paso 5).

11. Copia la URI de redireccionamiento de OAuth y pulsa en **Guardar**.
12. Vuelve a la consola de Facebook y selecciona en el menú de la izquierda, pulsa en **Añadir producto**, y luego, **Inicio de sesión de Facebook**. En el campo URI de redireccionamiento de OAuth válido, pega la URI copiada.
13. En la consola de Facebook selecciona en el menú de la izquierda Roles. Pulsa el botón **Agregar evaluadores**. Indica un usuario para probar la aplicación que sea diferente al administrador, pero que tenga amistad con él.



14. En build.gradle (Module: app) añade la dependencia:

```
implementation 'com.facebook.android:facebook-login:[4,5)'
```

15. Añade en res/values/strings.xml las siguientes definiciones:

```
<string name="facebook_application_id"
        translatable="false">1741979532785150</string>
<string name="facebook_login_protocol_scheme"
        translatable="false">fb1741979532785150</string>
```

Reemplaza el primer valor por el identificador de la aplicación y el segundo por `fb` seguido de este mismo valor.

16. En la clase `LoginActivity`, método `login()`, añade la línea subrayada:

```
startActivityForResult(AuthUI.getInstance()
   .createSignInIntentBuilder().setProviders(Arrays.asList(
      new AuthUI.IdpConfig.Builder(AuthUI.EMAIL_PROVIDER).build(),
      new AuthUI.IdpConfig.Builder(AuthUI.GOOGLE_PROVIDER).build(),
      new AuthUI.IdpConfig.Builder(AuthUI.FACEBOOK_PROVIDER).build()))
```

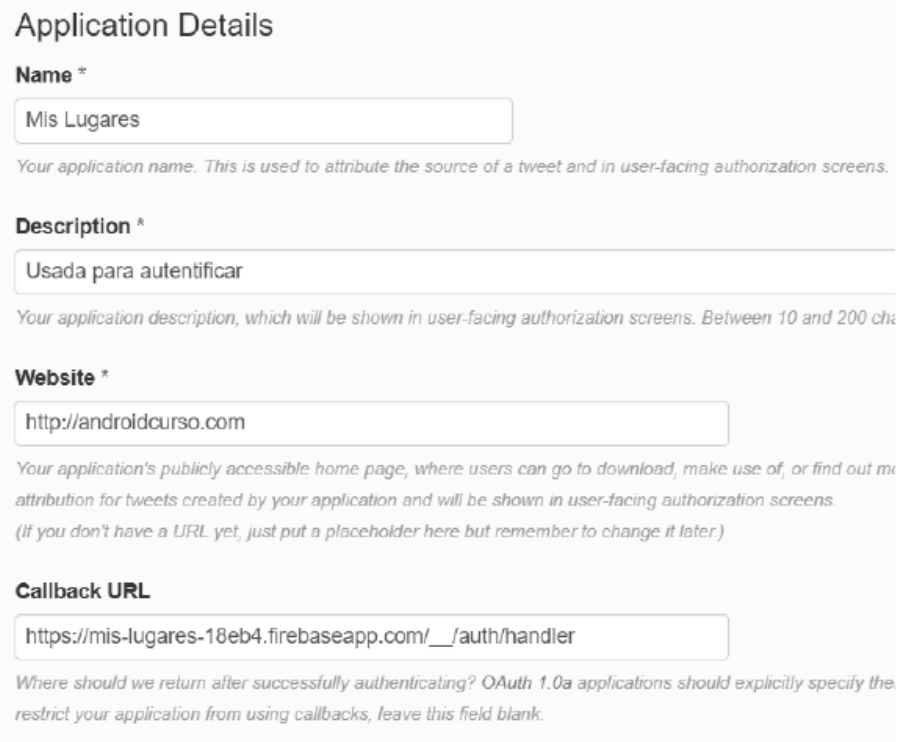
17. Ejecuta el proyecto y trata de autentificarte con Facebook.

Ejercicio: Autentificación con Twitter

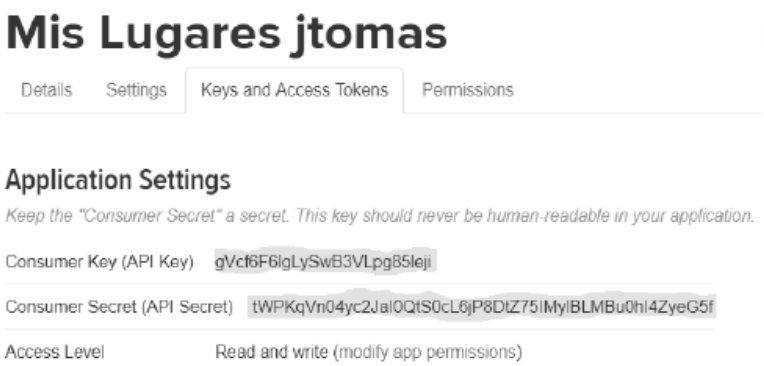
1. En la consola de Firebase, en Authentication / Proveedores, habilita Twitter:

2. Copia en el portapapeles el URL de devolución de llamada.
3. Entra en el sitio de Twitter para desarrolladores (https://apps.twitter.com) y regístrate como desarrollador.

4. Pulsa en el botón **Create New App** y rellena los datos. El nombre de la aplicación no puede coincidir con otros ya creados:



5. En Callback URL pega el contenido del portapapeles.
6. Tras aceptar, en la tercera pestaña, obtén la clave de API y el secreto:



7. Copia estos valores en la consola de Firebase (donde has encontrado el URL de callback).
8. En build.gradle (Module: app) añade la dependencia:

```
implementation 'com.twitter.sdk.android:twitter-core:3.1.1'
```

9. Copia también estos valores en res/values/strings.xml:

```
<string name="twitter_consumer_key"
      translatable="false">gVcf6F6lgLySwB3VLpg85leji</string>
<string name="twitter_consumer_secret" translatable=
      "false">tWPKqVn04yc2JaI0QtS0cL6jP8DtZ75IMylBLMBu0hI4ZyeG5f</string>
```

10. En la clase `LoginActivity`, método `login()`, añade la línea subrayada:

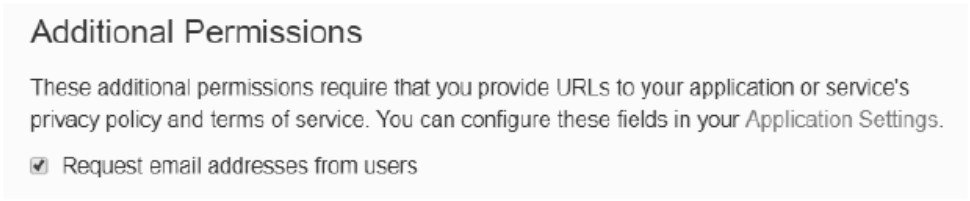
```
startActivityForResult(AuthUI.getInstance()
   .createSignInIntentBuilder().setProviders(Arrays.asList(
      …
      new AuthUI.IdpConfig.Builder(AuthUI.FACEBOOK_PROVIDER).build(),
      new AuthUI.IdpConfig.Builder(AuthUI.TWITTER_PROVIDER).build()))
```

11. Ejecuta el proyecto y trata de autentificarte con Twitter.

12. Por defecto, Twitter no comparte el correo electrónico.

Identificador	Proveedores	Creado	Accediste a tu cuenta	UID de usuario
jtomas00@gmail.com		16 ene. …	16 ene. …	kkt1XTYG3ETfm3E1wnQ…
–		16 ene. …	16 ene. …	nJDfk0gzqKgsTJQ20aO9…

Si lo necesitas en tu aplicación, puedes solicitarlo entrando en la consola de Twitter, pestaña **Permissions** y marcar la siguiente casilla:



Ejercicio: Uso de varios proveedores para un solo usuario

Cuando mostrábamos la información de usuario, hemos visto como un usuario puede estar validado por varios proveedores. En principio, puede parecer algo contradictorio. No obstante, cuando FirebaseUI observa que un usuario trata de registrase con un correo electrónico que ya ha sido utilizado con otro proveedor, no crea un nuevo usuario, sino que lo unifica con el proveedor ya existente. Solo existirá un usuario con un solo UID, pero este podrá utilizar indistintamente cualquiera de los proveedores:

Identificador	Proveedores	Creado	Accediste a tu cuenta	UID de usuario
jtomas00@gmail.com		13 en…	13 en…	LpeS8t46G6U8o0oo…

Para ver más claro cómo funciona vamos a hacer este ejercicio:

1. Accede a la aplicación utilizando como sistema de validación el correo electrónico. Utiliza un correo que coincida con una cuenta de Facebook.

2. Accede de nuevo utilizando validación por Facebook. Utiliza el mismo correo anterior.
3. En la consola la Firebase comprueba que se ha creado un solo usuario con dos proveedores.

1.3.1.5. Autentificación con número de teléfono

En algunas ocasiones puede ser interesante que el usuario utilice su número de teléfono como elemento de autentificación. A diferencia de otras medidas de acreditación, como un correo electrónico, no resulta costosa de crear y no es intransferible. Si un usuario solo dispone de un número de teléfono, solo podrá abrir una cuenta en nuestra aplicación.

Una vez introducido el número de teléfono, este es verificado mediante el envío de un SMS. Este SMS incluye un código númerico de verificación. En el caso de usar un teléfono inteligente, el proceso resulta muy cómodo, dado que el código es leído automáticamente por el sistema. Otra ventaja es que no es necesaria la introducción de una contraseña.

Este sistema de autentificación resulta menos seguro que otras alternativas, debido a que un número de teléfono es una información fácil de obtener. Además, en un teléfono con varios perfiles de usuario, cualquier usuario que reciba el SMS podrá acceder a la cuenta.

Otro inconveniente es que se trata de un servicio limitado. Tenemos 10 000 validaciones gratuitas, luego pueden haber costes.

Ejercicio: Autentificación con número de teléfono

1. Recuerda que para poder utilizar este servicio es imprescindible introducir la huella SHA1 del certificado digital de la aplicación, en la ficha de configuración de Firebase. Seguramente ya hayas realizado este paso.
2. Entra en la consola de Firebase y habilita la autentificación por teléfono:

3. En la clase `LoginActivity`, método `login()`, añade la línea subrayada:

```
.createSignInIntentBuilder().setProviders(Arrays.asList(
new AuthUI.IdpConfig.Builder(AuthUI.EMAIL_PROVIDER).build(),
… ,
new AuthUI.IdpConfig.Builder(AuthUI.PHONE_VERIFICATION_PROVIDER).build()))
```

4. Ejecuta el proyecto y autentifícate con número de teléfono.

5. Introduce tu número de teléfono. Se creará un código numérico que recibirás por SMS. La comprobación es muy cómoda, dado que en la mayoría de teléfonos se realiza de forma automática. En caso de querer utilizar el usuario en un dispositivo sin teléfono, no tienes más que copiar el código recibido en el SMS al dispositivo.

1.3.1.6. Personalización de FirebaseUI

Las opciones de configuración de FirebaseUI son limitadas[2]. No se permite crear nuestro propio layout y añadir los botones que nos ofrece FirebaseUI.

Se pueden configurar algunos aspectos como cambiar los textos, añadir un logo, colores, el fondo… Pero la apariencia final no cambiará demasiado. Por ejemplo, no podemos añadir nuevos botones para otros usos. A continuación, mostramos la apariencia actual y tras realizar el ejercicio de personalización:

[2] https://github.com/firebase/FirebaseUI-Android/tree/master/auth#ui-customization

Ejercicio: Personalizar la actividad inicial de FirebaseUI

1. En el método `login()` añade las líneas subrayadas:

```
startActivityForResult(AuthUI.getInstance()
        .createSignInIntentBuilder()
        .setLogo(R.mipmap.ic_launcher)
        .setTheme(R.style.FirebaseUITema)
        .setAvailableProviders(Arrays.asList(
             new AuthUI.IdpConfig.Builder(AuthUI.EMAIL_PROVIDER).build(),
```

2. En res/values/styles.xml añade:

```
<style name="FirebaseUITema" parent="FirebaseUI">
    <item name="windowNoTitle">false</item>
    <item name="windowActionBar">true</item>
    <item name="android:windowFullscreen">true</item>
    <item name="android:windowContentOverlay">@null</item>

    <item name="colorPrimary">#4CAF50</item>
    <item name="colorPrimaryDark">#388E3C</item>
    <item name="colorAccent">#AA00FF</item>
    <item name="colorControlNormal">#4CAF50</item>
    <item name="colorControlActivated">#AEEA00</item>
    <item name="colorControlHighlight">#69F0AE</item>
    <item name="android:windowBackground">@android:color/holo_green_light
         </item> <!-- o reemplaza por drawable-->
</style>
```

3. En res/values/strings.xml añade:

```
<string name="fui_default_toolbar_title">Tipo de Autentificación</string>
<string name="fui_sign_in_with_email">Entra con tu correo</string>
<string name="fui_progress_dialog_signing_up">Creando cuenta...</string>
```

La lista de las cadenas que puedes cambiar está en github.com[3].

NOTA: Según se muestra en la captura *"fui_sign_in_with_email"* no parece funcionar correctamente.

4. Ejecuta la aplicación y verifica el cambio de apariencia.

1.3.2. Autentificación personalizada con el SDK

En el apartado anterior, hemos visto cómo FirebaseUI nos permitía lanzar una actividad, en la que se preguntaba al usuario el método de autentificación y se completaba esta autentificación de forma automática. Es una alternativa muy sencilla de utilizar, pero también tiene sus inconvenientes: Las posibilidades de personalizar los

[3] https://github.com/firebase/FirebaseUI-Android/blob/master/auth/src/main/res/values/strings.xml

aspectos visuales de esta actividad son reducidas y tampoco podemos controlar el proceso de autentificación según nuestras necesidades.

En este apartado se propone un sistema alternativo. Vamos a realizar nuestro propio layout con textos y botones a nuestro gusto. Además, vamos a controlar cada uno de los métodos de autentificación usando de forma específica cada una de las librerías: Google, Facebook, Twitter…

1.3.2.1. Autentificación por correo y Google

Ejercicio: Autentificación personalizada con correo y contraseña

1. Define el layout activity_custom_login.xml con el siguiente código:

```
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/contenedor"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <android.support.constraint.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="16dp">
        <android.support.design.widget.TextInputLayout
            android:id="@+id/til_correo"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:minHeight="?android:attr/actionBarSize"
            android:transitionGroup="true"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintTop_toTopOf="parent">
            <EditText
                android:id="@+id/correo"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="Correo electrónico"
                android:imeOptions="actionNext"
                android:inputType="textEmailAddress"
                android:textSize="14dp" />
        </android.support.design.widget.TextInputLayout>
        <android.support.design.widget.TextInputLayout
            android:id="@+id/til_contraseña"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:minHeight="?android:attr/actionBarSize"
            android:transitionGroup="true"
            app:layout_constraintLeft_toLeftOf="parent"
```

```
            app:layout_constraintTop_toBottomOf="@+id/til_correo">
            <EditText
                android:id="@+id/contraseña"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="Contraseña"
                android:imeOptions="actionDone"
                android:inputType="textPassword"
                android:textSize="14dp" />
        </android.support.design.widget.TextInputLayout>
        <Button
            android:id="@+id/inicio_sesión"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:onClick="inicioSesiónCorreo"
            android:text="Iniciar Sesión"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toLeftOf="@+id/registro"
            app:layout_constraintTop_toBottomOf="@id/til_contraseña" />
        <Button
            android:id="@+id/registro"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:onClick="registroCorreo"
            android:text="Registro"
            app:layout_constraintLeft_toRightOf="@+id/inicio_sesión"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="@+id/inicio_sesión" />
        <Button
            android:id="@+id/firebase_ui"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="16dp"
            android:onClick="firebaseUI"
            android:text="Iniciar sesión con FirebaseUI"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/inicio_sesión" />
    </android.support.constraint.ConstraintLayout>
</ScrollView>
```

El resultado se muestra a continuación:

Observa cómo se utiliza un botón para inicio de sesión y otro para registro. El planteamiento de FirebaseUI es diferente. Se utiliza un único botón para el inicio de sesión y, en caso de que el correo electrónico no esté registrado, se solicita su registro. Se ha añadido un tercer botón con el que podrás acceder a la actividad creada en ejercicios anteriores para iniciar sesión usando FirebaseUI. En una aplicación real no tendría ningún sentido. Se ha introducido para que puedas testar ambas alternativas en la misma aplicación.

2. Crea la clase `CustomLoginActivity` con el siguiente código:

```
public class CustomLoginActivity extends Activity {
   private FirebaseAuth auth = FirebaseAuth.getInstance();
   private String correo = "";
   private String contraseña = "";
   private ViewGroup contenedor;
   private EditText etCorreo, etContraseña;
   private TextInputLayout tilCorreo, tilContraseña;
   private ProgressDialog dialogo;

   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_custom_login);
      etCorreo = (EditText) findViewById(R.id.correo);
      etContraseña = (EditText) findViewById(R.id.contraseña);
      tilCorreo = (TextInputLayout) findViewById(R.id.til_correo);
      tilContraseña = (TextInputLayout) findViewById(R.id.til_contraseña);
      contenedor = (ViewGroup) findViewById(R.id.contenedor);
      dialogo = new ProgressDialog(this);
      dialogo.setTitle("Verificando usuario");
      dialogo.setMessage("Por favor espere...");
      verificaSiUsuarioValidado();
   }

   private void verificaSiUsuarioValidado() {
      if (auth.getCurrentUser() != null) {
         Intent i = new Intent(this, MainActivity.class);
         i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                 | Intent.FLAG_ACTIVITY_NEW_TASK
                 | Intent.FLAG_ACTIVITY_CLEAR_TASK);
         startActivity(i);
         finish();
      }
   }
…
```

Comenzamos declarando una serie de variables globales a la clase. El más importante es `auth`, con el que podremos acceder a los servicios de autentificación de Firebase. Las variables `correo` y `contraseña` son las introducidas por el usuario para hacer la autentificación. El resto son elementos del interfaz de usuario.

El método `onCreate()` se limita a inicializar el interfaz de usuario, hasta la última línea, que llama a `verificaSiUsuarioValido()`. Aquí se comprueba si ya se dispone de un usuario, que seguramente se validó la última vez que se utilizó la aplicación. En caso afirmativo, se lanza `MainActivity`.

3. Añade los métodos asociados a los botones **Inicio de sesión** y **Registro**:

```
public void inicioSesiónCorreo(View v) {
   if (verificaCampos()) {
      dialogo.show();
      auth.signInWithEmailAndPassword(correo, contraseña)
       .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
             @Override
             public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()) {
                   verificaSiUsuarioValidado();
                } else {
                   dialogo.dismiss();
                   mensaje(task.getException().getLocalizedMessage());
                }
             }
      });
   }
}

public void registroCorreo(View v) {
   if (verificaCampos()) {
      dialogo.show();
      auth.createUserWithEmailAndPassword(correo, contraseña)
       .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
             @Override
             public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()) {
                   verificaSiUsuarioValidado();
                } else {
                   dialogo.dismiss();
                   mensaje(task.getException().getLocalizedMessage());
                }
             }
      });
   }
}
```

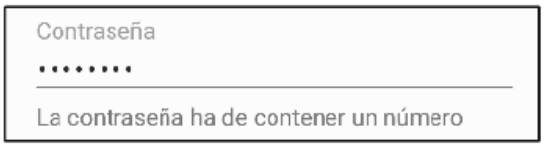
Los dos métodos tienen una estructura similar. Comienzan llamando a `verificaCampos()`, donde se verifica que tanto correo como contraseña son adecuados y se actualizan las variables globales `correo` y `contraseña`. El siguiente paso es mostrar un cuadro de diálogo para indicar al usuario que se está consultando con el servidor. Esto es interesante, dado que la siguiente llamada se hace a un método asíncrono. No nos interesa que el usuario interaccione con el interfaz de usuario hasta obtener una respuesta. Las llamadas para inicio de sesión y registro son similares. También la obtención de la respuesta, que se hace a través de la interface `OnCompleteListener`. En caso de una autentificación satisfactoria se llama a `verificaSiUsuarioValido`

para pasar a la siguiente actividad. En caso contrario, se muestra la causa del error.

4. Añade los siguientes métodos, para completar la clase:

```
   private void mensaje(String mensaje) {
      Snackbar.make(contenedor, mensaje, Snackbar.LENGTH_LONG).show();
   }

   private boolean verificaCampos() {
      correo = etCorreo.getText().toString();
      contraseña = etContraseña.getText().toString();
      tilCorreo.setError(""); tilContraseña.setError("");
      if (correo.isEmpty()) {
         tilCorreo.setError("Introduce un correo");
      } else if (!correo.matches(".+@.+[.].+")) {
         tilCorreo.setError("Correo no válido");
      } else if (contraseña.isEmpty()) {
         tilContraseña.setError("Introduce una contraseña");
      } else if (contraseña.length()<6) {
         tilContraseña.setError("Ha de contener al menos 6 caracteres");
      } else if (!contraseña.matches(".*[0-9].*")) {
         tilContraseña.setError("Ha de contener un número");
      } else if (!contraseña.matches(".*[A-Z].*")) {
         tilContraseña.setError("Ha de contener una letra mayúscula");
      } else {
         return true;
      }
      return false;
   }

   public void firebaseUI(View v) {
      startActivity(new Intent(this, LoginActivity.class));
   }
}
```

El método más interesante es `verificaCampos()`, donde se obtienen las variables `correo` y `contraseña` y se verifica si cumplen ciertas reglas. Para mostrar los errores de forma muy adecuada para la experiencia de usuario se utiliza `TextInputLayout`.



Otro aspecto a destacar es el uso de expresiones regulares para implementar estas reglas. El método `matches()` verifica si el string cumple una expresión regular. Por ejemplo, la expresión regular `".+@.+[.].+"` significa una secuencia de uno o más caracteres, seguido de `@`, y a continuación, uno o más caracteres, seguida de `.` y uno o más caracteres. Y `".*[0-9].*"` significa una secuencia de cero o más caracteres, seguida de un dígito, y a continuación, cero o más caracteres.

5. Edita AndroidManifest para que nuestra aplicación arranque como primera actividad `CustomLoginActivity`.
6. Arranca la aplicación y verifica que el método de autentificación que acabamos de definir funciona correctamente.
7. Si cierras sesión, comprobarás que no se vuelve a esta actividad. Para solucionarlo, añade el código subrayado en `UsuarioFragment`:

```
Intent i = new Intent(getActivity(), CustomLoginActivity.class);
```

Ejercicio: Autentificación personalizada con Google

1. Verifica que está habilitado el método de acceso de Google y que se ha introducido la huella SHA1 del certificado con el que se firma la aplicación.
2. En el layout de la actividad añade un botón con título **Iniciar sesión con Google**. Ha de llamar al método `autentificarGoogle`.
3. En `CustomLoginActivity` añade las siguientes declaraciones:

```
private static final int RC_GOOGLE_SIGN_IN = 123;
private GoogleApiClient googleApiClient;
```

4. Añade las siguientes líneas en el `onCreate()`, que configuran las opciones de autentificación de Google.:

```
//Google
GoogleSignInOptions gso = new GoogleSignInOptions.Builder(
    GoogleSignInOptions.DEFAULT_SIGN_IN)
    .requestIdToken(getString(R.string.default_web_client_id))
    .requestEmail()
    .build();
googleApiClient = new GoogleApiClient.Builder(this)
    .enableAutoManage(this , this)
    .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
    .build();
```

5. Cuando se pulse el botón insertado, llamaremos al siguiente método:

```
public void autentificarGoogle(View v) {
   Intent i = Auth.GoogleSignInApi.getSignInIntent(googleApiClient);
   startActivityForResult(i, RC_GOOGLE_SIGN_IN);
}
```

   Simplemente se lanzará una actividad, a partir de un `Intent` generado mediante la librería de Google.

6. La respuesta nos volverá a `onActivityResult()`. Las siguientes líneas detectan si se ha recibido y le pasa el resultado a `googleAuth()`:

```
@Override public void onActivityResult(int requestCode, int resultCode,
                                       Intent data) {
```

```
   super.onActivityResult(requestCode, resultCode, data);
   if (requestCode == RC_GOOGLE_SIGN_IN) {
      if (resultCode == RESULT_OK ) {
         GoogleSignInResult result = Auth.GoogleSignInApi
                 .getSignInResultFromIntent(data);
         if (result.isSuccess()) {
            googleAuth(result.getSignInAccount());
         } else {
            mensaje("Error de autentificación con Google");
         }
      }
   }
}
```

7. El método `googleAuth()` simplemente le pasa este valor a Firebase:

```
private void googleAuth(GoogleSignInAccount acct) {
   AuthCredential credential = GoogleAuthProvider.getCredential(
       acct.getIdToken(), null);
   auth.signInWithCredential(credential)
       .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
           @Override
           public void onComplete(@NonNull Task<AuthResult> task) {
              if (!task.isSuccessful()) {
                 mensaje(task.getException().getLocalizedMessage());
              }else{
                 verificaSiUsuarioValidado();
              }
           }
        });
}
```

8. Finalmente, necesitamos que nuestra clase implemente la interfaz `GoogleApiClient.OnConnectionFailedListener`. Cambia también la base de la clase de `Activity` a `FragmentActivity`. Para implementar esta interfaz añade el siguiente método:

```
@Override
public void onConnectionFailed(@NonNull ConnectionResult result) {
   mensaje("Error de autentificación con Google");
}
```

9. Verifica el resultado.

1.3.2.2. Autentificación por Facebook y Twitter

Ejercicio: Autentificación personalizada con Facebook

1. Comprueba que has realizado toda la configuración previa que se indica en el ejercicio Autentificación con Facebook.

2. Añade el siguiente botón antes de **Iniciar sesión con FirebaseUI**:

```
<com.facebook.login.widget.LoginButton
    android:id="@+id/facebook"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/google"/>
```

3. Añade las siguientes declaraciones:

```
private CallbackManager callbackManager;
private LoginButton btnFacebook;
```

4. Es necesario inicializar el SDK de Facebook antes de mostrar el botón. Añade en `onCreate()`, antes del `setContentView()` la línea subrayada. Añadir antes de `verificaSiUsuarioValidado()` el resto del código:

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   FacebookSdk.sdkInitialize(this);
   setContentView(R.layout.activity_custom_login);
   …
   //Facebook
   callbackManager = CallbackManager.Factory.create();
   btnFacebook = (LoginButton) findViewById(R.id.facebook);
   btnFacebook.setReadPermissions("email", "public_profile");
   btnFacebook.registerCallback(callbackManager,
        new FacebookCallback<LoginResult>() {
           @Override  public void onSuccess(LoginResult loginResult) {
              facebookAuth(loginResult.getAccessToken());
           }
           @Override public void onCancel() {
              mensaje("Cancelada autentificación con facebook");
           }
           @Override public void onError(FacebookException error) {
              mensaje(error.getLocalizedMessage());
           }
        });
   verificaSiUsuarioValidado();
}
```

5. Cuando se pulse sobre el botón que acabamos de definir, el API de Facebook lanzará una actividad que realizará la autentificación. Al volver de esta actividad podrás analizar el resultado si añades a `onActivityResult()` el siguiente código:

```
else if (requestCode == btnFacebook.getRequestCode()) {
    callbackManager.onActivityResult(requestCode, resultCode, data);
}
```

Al invocar el método `onActivityResult()` del botón, estamos consiguiendo que se llame al `Callback` que hemos definido anteriormente.

6. En caso de un resultado positivo, el `Callback` llamará al siguiente método:

```
private void facebookAuth(AccessToken accessToken) {
   final AuthCredential credential = FacebookAuthProvider.getCredential(
       accessToken.getToken());
   auth.signInWithCredential(credential)
       .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
              @Override
              public void onComplete(@NonNull Task<AuthResult> task) {
                 if (!task.isSuccessful()) {
                    if (task.getException() instanceof
                           FirebaseAuthUserCollisionException) {
                       LoginManager.getInstance().logOut();
                    }
                    mensaje(task.getException().getLocalizedMessage());
                 } else {
                    verificaSiUsuarioValidado();
                 }
              }
       });
}
```

donde se extraen los datos de autentificación y se envían a Facebook. Si la autentificación es positiva, se llama de nuevo a `verificaSiUsuarioValidado`, desde donde se abrirá `MainActivity`.

7. Verifica el resultado.

Ejercicio: Autentificación personalizada con Twitter

1. Comprueba que has realizado toda la configuración previa que se indica en el ejercicio Autentificación con Twitter.
2. Añade el siguiente botón antes de **Iniciar sesión con FirebaseUI**:

```
<com.twitter.sdk.android.core.identity.TwitterLoginButton
    android:id="@+id/twitter"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/facebook"/>
```

3. Añade la siguiente declaración:

```
private TwitterLoginButton btnTwitter;
```

4. Añade en `onCreate()` el siguiente código, que se encargará de pasarle un callback al botón de Twitter para que te avise cuando el usuario se haya autentificado.

5. Es necesario inicializar el SDK de Twitter antes de mostrar el botón. Añade en `onCreate()`, antes del `setContentView()`, las líneas subrayadas. Añadir el resto del código antes de `verificaSiUsuarioValidado()`:

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   …
   Twitter.initialize(new TwitterConfig.Builder(this)
       .logger(new DefaultLogger(Log.DEBUG))
       .twitterAuthConfig(new TwitterAuthConfig(getString(R.string.
   twitter_consumer_key), getString(R.string.twitter_consumer_secret)))
       .debug(true)
       .build());
   setContentView(R.layout.activity_custom_login);
   …
   //Twitter
   btnTwitter = (TwitterLoginButton) findViewById(R.id.twitter);
   btnTwitter.setCallback(new Callback<TwitterSession>() {
      @Override public void success(Result<TwitterSession> result) {
         twitterAuth(result.data);
      }
      @Override public void failure(TwitterException exception) {
         mensaje(exception.getLocalizedMessage());
      }
   });
   verificaSiUsuarioValidado();
}
```

6. Cuando se pulse sobre el botón que acabamos de definir, el API de Twitter lanzará una actividad que realizará la autentificación. Al volver de esta actividad podrás analizar el resultado si añades a `onActivityResult()` el siguiente código:

```
else if (requestCode == TwitterAuthConfig.DEFAULT_AUTH_REQUEST_CODE) {
    btnTwitter.onActivityResult(requestCode, resultCode, data);
}
```

Al invocar el método `onActivityResult()` del botón, estamos consiguiendo que se llame al `Callback` que hemos definido en el punto 1.

7. En caso de un resultado positivo, el `Callback` llama al siguiente método:

```
private void twitterAuth(TwitterSession session) {
   AuthCredential credential = TwitterAuthProvider.getCredential(
       session.getAuthToken().token,
       session.getAuthToken().secret);
   auth.signInWithCredential(credential)
      .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
        @Override public void onComplete(@NonNull Task<AuthResult> task) {
           if (!task.isSuccessful()) {
              mensaje(task.getException().getLocalizedMessage());
            } else {
               verificaSiUsuarioValidado();
            }
        }
```

```
        });
}
```

donde se extraen los datos de autentificación y se envían a Twitter. Si la autentificación es positiva, se llama de nuevo a `doLogin()`.

8. Verifica el resultado.

1.3.2.3. Autentificación anónima y unificación de cuentas

Muchos usuarios son reacios a introducir sus credenciales en una aplicación que simplemente quieren probar. Para evitar la pérdida de estos usuarios puede ser interesante dejar que prueben nuestra aplicación de forma anónima.

En otras ocasiones no necesitamos una autentificación, pero sí que nos interesa que nuestra aplicación registre todas las peticiones de un mismo dispositivo como pertenecientes a una misma sesión.

Para resolver cualquiera de estas dos circunstancias, Firebase nos ofrece el concepto de usuario anónimo.

Ejercicio: Autentificación con usuario Anónimo

1. Accede a la consola de Firebase y activa la autentificación anónima:

2. En el layout de la actividad añade un botón con título **Entrar con usuario anónimo**. Ha de llamar al método `autentificaciónAnónima`.

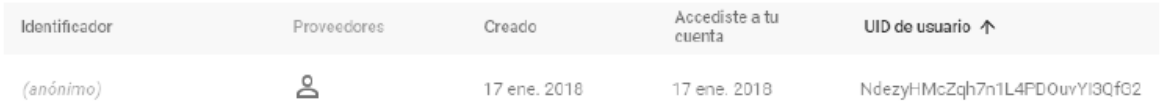
3. Añade el método para este botón:

```
public void autentificaciónAnónima(View v) {
   dialogo.show();
   auth.signInAnonymously()
      .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
         @Override public void onComplete(@NonNull Task<AuthResult> task){
            if (task.isSuccessful()) {
               verificaSiUsuarioValidado();
            } else {
               dialogo.dismiss();
               Log.w("MisLugares", "Error en signInAnonymously",
                                                   task.getException());
               mensaje( "ERROR al intentarentrar de forma anónima");
            }
         }
      });
}
```

Observa que el proceso es muy similar al del resto de métodos. La diferencia es que el usuario es creado sin que se solicite al usuario las credenciales.

4. Verifica el resultado. Si seleccionas la opción Detalles de usuario, verás que todos los datos están vacíos, menos el ID.

5. En la consola de Firebase verifica que se ha creado un nuevo usuario.



Una vez que un usuario anónimo ya ha probado tu app, tal vez sea conveniente permitirle que continúe su trabajo con una cuenta permanente. Por ejemplo, una vez que ha seleccionado varios elementos en el carrito y va a realizar la compra, puede ser buena idea solicitarle que cree una cuenta permanente. La nueva cuenta tendrá el mismo UID, por lo que, si asociaste esta compra al UID, no tendrás que hacer cambios en la base de datos.

El siguiente ejercicio nos va a permitir realizar este proceso, pero además permitir mucho más. Vamos a poder unificar una cuenta con una cuenta de otro proveedor.

Ejercicio: Unificar la cuenta actual con otra

1. En el layout fragment_usuario.xml añade un botón con título Unificar con otra cuenta.

2. En `onCreate` de `UsuarioFragment` añade el siguiente código:

```
Button unirCuenta = (Button) vista.findViewById(R.id.btn_unir_cuenta);
unirCuenta.setOnClickListener(new View.OnClickListener() {
   public void onClick(View view) {
       Intent i = new Intent(getActivity(),CustomLoginActivity.class);
       i.putExtra("unificar",true);
       i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                | Intent.FLAG_ACTIVITY_NEW_TASK
                | Intent.FLAG_ACTIVITY_CLEAR_TASK);
       startActivity(i);
   }
});
```

3. Observa como nos limitamos a abrir la actividad de autentificación, pero pasándole el extra `unificar` y sin cerrar la sesión actual.

4. En `CustomLoginActivity` añade la siguiente variable;

```
private boolean unificar;
```

5. Inicialízala en `onCreate()`:

```
unificar = getIntent().getBooleanExtra("unificar", false);
```

En caso de una llamada desde el inicio de la aplicación, el extra no estará definido, por lo que la variable se iniciará a `false`.

6. Añade el código subrayado:

```
private void verificaSiUsuarioValidado() {
   if (!unificar && auth.getCurrentUser() != null) {
     …
   }
}
```

Como tenemos un usuario ya validado, si no introducimos esta comprobación, saldríamos de la actividad nada más entrar.

7. Añade el siguiente código:

```
private void googleAuth(GoogleSignInAccount acct) {
   AuthCredential credential = GoogleAuthProvider.getCredential(
           acct.getIdToken(), null);
   if (unificar) {
      unificarCon(credential);
   } else {
      auth.signInWithCredential(credential)
      …
   }
}
```

Cuando el usuario pide una autentificación por Google, acabamos llegando a este método. En caso de que estemos usando un usuario unificar, en lugar de llamar a `signInWithCredential()` para pedir que nos autentifiquen, vamos a llamar a `unificarCon()`, donde uniremos la cuenta actual con las credenciales de Google.

8. Añade el siguiente método:

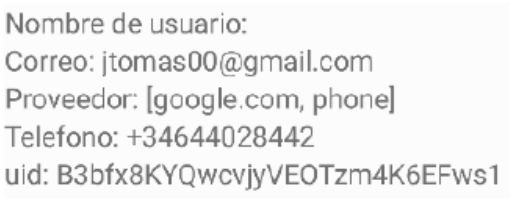
```
private void unificarCon(AuthCredential credential) {
   auth.getCurrentUser().linkWithCredential(credential)
       .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
          @Override
          public void onComplete(@NonNull Task<AuthResult> task) {
             if (task.isSuccessful()) {
                unificar = false;
                verificaSiUsuarioValidado();
             } else {
                Log.w("MisLugares", "Error en linkWithCredential",
                                                    task.getException());
                mensaje("Error al unificar cuentas.");
              }
          }
       });
}
```

El trabajo lo hace `linkWithCredential()`, que une la cuenta actual con las credenciales que le hemos pasado. El resto del código recoge el resultado. Si este es correcto, llamamos a `verificaSiUsuarioValidado()`, pero antes

indicamos que ya no somos un usuario a unificar. En caso de error, lo mostramos.

9. Ejecuta el proyecto. Para probarlo entra primero con un usuario anónimo, apunta el UID asignado. Pulsa el botón **Unificar cuenta con otra** y selecciona una cuenta de Google que no esté ya registrada en el sistema. Verifica que el UID del nuevo usuario coincide con el anterior.

10. Prueba ahora a unificar una cuenta de teléfono con una de Google. Entra primero con un número de teléfono, apunta el UID asignado. Pulsa el botón **Unificar cuenta con otra** y selecciona una cuenta de Google que no esté ya registrada en el sistema. Verifica que el UID del nuevo usuario coincide con el anterior. En este caso estarás validado por dos proveedores y podrás obtener tanto información del correo como del teléfono:



11. Observa cómo ambas cuentas aparecen unificadas en la consola:

12. Cierra la sesión y autentifícate con la cuenta de teléfono o la de Google. Comprueba cómo en ambos casos obtienes tanto el número teléfono como el correo.

Práctica: Convertir cuenta anónima en cualquier permanente

El ejercicio anterior solo funciona si unificas la cuenta actual a una de Google. Consigue que este cambio funcione unificando con otros tipos de cuentas. Con Facebook y Twitter es muy similar. Con correo tendrás que obtener primero las credenciales. Puedes usar el siguiente código:

```
AuthCredential credential=EmailAuthProvider.getCredential(correo,contraseña);
```

Incluso puedes unificar con una tercera cuenta, siempre que el correo coincida:

1.3.2.4. Recuperación de contraseña

Ejercicio: Mandar correo de recuperación de contraseña

1. En el layout de la actividad añade un botón que llame al siguiente método:

```
public void reestablecerContraseña(View v) {
   correo = etCorreo.getText().toString();
   tilCorreo.setError("");
   if (correo.isEmpty()) {
      tilCorreo.setError("Introduce un correo");
   } else if (!correo.matches(".+@.+[.].+")) {
      tilCorreo.setError("Correo no válido");
   } else {
      dialogo.show();
      auth.sendPasswordResetEmail(correo)
         .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override public void onComplete(@NonNull Task<Void> task) {
               dialogo.dismiss();
               if (task.isSuccessful()) {
                  mensaje("Verifica tu correo para cambiar contraseña.");
               } else {
                  mensaje("ERROR al mandar correo para cambiar contraseña");
               }
            }
         });
   }
}
```

   Tras verificar que el correo es válido, se llama a `sendPasswordResetEmail()`. Según el resultado se muestra el mensaje correspondiente.

2. Verifica el resultado.

3. Entra en la consola de Google entra en Authentication / Plantillas / Restablecimiento de contraseña. En la parte inferior izquierda, selecciona como idioma el español. Personaliza la plantilla para el envío del correo.

CAPÍTULO 2.

Bases de datos

JESÚS TOMÁS

La base de datos puede considerarse la piedra angular de una aplicación. El almacenamiento y acceso a los datos suele ser la clave de que todo funciona de forma fluida y eficiente. Si la base de datos se sitúa en la nube, nos permite que los usuarios compartan la información o que un usuario acceda a sus datos desde cualquier dispositivo.

Firebase nos ofrece una herramienta de bases de datos de una gran potencia y flexibilidad. Es de tipo NoSQL, lo que le permite un crecimiento en tamaño y número de accesos casi ilimitado. Además, es en tiempo real;estova a permitir que nuestro programa reaccione de forma automática ante cualquier cambio en los datos que nos interesen.

Empezaremos el capítulo describiendo cómo trabajar con este nuevo tipo de bases de datos. Luego aprenderemos a utilizar los dos sistemas de bases de datos que nos ofrece Firebase: Realtime Database y Cloud Firestore.

Objetivos

- Introducir los principios de diseño de bases de datos NoSQL y de tiempo real.
- Mostrar ejemplos de utilización de bases de datos en Firebase.
- Aprender a almacenar información asociada a un usuario.
- Utilizar FirebaseUI para visualizar de forma rápida datos en un RecyclerView.
- Conocer el API que nos permite trabajar con las dos bases de datos de Firebase: Realtime Database y Cloud Firestore.

2.1. Bases de datos NoSQL en tiempo real

Las principales características que podemos destacar de las bases de datos en tiempo real integradas en Firebase son:

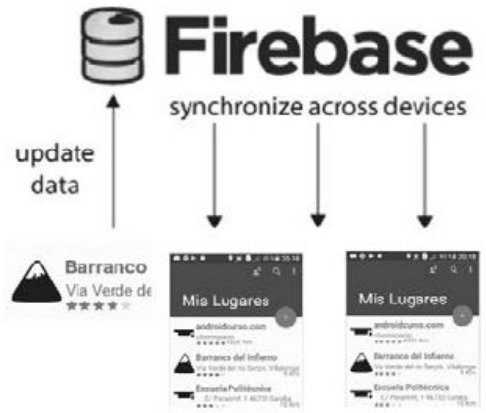
- La base de datos es alojada en la nube. No tenemos que preocuparnos de buscar un servidor e instalar software. Google se encarga de todo.
- Los datos se almacenan en formato similar a JSON. Se trata de una base de datos NoSQL. Más adelante ampliaremos este concepto.
- Sincronización en tiempo real. Cualquier cambio en los datos hace que los clientes reciban actualizaciones en cuestión de milisegundos.
- Permite apps multiplataforma iOS, Android, Web (JavaScript)… Todos nuestros clientes comparten la misma base de datos y reciben actualizaciones de forma automática con los datos más nuevos, independientemente de su plataforma. Además, disponemos de un API REST para acceder a la base de datos desde otras plataformas.
- Si el dispositivo pierde la conexión a Internet, nuestra aplicación podrá seguir funcionando. Mientras tanto se trabajará con una caché. Cuando se recupere la conexión, la información será actualizada.
- Gran escalabilidad. Solución especialmente indicada para bases de datos de gran tamaño (big data).

Vídeo [Tutorial]: Introducing Firebase Realtime Database

2.1.1. Bases de datos en tiempo real

En una base de datos tradicional estamos acostumbrados a realizar una consulta cada vez que necesitamos obtener una información.

En un planteamiento en tiempo real, hemos de indicar en qué datos estamos interesados. Cada vez que cambien estos datos, seremos notificados. En la notificación se incluirá la nueva información para que realicemos con ella las acciones oportunas.



Cuando trabajes con Firebase es muy importante que tengas en cuenta este nuevo planteamiento.

2.1.2. Bases de datos NoSQL

Firebase no utiliza una base de datos SQL, hemos de olvidarnos de crear tablas con sus columnas y definir las relaciones entre ellas. La forma de consultar los datos también va a ser muy diferente.

Vamos a almacenar los datos en un árbol JSON alojado en la nube. Cada vez que añadimos datos, estos han de estar en un nodo de la estructura JSON.

Para ilustrar cómo se crea una base de datos NoSQL, vamos a describir cómo podríamos hacerlo con la aplicación “Mis Lugares”. Dispondremos de tres nodos que colgarán de la raíz: “lugares”, “usuarios” y “valoraciones”.

1. Cada lugar quedará definido con una estructura similar a la que se tenía definida en la aplicación con datos locales.

```
"lugares" : {
  "1" : {
    "nombre" : "Escuela Politécnica Superior de Gandía",
    "direccion" : "C/ Paranimf, 1 46730 Gandia (SPAIN)",
    "posicion" : {
       "longitud" : -0.166093,
       "latitud" : 38.995656
    },
    "tipo" : "EDUCACION",
    "foto" : "http://mmoviles.upv.es/mis_lugares/foto_epsg.jpg",
    "telefono" : 962849300,
    "url" : "http://www.epsg.upv.es",
    "comentario" : "Uno de los mejores lugares para formarse",
    "fecha" : 1504515893130,
    "valoracion" : 3
  },
  "2" : {
    "nombre" : "Barranco del Infierno",
    …
  }
}
```

2. Cada usuario quedará almacenado usando su identificador de usuario como clave principal y, bajo este, su nombre y correo (si están disponibles) y la fecha del último inicio de sesión. Usarás como identificador de usuario el UID generado por Firebase en el módulo de autentificación.

```
"usuarios" : {
   "0yuhn6aEODOzM60xOA4uxlefP253" : {
      "nombre" : "Jesús Tomás",
      "correo" : "jtomas@upv.es",
      "inicioSesion" : 1504515893130
   },
```

```
   "PZLu0o1S4wOHisR726W7WycaMkp1" : {
      "inicioSesion" : 1589313015045
   }
}
```

Dado que el API de autentificación de Firebase ya almacena mucha información del usuario (teléfono y fotografía), no va a ser necesario que almacenemos de nuevo esta información. Podremos obtenerla cuando el usuario está validado.

3. Por último, para almacenar las valoraciones que los usuarios han hecho de diferentes lugares, se representaría de esta forma:

```
 "valoraciones" : {
   "OyuhnGaEODOzM60xOA4uxlefP253" : {
     "1" : 3,
     "4" : 5,
     "5" : 1
   },
   "PZLu0o1S4wOHisR726W7WycaMkp1" : {
     "4" : 3
   }
 }
```

En este ejemplo, el primer usuario ha valorado los lugares 1, 4 y 5, y el segundo, el lugar 3. Existen otras alternativas para codificar esta información que veremos más adelante.En la aplicación en la que se está trabajando, queremos conocer la valoración media de un lugar. Cuando trabajábamos con datos locales solo existía un usuario, por lo que un simple valor era suficiente para almacenar esta información. Ahora, los datos son compartidos y pueden existir múltiples usuarios. Más adelante trataremos de resolver este problema.

Ejercicio: Importar/Exportar datos desde un fichero JSON

1. Descarga el siguiente fichero, que contiene una lista con todos los lugares en JSON: http://mmoviles.upv.es/mis_lugares/mis_lugares.json.
2. En la consola de Firebase, entra en el proyecto “Mis Lugares” y selecciona el apartado DATABASE. Pulsa en Realtime Database / Comenzar:

3. Selecciona Datos / ⋮ / Importar JSON. Indica el fichero anterior. Los datos importados se muestran de la siguiente manera:

4. Cada nodo de esta estructura de datos tiene un URL asociado. La raíz es https://mis-lugares-18eb4.firebaseio.com/. Abre algunos de los siguientes URL:

   https://mis-lugares-18eb4.firebaseio.com/lugares
   https://mis-lugares-18eb4.firebaseio.com/lugares/1
   https://mis-lugares-18eb4.firebaseio.com/lugares/1/nombre

5. Si quieres descargar el nodo correspondiente en formato JSON, puedes usar el mismo URL acabado en .json. Por ejemplo:

   https://mis-lugares-18eb4.firebaseio.com/lugares/1.json

6. Si abres este URL comprobarás que te deniega el acceso. Vamos a dar acceso de lectura a todo el mundo. Selecciona la pestaña **Reglas**:

Reemplaza la regla `read` como se muestra en la captura anterior. De esta forma, se podrá acceder a la base de datos, aunque los usuarios no estén autentificados. Si los datos son privados nunca has de hacer esto en una aplicación real. Por ejemplo, los correos de todos tus usuarios podrían ser extraídos.

7. Recarga el URL anterior. El resultado ha de ser:

```
{"comentario":"Uno de los mejores lugares para formarse", "direccion":
"C/ Paranimf, 1 46730 Gandia (SPAIN)", "fecha":1504515893130,
"foto":"http://mmoviles.upv.es/mis_lugares/foto_epsg.jpg",
"nombre":"Escuela Politécnica Superior de Gandía", "posicion": {
"latitud":38.995656,"longitud":-0.166093}, "telefono":962849300, "ti-
po":"EDUCACION", "url":"http://www.epsg.upv.es", "valoracion":3}
```

Todos los datos de tu base de datos son accesibles vía https. Piensa en lo versátil de esta forma de trabajar. Por ejemplo, si necesitas montar un servicio Web para dar acceso a terceros, ya lo tendrías montado.

8. Otra alternativa para exportar cualquier nodo de la base de datos en JSON, sin tener que modificar los permisos es utilizar desde el backend la opción ⋮ / Exportar. Prueba esta alternativa.

9. El backend de Firebase permite modificar directamente los datos almacenados. Prueba a eliminar nodos, insertarlos, cambiar valores, etc.

10. Cuando termines, puede ser una buena idea importar de nuevo el fichero con los datos originales.

NOTA: Aunque estés acostumbrado a trabajar con ficheros JSON en formato de texto, internamente los datos son representados usando una codificación más eficiente. Los tipos JSON disponibles son String, Long, Double, Boolean, Map<String, Object> y List<Object>. No podemos usar otros como Map<Long, Object>.

```
"map_string_object" : {
  "1" : {
    "string" : "Escuela Politécnica Superior de Gandía",
    "long" : -4567
  },
  "2" : {
    "double" : -0.166093,
    "bolean" : true  }
},
"list_object" : {
  { "prop1" : "valor", "prop2" : 5 },
  { "prop1" : "valor", "prop2" : 5 }
}
```

Como recomendación general evita crear árboles complejos con un nivel de profundidad grande. Esto resulta importante para mejorar la eficiencia. Cada vez que obtienes datos en un nodo de tu base de datos, también recuperas todos los nodos que dependen de él.

2.1.2.1. Recomendaciones para estructurar los datos

Compacta las estructuras de datos

Divide los datos en nodos y asígnales un ID. Mira el ejemplo mostrado al principio de este apartado para lugares y usuarios. Esta regla no se aplica si la estructura es muy pequeña o nunca se repite. Por ejemplo, el nodo `posicion` de un lugar, es mejor dejarlo dentro del lugar que sacarlo con un ID.

Evita la anidación de datos

Firebase Realtime Database permite anidar datos hasta un máximo de 32 niveles. Pero no suele ser recomendable un anidamiento superior a 4 o 5. Debestener en cuenta que cuando pedimos descargar un nodo, también se descargan todos los nodos anidados. Por ejemplo, si dentro del nodo de un lugar, además de su título, autor… añadimos un nodo con todos los UID de usuarios que lo han valorado, cuando queramos la información básica del lugar, estaremos obligados a descargar también esta información. Como se ha mostrado en el ejemplo, para resolverlo adecuadamente, en la raíz podríamos poner un nodo con nombre `valoraciones`, dentro un listado de códigos de lugar y, dentro de cada código, los UID que han valorado el lugar.

Optimiza el acceso a los datos

Es muy importante que recapacites sobre cómo vas a acceder a la información. Si dado un usuario queremos saber las valoraciones que ha hecho, sería correcto utilizar el esquema que cuelga del nodo `valoraciones`. En esta aplicación lo más normal es que el acceso sea al revés. Queremos saber qué valoraciones tiene un lugar. Para obtener esta información estaríamos obligados a descargar todo el nodo `valoraciones`. Por lo tanto, resultaría mucho más eficiente almacenar los datos que que cuelgan del nodo `valoraciones_2`. Donde, para cada clave de lugar, almacenar los UID de usuario con sus valoraciones. Si necesitas hacer los dos tipos de acceso, es decir buscar por lugar y por usuario, tendrías que mantener las dos estructuras. Es decir, dos nodos en la raíz (`valoraciones` y `valoraciones_2`) y cada vez que un usuario visite un lugar, actualizar ambas estructuras de datos. No ha de preocuparte duplicar información, si con esto se mejora el acceso.

```
"valoraciones" : {
  "Oyuhn6aEODOzM60xOA4uxlefP253":{
     "1" : 3,
     "4" : 5,
     "5" : 1
  },
  "PZLu0o1S4wOHisR726W7WycaMkp1":{
     "4" : 3
  }
}
```

```
"valoraciones_2" : {
  "1" : {
    "Oyuhn6aEODOzM60xOA4uxlefP253":3,
  },
  "4" : {
    "Oyuhn6aEODOzM60xOA4uxlefP253":5,
    "PZLu0o1S4wOHisR726W7WycaMkp1":3,
  },
  "5" : {
    "Oyuhn6aEODOzM60xOA4uxlefP253":3,
  }
}
```

Para acabar este punto vamos a ver un resumen con las diferencias entre las dos alternativas:

SQL	NoSQL
Datos estructurados en filas y columnas	No hay estructura prefijada Cada nodo del árbol puede tener una estructura diferente
Uso del lenguaje de consulta SQL	No hay lenguaje de consulta
Problemas de escalabilidad No trabajan en paralelo	Escalabilidad horizontal Apropiadas para big data
Suelen trabajar sobre disco	Pueden trabajar directamente en memoria, y volcar a disco cada cierto tiempo
Lentas, dadas las dos características anteriores	Rápidas, dadas las dos características anteriores

El término escalabilidad horizontal se refiere a que en NoSQL el número de nodos puede aumentar exponencialmente sin afectar de forma drástica al rendimiento. Esto se consigue gracias a que las consultas pueden realizarse en paralelo en todos los nodos a la vez, y reunir luego los resultados para devolvérselos al cliente (consultas Map-Reduce).

2.2. Realtime Database

Se trata de la primera versión de base de datos incorporada en Firebase. Es una solución sencilla, optima, de baja latencia y muy probada. Recientemente se ha lanzado la versión beta de Cloud Firestore, como alternativa de base de datos. Será estudiada en el siguiente apartado. Si dispones de poco tiempo, puedes pasar directamente a este apartado. En este caso, tendrás que realizar algunos ejercicios como los de definición de POJO.

2.2.1. Trabajar con bases de datos

Vamos a realizar una serie de pequeños ejercicios para ir familiarizándonos con el trabajo con bases de datos usando Realtime Database.

Ejercicio: Escribir en la base datos

1. Para los dos ejercicios de este apartado, puedes usar la aplicación “Mis Lugares” y luego borrar el código introducido. O crea una nueva aplicación, con

nombre Prueba, en la consola de Firebase y vincúlala a un nuevo proyecto de Android Studio. En este segundo caso, puedes basarte en el primer ejercicio de esta unidad, o en el tutorial http://www.androidcurso.com/index.php/694.

2. Los datos de tu base de datos están protegidos para que solo los usuarios autorizados puedan acceder. Ya hemos visto cómo Firebase incorpora un completo sistema de autentificación. Aunque en este ejemplo no tendremos usuarios autentificados y vas a dar acceso público a toda la base de datos.

   Accede a la consola de Firebase y selecciona el proyecto “Prueba”. En la columna de la izquierda selecciona **Database** y en las pestañas de la parte superior selecciona **Rules**.

   Reemplaza las reglas predeterminadas por las siguientes:

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

   De esta forma se podrá acceder a la base de datos, aunque los usuarios no estén identificados. Recuerda que nunca has de hacer esto en una aplicación real. Para más información sobre las reglas de las bases de datos: https://firebase.google.com/docs/database/security/. Al final del capítulo se darán más detalles de cómo definir estas reglas de seguridad.

3. Añade al fichero gradle Scripts/Bulid.gradle (Module:app) la siguiente dependencia:

```
dependencies {
    …
    implementation 'com.google.firebase:firebase-database:11.8.0'
}
```

4. En la clase `MainActivity`, dentro del método `onCreate()`, dentro del `onClick()` del botón, inserta las líneas:

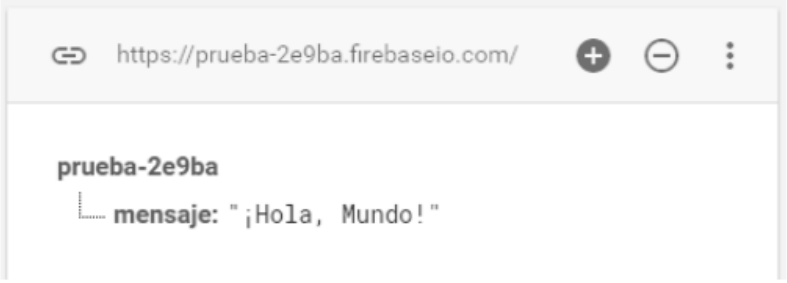
```
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("mensaje");
myRef.setValue("¡Hola, Mundo!");
```

   Comenzamos obteniendo una instancia de tu base de datos usando `getInstance()`. A continuación se obtiene una referencia a la localización donde se quiere escribir usando `getReference()`. Mediante `setValue()` asignamos un valor a la referencia. Si estás acostumbrado a trabajar con bases de datos convencionales (SQL) habrás observado que esta forma de proceder no se parece en nada a lo que conoces.

5. Dentro del mismo método donde acabas de insertar estas líneas se muestra un mensaje con un `Snackbar`. Reemplaza este mensaje por “Escribiendo en la base de datos”.

6. Ejecuta la aplicación y pulsa en el botón flotante.

7. En la consola de Firebase selecciona la pestaña **Datos**. Observa cómo se ha añadido un nuevo nodo a la base de datos:

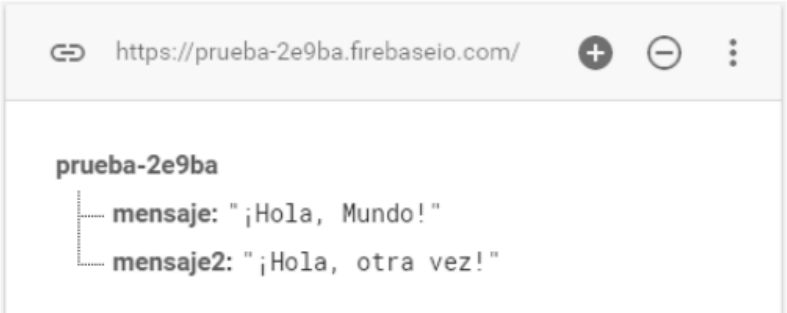


8. En el código de la aplicación reemplaza “mensaje” por “mensaje2”. Puedes cambiar también la información que vas a escribir en este nodo.

9. Pon el dispositivo donde estás ejecutando la aplicación en modo avión. Ejecuta la nueva versión de la aplicación y pulsa en el botón flotante.

10. Observa que la aplicación no muestra ningún error, aunque no disponga de conexión con el servidor. Desde la consola de Firebase, el nuevo nodo no ha de aparecer.

11. Quita el modo avión del dispositivo. Observa cómo, sin necesidad de realizar ninguna acción, el nuevo nodo aparece en la consola.



Ejercicio: Leer de una base datos

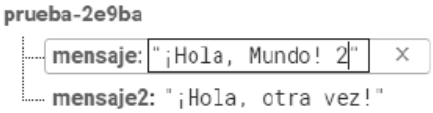
Podemos conseguir que nuestra aplicación ejecute un código para leer el valor anterior y cada vez que se modifique. Para ver cómo conseguirlo, sigue este ejercicio:

1. Inserta el siguiente código dentro del método `onCreate()`:

```
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("mensaje");
myRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        String value = dataSnapshot.getValue(String.class);
        Log.d("Ejemplo Firebase", "Valor: " + value);
    }

    @Override
    public void onCancelled(DatabaseError error) {
        Log.w("Ejemplo Firebase", "Error al leer.", error.toException());
    }
});
```

   Las dos primeras líneas coinciden con el ejercicio anterior. A continuación, añadimos un escuchador de eventos a la referencia. El primer método será ejecutado una primera vez con el valor almacenado, además, volverá a ser llamado cada vez que el valor de la referencia cambie. Se nos pasará como parámetro una instancia de los datos. En este método obtenemos el valor y lo mostramos en el log. El segundo método será ejecutado en caso de cancelación de la operación.

2. Esta vez vamos a modificar el valor de mensaje, directamente desde la consola de Firebase. Selecciona la pestaña **Datos**. pulsa sobre el nodo mensaje e introduce un nuevo valor:



3. En Android Studio selecciona en la pestaña **Android Monitor**. Observa cómo en el logcat aparecen dos líneas. Una con el valor inicial y otra con el nuevo:

```
…/com.example.ejemplofirebase D/Ejemplo Firebase: Valor: ¡Hola, Mundo!
…
…/com.example.ejemplofirebase D/Ejemplo Firebase: Valor: ¡Hola, Mundo! 2
```

4. También puedes reemplazar:

```
myRef.addValueEventListener(new ValueEventListener() {
```

   por:

```
myRef.addListenerForSingleValueEvent(new ValueEventListener() {
```

Observa cómo ahora el método `onDataChange()` solo es llamado una vez.

2.2.2. Definición de POJO

Cuando trabajemos con Firebase, vamos a utilizar un objeto que represente cada una de las entidades que vamos a almacenar (Lugar, GeoPunto y Usuario). La transformación de estos objetos en su representación JSON va a poderse realizar de forma automática, siempre que cumplan una serie de requisitos.

Este tipo de clases suele ser nombrada con las siglas POJO (Plain Old Java Object). Hace referencia a clases simples que no extienden ni implementan nada en especial y que no dependen de un framework, como Firebase o Android. Ejemplos de POJO los tenemos en las clases `Lugar` y `GeoPunto`. En ellas se definen sus atributos, constructores, getters, setters, otros métodos estándares como `toString()` o `equals()` y métodos auxiliares como `GeoPunto.distancia()`.

Al trabajar con la biblioteca FirebaseUI es necesario tener un POJO de las entidades con la que vamos a trabajar. Parte de las características de esta biblioteca es la conversión de los datos (DataSnapshot) entre JSON y una clase concreta definida por el programador. Ya disponemos de una clase que representa un lugar, aunque necesitamos realizar algunos cambios para que sigan las especificaciones de POJO que utiliza Firebase. Algunos datos a tener en cuenta se muestran a continuación:

- Hemos de definir getters y setters de todos los atributos de la clase que queramos que se almacenen en JSON.
- Ha de existir un constructor sin atributos, que permita crea un POJO vacío.
- Para atributos que son clases, que son a su vez POJO, el proceso es automático.
- Para atributos que son `Enum` o clases que no son POJO válidas, hemos de crear un getter y un setter que transformen el objeto en un tipo simple como un `String` o un valor numérico.

NOTA: En la última versión, los Enum, que son tratados directamente.

Ejercicio: Modificar POJO de Lugar

Para poder almacenar un objeto de la clase Lugar en JSON, hemos de asegurarnos de que es un POJO válido. Todos sus atributos son tipos simples con excepción de un `GeoPunto` y un `Enum`. Veamos cómo resolver estos dos casos.

1. La clase `GeoPunto` solo necesita la definición de un constructor vacío para cumplir todos los requisitos. Añade la siguiente línea en la clase:

```
public GeoPunto() {}
```

2. La clase `Lugar` tiene el atributo `tipo` correspondiente a un `Enum`. Al no ser de tipo simple, ni una clase que cumpla las restricciones de un POJO, este atributo será ignorado y no se almacenará en JSON. Para resolver este problema podríamos añadir los siguientes métodos getter y setter (no lo hagas todavía):

```
public String getTipo(){
    if (tipo == null) return null;
    else              return tipo.name();
}

public void setTipo(String nombre){
    if (nombre == null) tipo = null;
    else                tipo = TipoLugar.valueOf(nombre);
}
```

   Estos métodos simplemente transforman el `Enum` en la cadena de caracteres correspondiente.

3. Pero también queremos acceder a este atributo como un `Enum`. Para resolver el problema, vamos a dejar los métodos actuales, cambiando el nombre de los métodos. Para ello, con el botón derecho del ratón pulsa sobre `getTipo` y selecciona Refactor / Rename… Reemplaza el nombre de método por `getTipoEnum`. Realiza el mismo proceso con `setTipo`. De esta forma hemos cambiado tanto el nombre de los métodos, como el de todas sus llamadas.

```
public TipoLugar getTipoEnum() { return tipo;}
public void setTipoEnum(TipoLugar tipo) {this.tipo = tipo;}
```

4. Ya puedes añadir el código del punto 2 para conseguir que el atributo `tipo` sea almacenado en forma de `String`.

Ejercicio: Crear POJO de usuarios

Dado que la aplicación ya no va a funcionar de forma aislada en un único dispositivo, sino que vamos a permitir que los usuarios compartan información, va a ser necesario guardar información de cada usuario.

1. Crea una nueva clase llamada `Usuario` y en ella se colocarán los campos correspondientes para nombre y correo electrónico.

```
public class Usuario {
   private String nombre;
   private String correo;
   private long inicioSesion;

   public Usuario () {}

   public Usuario (String nombre, String correo, long inicioSesion) {
      this.nombre = nombre;
      this.correo = correo;
```

```
      this.inicioSesion = inicioSesion;
   }

   public Usuario (String nombre, String correo) {
      this(nombre, correo, System.currentTimeMillis());
   }
}
```

Además, añadimos tres constructores. Uno vacío, uno que inicializa todos los atributos y utiliza el instante actual para inicializar `inicioSesion`.

2. Por último, son necesarios los getters y setters. Usa el procedimiento anterior.

Ejercicio: Guardar información del usuario

Nos interesa almacenar información de los usuarios que acceden a la aplicación. No sería estrictamente obligatorio, dado que este trabajo también lo hace Firebase Auth. Sin embargo, Firebase Auth solo nos da información del usuario actualmente validado. Por lo tanto, no podríamos obtener una lista de todos los correos de nuestros usuarios. Para conseguirlo realiza el siguiente ejercicio.

1. Verifica que tienes siguiente dependencia en el archivo Gradle módulo app:

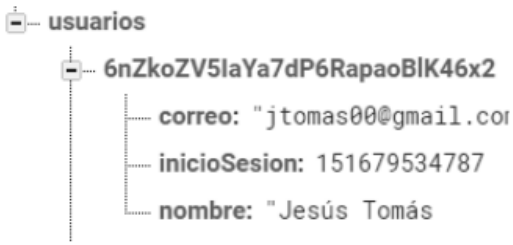
```
implementation 'com.google.firebase:firebase-database:11.8.0'
```

2. Crea la clase `Usuarios`. Será la responsable de almacenar de forma permanente los diferentes objetos `Usuario` con los que trabajaremos en la aplicación. Es un planteamiento similar al usado en la clase `Lugar` y `LugaresVector`, `LugaresBD`... Consiste en separar en una clase el modelo de datos dentro de un POJO y en otra. la forma de almacenarlos usando una tecnología o API concreto.

```
public class Usuarios {

   static void guardarUsuario(final FirebaseUser user) {
      Usuario usuario = new Usuario(
                user.getDisplayName(), user.getEmail());
      FirebaseDatabase database = FirebaseDatabase.getInstance();
      database.getReference("usuarios/"+user.getUid()).setValue(usuario);
   }
}
```

Esta clase solo tiene un método estático para almacenar un usuario. Por lo tanto, no tendremos que crear objetos de esta clase, con llamar al método, suficiente. Además, observa que no se nos pasa un objeto de tipo `Usuario`, sino de tipo `FirebaseUser`. Es así al necesitar el UID como clave.

Empezamos creando un nuevo objeto `Usuario`, que contendrá los tres valores a guardar `nombre`, `correo` e `inicioSesion`. Obtenemos la base de datos e indicamos que en el nodo `usuario/UID` escriba el objeto creado. Por ejemplo, si UID fuera “6nZk…” el resultado podría ser:



Si el usuario entra por primera vez, se creará tanto el nodo "6nZk…" como su contenido. Si entra por segunda vez, el nodo no se creará y lo más seguro es que solo cambie el inicio de sesión. Si solo queremos cambiar este valor, podríamos escribir:

```
database.getReference("usuarios/"+user.getUid()+"/inicioSesion")
                                        .setValue(usuario.getInicioSesion);
```

Resulta más conveniente realizarlo como se muestra en el ejemplo. No hace falta verificar si el usuario existe, además, es posible que haya cambiado su correo electrónico o el nombre.

Existe una forma alternativa de indicar un nodo:

```
database.getReference("usuarios").child(user.getUid()).setValue(usuario);
```

3. En `LoginActivity` añade dentro de `login()` una llamada al método anterior:

```
private void login() {
   FirebaseUser usuario = FirebaseAuth.getInstance().getCurrentUser();
   if (usuario != null) {
      guardarUsuario(usuario);
      …
```

4. En `CustonLoginActivity` añade otra llamada a este método:

```
private void verificaSiUsuarioValidado() {
   if (!unificar && auth.getCurrentUser() != null) {
      guardarUsuario(auth.getCurrentUser());
      …
```

5. En la consola de Firebase, sección DATABASE / REGLAS, verifica que tienes permiso para leer y escribir.

6. Ejecuta la aplicación y valídate con un nuevo usuario. Desde la consola de Firebase, verifica que el usuario ha sido registrado.

2.2.3. Trabajar con FirebaseUI

Vamos a utilizar la biblioteca FirebaseUI, que al igual que pasaba en autentificación, nos va a simplificar ciertas tareas relacionadas con la interfaz de usuario. En concreto, FirebaseUI para Realtime Database nos ofrece `FirebaseRecyclerAdapter`, con el que podremos visualizar información de la base de datos en un `RecyclerView`.

Con la información almacenada de forma local, se hizo uso de un `RecyclerView` y su respectivo `Adapter` para mostrar los datos. Gracias a la biblioteca FirebaseUI, cambiarlo para que los datos sean leídos de una base de datos Firebase,

va a ser muy sencillo. Simplemente, hemos de cambiar el `Adapter` por `FirebaseRecyclerAdapter` e implementar un par de métodos.

Este adaptador tiene una característica muy interesante: cada vez que se produzca un cambio en la base de datos remota, se realizará la correspondiente sincronización de forma totalmente automática.

Ejercicio: Mostrar datos con FirebaseRecyclerAdapter

1. Añade la dependencia:

```
implementation 'com.firebaseui:firebase-ui-database:3.1.3'
```

2. Crea la siguiente clase:

```
public class AdaptadorLugaresFirebaseUI extends
        FirebaseRecyclerAdapter<Lugar, AdaptadorLugares.ViewHolder> {

   protected View.OnClickListener onClickListener;

   public AdaptadorLugaresFirebaseUI(
                       @NonNull FirebaseRecyclerOptions<Lugar> opciones) {
      super(opciones);
   }

   @Override public AdaptadorLugares.ViewHolder onCreateViewHolder(
                                          ViewGroup parent, int viewType) {
      View view = LayoutInflater.from(parent.getContext())
              .inflate(R.layout.elemento_lista, parent, false);
      return new AdaptadorLugares.ViewHolder(view);
   }

   @Override protected void onBindViewHolder(@NonNull AdaptadorLugares
                .ViewHolder holder, int position, @NonNull Lugar lugar) {
      AdaptadorLugares.personalizaVista(holder, lugar);
      holder.itemView.setOnClickListener(onClickListener);
   }

   public void setOnItemClickListener(View.OnClickListener onClick) {
      onClickListener = onClick;
   }

   public String getKey(int pos) {
      return super.getSnapshots().getSnapshot(pos).getKey();
   }
}
```

Observa cómo se extiende de `FirebaseRecyclerAdapter` e indica la clase del modelo de datos (POJO) y el `ViewHolder` que personalizará la lista. No es necesario definir un nuevo `holder`, utilizamos el definido en `AdaptadorLugares`.

A continuación, se incluye el constructor, que simplemente se limita a llamar al `super`. El parámetro opciones se utilizará para incluir una consulta sobre la base de datos. Lo veremos más adelante.

El método `onCreateViewHolder()` es llamado para crear los `holder`. Simplemente creamos una vista inflando el layout y la convertimos en `ViewHolder` usando el método que ya tenemos en `AdaptadorLugares`.

El método `onBindViewHolder()` es llamado cada vez que se descarga un nuevo lugar y hemos de mostrarlo en el `holder` correspondiente. Como esta tarea ya la hacíamos en `AdaptadorLugares`, nos limitamos a llamar al método `personalizaVista`.

También queremos tener la posibilidad de que los elementos listados dispongan de un escuchador `onClick`. Esto se realiza de la misma forma que en un `RecyclerView` normal.

El último método nos permite obtener la clave de un lugar en una determinada posición del `RecyclerView`.

3. Para poder llamar a `personalizaVista` es necesario que sea estático. Añade `static` al método en la clase `AdaptadorLugares`:

```
public static void personalizaVista(ViewHolder holder, Lugar lugar) {
```

4. En la clase `SelectorFragment` vamos a indicar que se use el nuevo adaptador, de momento sin eliminar la definición del anterior. Añade:

```
public static AdaptadorLugaresFirebaseUI adaptador2;
```

5. Añade en `onActivityCreate` el siguiente código:

```
//recyclerView.setAdapter(adaptador);
Query query = FirebaseDatabase.getInstance()
        .getReference()
        .child("lugares")
        .limitToLast(50);
FirebaseRecyclerOptions<Lugar> opciones = new FirebaseRecyclerOptions
        .Builder<Lugar>().setQuery(query, Lugar.class).build();
adaptador2 = new AdaptadorLugaresFirebaseUI(opciones);
recyclerView.setAdapter(adaptador2);
```

Utilizamos la clase `Query` para indicar la consulta a visualizar. En el ejemplo se indica que se muestren los últimos 50 elementos del nodo `lugares`. Con esta consulta se crea el parámetro `opciones`, que, como acabamos de ver, se utiliza en el constructor del adaptador.

6. Para que el adaptador reciba actualizaciones desde la base de datos es necesario llamar al método `startListening()`. Llamaremos a `stopListening()` para dejar de recibirlas. Aunque en la documentación se recomienda hacerlo en los siguientes métodos, nosotros no lo vamos a hacer:

```
@Override public void onStart() {
   super.onStart();
   adaptador2.startListening();
}
```

```
@Override public void onStop() {
   super.onStop();
   adaptador2.stopListening();
}
```

Dado que queremos seguir recibiendo notificaciones tras arrancar otras actividades, Llamaremos a estos métodos en:

```
@Override public void onActivityCreated(Bundle state) {
   …
   adaptador2.startListening();
}
@Override public void onDestroy() {
   super.onDestroy();
   adaptador2.stopListening();
}
```

Llamar a estos métodos en un lugar u otro tiene su importancia. Veamos qué pasa si un usuario que está visualizando el `RecyclerView`, pulsa en un lugar, consulta su contenido y regresa. Con la primera opción (llamamos a `stopListening()` en `onStop()`), toda la lista de lugares que se han descargado para verlos en el `RecyclerView` se destruirá. Al regresar, toda esta información se vuelve a generar. Esto es claramente un inconveniente. Aunque tenemos la ventaja de que mientras estamos viendo el lugar, no nos molestan con actualizaciones, que en este momento no son necesarias. Entonces, ¿qué opción escoger? Para nuestra aplicación pensamos que es mejor la segunda, dado que la consulta a un lugar es muy rápida. Mirar un dato y volver. No obstante, como norma general, se recomienda la primera opción.

7. Verifica el resultado. En la actividad principal ha de mostrarse el listado de los lugares según la información almacenada en Firebase. Sin embargo, al pulsar sobre un lugar, no ha de pasar nada. Es debido a que no hemos usado el método `setOnClickListener()` del nuevo adaptador. Lo haremos en el próximo ejercicio.

8. Además, es posible que, con este nuevo adaptador, el alto del `RecyclerView` quede demasiado pequeño para mostrar todos los elementos. En tal caso, abre el layout `elemento_lista` y modifica el siguiente atributo:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="?android:attr/listPreferredItemHeightLarge"
    android:padding="8dp">
```

9. Accede a la consola de Firebase, sección DATABASE. Si modificas, creas o borras un lugar, podrás observar que los datos son actualizados en el `RecyclerView` de forma automática. Esta es una de las grandes ventajas que nos ofrece `FirebaseRecyclerAdapter` dentro de la librería `FirebaseUI`. También resulta posible realizar este proceso de una forma manual, sin la ayuda de `FirebaseUI`. De esta forma podremos escribir nuestros propios escuchadores de eventos, teniendo mucho más control sobre el proceso. Lo harás en un próximo ejercicio.

2.2.4. Interfaz CRUD asíncrona para Realtime Database

Mostrar los datos almacenados en Firebase en un `RecyclerView` ha sido muy sencillo. Sin embargo, las operaciones básicas CRUD (Create, Read, Update, Delete) no funcionan en la versión actual. La razón es que en la actualidad estamos usando un objeto `LugaresBD` para acceder a los datos, que ha de ser reemplazado por uno de tipo `LugaresFirebase`. Lo razonable sería que la nueva clase siguiera la interfaz `Lugares`, que recordemos tenía los siguientes métodos:

```
public interface Lugares {
   Lugar elemento(int id);  //Devuelve el elemento dado su id
   void anyade(Lugar lugar);//Añade el elemento indicado
   int nuevo();             //Añade un elemento en blanco y devuelve su id
   void borrar(int id);     //Elimina el elemento con el id indicado
   void actualiza(int id, Lugar lugar); //Reemplaza un elemento
   int tamanyo();           //Devuelve el número de elementos
}
```

Por desgracia, esto no va a ser posible debido a dos razones. En esta interfaz la clave para acceder a un elemento (`id`) es de tipo `int`, y Firebase trabaja siempre con claves de tipo `String`. Además, algunos de estos métodos han de ser implementados de forma asíncrona. Por ejemplo, el método `Lugar elemento(int id)` no puede realizar la búsqueda y devolvernos el `Lugar` en la misma llamada, sin bloquear de forma excesiva al sistema. Resulta más adecuado realizar esta consulta de forma asíncrona, es decir, indicamos junto al `id` y un escuchador que será llamado cuando el método haya averiguado el `Lugar` correspondiente. Por lo tanto, va a ser necesario definir una nueva interfaz de acceso a los datos alternativa:

```
public interface LugaresAsinc {
   interface EscuchadorElemento{
      void onRespuesta(Lugar lugar);
   }
   interface EscuchadorTamanyo{
      void onRespuesta(long tamanyo);
   }

   void elemento(String id, EscuchadorElemento escuchador);
   void anyade(Lugar lugar);
   String nuevo();
   void borrar(String id);
   void actualiza(String id, Lugar lugar);
   void tamanyo(EscuchadorTamanyo escuchador);
}
```

Si comparas las dos interfaces observarás que los parámetros id han sido convertidos en `String` y que la respuesta de los métodos `elemento()` y `tamanyo()` ahora se obtiene por medio de un escuchador. Cuando llamemos a uno de estos métodos, ya no nos devolverá directamente el resultado, sino que tendremos que indicar un objeto que implemente la interfaz correspondiente. Cuando el resultado sea averiguado, se llamará al método `onRespuesta()` del objeto que hemos indicado.

Una vez definida esta interfaz, ya podemos implementar una clase que realice estas operaciones en Firebase:

```
public class LugaresFirebase implements LugaresAsinc{
   private final static String NODO_LUGARES  = "lugares";
   private DatabaseReference nodo;

   public LugaresFirebase(){
      FirebaseDatabase database = FirebaseDatabase.getInstance();
      nodo = database.getReference().child(NODO_LUGARES);
   }

   public void elemento(String id, final EscuchadorElemento escuchador) {
      nodo.child(id).addListenerForSingleValueEvent(
        new ValueEventListener() {
          @Override public void onDataChange(DataSnapshot dataSnapshot) {
             Lugar lugar = dataSnapshot.getValue(Lugar.class);
             escuchador.onRespuesta(lugar);
          }
          @Override public void onCancelled(DatabaseError error) {
             Log.e("Firebase", "Error al leer.", error.toException());
             escuchador.onRespuesta(null);
          }
        });
   }

   public void anyade(Lugar lugar) {
      nodo.push().setValue(lugar);
   }

   public String nuevo() {
      return nodo.push().getKey();
   }

   public void borrar(String id) {
      nodo.child(id).setValue(null);
   }

   public void actualiza(String id, Lugar lugar) {
      nodo.child(id).setValue(lugar);
   }

   public void tamanyo(final EscuchadorTamanyo escuchador) {
      nodo.addListenerForSingleValueEvent(new ValueEventListener() {
          @Override public void onDataChange(DataSnapshot dataSnapshot) {
             escuchador.onRespuesta(dataSnapshot.getChildrenCount());
          }
          @Override public void onCancelled(DatabaseError error) {
             Log.e("Firebase", "Error en tamanyo.", error.toException());
             escuchador.onRespuesta(-1);
          }
       });
   }
}
```

La constante `NODO_LUGARES` corresponde al nombre del nodo en JSON. El atributo `nodo` es una referencia de base de datos a este nodo. Este atributo es calculado en el constructor y, así, no es necesario obtenerlo en cada operación con este nodo.

El método `elemento()` es uno de los más extensos. Partiendo de `nodo`, pedimos que busque el nodo hijo con clave `id`. Con este nodo usamos `addListenerForSingleValueEvent()`. Es similar al que ya hemos estudiado, `addListenerValueEvent()`, pero ahora el escuchador solo estará activo una vez. Es decir, leeremos el valor del nodo que será pasado al escuchador, pero si este nodo cambia en el futuro, ya no se volverá a llamar. Los métodos a implementar también son los mismos: `onDataChange()` cuando se lee el nodo y `onCancelled()` cuando hay algún error. La `Interface` definida por nosotros solo tiene un método, `onRespuesta()`. Lo que hacemos es devolver el objeto `Lugar` si hay éxito y `null` en caso contrario.

> **NOTA:** El nombre **`onDataChange()`** puede resultar confuso, dado que en la primera llamada (y, en este caso, también la última) es invocado aunque los datos no hayan cambiado.

El método `anyade()` se limita a crear un nuevo nodo desde el principal (`push()`) y le asigna el objeto que nos han pasado (`setValue()`). El método, `nuevo()` crea un nuevo nodo desde el principal (`push()`), pero sin asignarle un valor. Obtenemos la clave creada para este método (`getKey()`), que es devuelta de forma síncrona.

> **NOTA:** Esta operación puede ser síncrona, dado que la clave es creada desde el cliente antes de ser enviada a Firebase.

Los métodos `borrar()` y `actualiza()` también resultan sencillos. Accedemos al hijo según la clave que nos han indicado y modificamos su valor. En `borrar()` se asigna `null` y en `actualiza()`, el nuevo valor.

Finalmente, `tamanyo()` tiene un funcionamiento similar a `elemento()`, al usar también una interfaz asíncrona. Aunque existen tres diferencias. Ahora pedimos que se descargue todo el nodo raíz con todos los lugares, en lugar de pedir solo un lugar. Vamos a contar el número de elementos (`getChildrenCount()`). En caso de que se produzca un error en la consulta, devolveremos en valor –1.

Ejercicio: Operaciones CRUD con Firebase

En este ejercio vamos a realizar los cambios que acabamos de proponer y modificar el resto de la aplicación para adaptarla al nuevo funcionamiento.

1. Añade al proyecto la interfaz `LugaresAsinc` y la clase `LugaresFirebase`.
2. En `MainActivity`, cambia el tipo al atributo `lugares`:

```
public static LugaresBDAsinc lugares;
```

3. En el método `onCreate()`, cambia la llamada al constructor:

```
lugares = new LugaresBDFirebase (this);
```

4. En este mismo método, cambia el tipo usado para la clave a `String`:

```
longString _id = lugares.nuevo();
```

5. Terminando con esta clase, elimina la siguiente línea de `onActivityResult()`:

```
SelectorFragment.adaptador.setCursor(MainActivity.lugares.extraeCursor());
```

El tema del cursor es propio de SQL, pero no se usa en Firebase.

6. Pasemos a la clase `SelectorFragment`. Elimina el atributo `adaptador` y su inicialización. Pero no elimines `adaptador.setOnItemClickListener(` …
7. Renombra `adaptador2` por `adaptador` (shift-F6).
8. Pasemos a la clase `VistaLugarFragment`. En `onOptionsItemSelected()`, cambia el código que se ejecuta al seleccionar borrar por:

```
case R.id.accion_borrar:
    borrarLugar((int) id);
    return true;
```

9. En el método `borrarLugar()`, cambia el siguiente código:

```
String _id = SelectorFragment.adaptador.getKey(id);
MainActivity.lugares.borrar(_id);
SelectorFragment.adaptador.setCursor(MainActivity.lugares.extraeCursor());
```

Recuerda que `id` es el índice del lugar según su orden en el `RecyclerView`, mientras que `_id` es la clave asociada al lugar. En la primera línea se averigua esta clave, accediendo a la posición en el adaptador. La segunda borra el lugar correspondiente a esta clave. Y la tercera ha de ser eliminada por tratarse de una operación propia de SQL.

10. El método `actualizaLugar()` es llamado cada vez que el usuario realiza un cambio en el lugar que estamos visualizando (como la valoración o la fecha) y queremos almacenar la información. Reemplaza este método por el siguiente:

```
void actualizaLugar(){
   String _id = SelectorFragment.adaptador.getKey((int) id);
   MainActivity.lugares.actualiza(_id, lugar);
}
```

11. Para finalizar, pasemos a la clase `EdicionLugarActivity`. En los atributos realiza el siguiente cambio:

```
private long String _id;
```

12. En `onCreate()`, reemplaza el siguiente código:

```
_id = extras.getLongString("_id", -1 null);
if (_id!=-1null) {
   lugar = MainActivity.lugares.elemento((int) _id) new Lugar();
} else {
   lugar = SelectorFragment.adaptador.lugarPosicion getItem((int) id);
```

```
    _id = SelectorFragment.adaptador.getKey((int) id);
}
```

13. En `onOptionsItemSelected()`, reemplaza el siguiente código:

```
case R.id.accion_cancelar:
   if(getIntent().getExtras().getBoolean("nuevo", false)) {
      MainActivity.lugares.borrar((int) id _id);
   }
   …
case R.id.accion_guardar:
   …
   if (_id==-1) {
      _id = SelectorFragment.adaptador.idPosicion((int) id);
   }
   MainActivity.lugares.actualiza((int) _id, lugar);
   SelectorFragment.adaptador.setCursor(MainActivity.lugares.extraeCur…);
   if (id!=-1) {
      SelectorFragment.adaptador.notifyItemChanged((int) id);
   } else {
      SelectorFragment.adaptador.notifyDataSetChanged();
   }
   …
```

14. En las clases `MapaActivity` y `VistaLugarFragment`, reemplaza:

```
SelectorFragment.adaptador.lugarPosicion(...
```

por:

```
SelectorFragment.adaptador.getItem(...
```

15. En la versión anterior de “Mis Lugares”, las imágenes eran almacenadas en la memoria interna. En la versión actual de la base de datos, las imágenes se almacenan en un servidor Web (http://...). Para poder descargar estas imágenes hay que realizar ciertos cambios. Primero, añadir en `onCreate()` del `MainActivity` las siguientes líneas:

```
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()
                                                .permitAll().build());
```

Vamos a permitir descargar contenido de Internet desde el hilo principal. Resulta más conveniente hacerlo desde otro hilo o usar una librería como Volley. Más adelante usaremos Firebase Storage para descargar estas imágenes.

16. En la clase `VistaLugarFragment`, cambiar la función `reduceBitmap()` por:

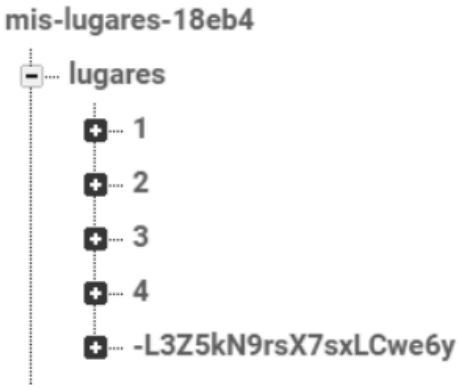
```
public static Bitmap reduceBitmap(Context contexto, String uri,
                                                int maxAncho, int maxAlto) {
   try {
      InputStream input = null;
      Uri u = Uri.parse(uri);
      if (u.getScheme().equals("http") || u.getScheme().equals("https")) {
         input = new URL(uri).openStream();
```

```
      } else {
         input = contexto.getContentResolver().openInputStream(u);
      }
      final BitmapFactory.Options options = new BitmapFactory.Options();
      options.inJustDecodeBounds = true;
      options.inSampleSize = (int) Math.max(
               Math.ceil(options.outWidth / maxAncho),
               Math.ceil(options.outHeight / maxAlto));
      options.inJustDecodeBounds = false;
      return BitmapFactory.decodeStream(input, null, options);
   } catch (FileNotFoundException e) {
      Toast.makeText(contexto, "Fichero/recurso de imagen no encontrado",
                Toast.LENGTH_LONG).show();
      e.printStackTrace();
      return null;
   } catch (IOException e) {
      Toast.makeText(contexto, "Error accediendo a imagen",
                Toast.LENGTH_LONG).show();
      e.printStackTrace();
      return null;
   }
}
```

Anteriormente, esta función solo permitía indicar en el parámetro `uri` imágenes almacenadas en la memoria interna. Con la modificación planteada, también podemos acceder a imágenes en un servidor Web, siempre que `uri` empiece por http:// o https://.

17. Ejecuta el proyecto y verifica el resultado. Si cambias la imagen de algún lugar, se utilizará una URI de almacenamiento interno. Esto solo funcionará en el dispositivo desde donde la imagen ha sigo cargada. En el siguiente apartado aprenderás a usar Firebase Storage para compartir las imágenes.

18. Inserta un nuevo lugar desde la aplicación. Comprueba cómo los primeros lugares, que han sido importados desde JSON, utilizan índices correlativos. Sin embargo los que hemos insertado desde la aplicación, usando `push()`, utilizan como índice una marca temporal:



Práctica: Almacenar usuario que creó cada lugar

¿Podrías almacenar el usuario que creó cada lugar? Los lugares creados al importar no tienen usuario, puedes resolverlo como estimes oportuno. A la hora de borrar un lugar, sería interesante verificar que el usuario que creó el lugar es el mismo que intenta borrarlo.

2.2.5. Creación de un adaptador usando el SDK

En este apartado vamos a realizar la misma tarea que en el apartado Trabajando con FirebaseUI, aunque ahora usando el SDK de Realtime Database. Igual como ocurría en autentificación, trabajar con el SDK va a ser más laborioso, pero vamos a tener más control sobre cómo se realizan las cosas. Hay algunos pasos que son iguales, como la definición de las clases POJO.

Ejercicio: AdaptadorLugares basado en Firebase SDK

1. Verifica que tienes la siguiente dependencia en el archivo build.gradle (Module: app):

```
implementation 'com.google.firebase:firebase-database:11.8.0'
```

2. Crea la siguiente clase:

```
public class AdaptadorLugaresFirebase
               extends RecyclerView.Adapter<AdaptadorLugares.ViewHolder>
               implements ChildEventListener {
   private DatabaseReference reference;
   private ArrayList<String> keys;
   private ArrayList<DataSnapshot> items;
   private LayoutInflater inflador;
   private View.OnClickListener onClickListener;
```

Observa cómo esta vez extendemos de un `RecyclerView` normal. Como en la anterior ocasión, aprovecharemos el `ViewHolder` creado en `AdaptadorLugares`. Además, implementamos la interfaz `ChildEventListener`, para el manejo de los eventos de la base de datos. Consta de los cinco métodos antes descritos.

Cuando trabajábamos con FirebaseUI, la biblioteca se encargaba de descargar los datos y almacenarlos. Al trabajar con SDK, vamos a tener que definir una serie de estructuras para almacenar la lista de lugares. En `items` se guardará la lista de lugares descargada. Es de tipo `DataSnapshot`, en vez de `Lugar`, para que guarde la referencia al nodo Firebase. En `keys` guardaremos la clave de cada uno de los lugares.

3. Añade los siguientes métodos:

```
public AdaptadorLugaresFirebase(Context contexto, DatabaseReference ref){
   keys = new ArrayList<String>();
   items = new ArrayList<DataSnapshot>();
   reference = ref;
   inflador = (LayoutInflater) contexto
                     .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
}

@Override public AdaptadorLugares.ViewHolder onCreateViewHolder(
                                          ViewGroup parent, int viewType) {
   View v = inflador.inflate(R.layout.elemento_lista, null);
   v.setOnClickListener(onClickListener);
   return new AdaptadorLugares.ViewHolder(v);
}

@Override public void onBindViewHolder(AdaptadorLugares.ViewHolder holder,
                                                              int posicion) {
   Lugar lugar = getItem(posicion);
   personalizaVista(holder, lugar);
}
```

El constructor se limita a inicializar las variables. `onCreateViewHolder` es idéntico al usado en otros adaptadores. En el último método, obtenemos el `Lugar` en la posición que nos indican y lo utilizamos para personalizar el `ViewHolder` usando un método ya definido en `AdaptadorLugares`.

4. Añade los métodos:

```
public Lugar getItem(int pos) {
   return items.get(pos).getValue(Lugar.class);
}

public String getKey(int pos) {
   return items.get(pos).getRef().getKey();
}

public void setOnItemClickListener(View.OnClickListener onClick) {
   onClickListener = onClick;
}

@Override public int getItemCount() {return items.size();}
```

Los métodos `getItem()` y `getRef()` serán útiles para obtener el lugar en una determinada posición o la referencia de este lugar. Los dos siguientes métodos son para implementar el evento `onItemClick` y para obtener número de elementos.

5. A diferencia de una base de datos tradicional, al trabajar con tiempo real, en vez de realizar consultas esperamos por eventos de actualización. Es importante recordar que los eventos usan un nodo como referencia. En este caso, el nodo que vamos a usar como referencia es “lugares”. Añade el siguiente código asociado al evento `onChildAdded`:

```
@Override public void onChildAdded(DataSnapshot dataSnapshot, String s) {
    items.add(dataSnapshot);
    keys.add(dataSnapshot.getKey());
    notifyItemInserted(getItemCount()-1);
}
```

Este evento se produce cada vez que se añada un nuevo nodo dentro del nodo "lugares". También se producirá este evento por cada uno de los nodos que ya existan dentro de "lugares". Por lo tanto, vamos a recibir todos los nodos que ya existen. Si se insertara un nuevo nodo dentro de un hijo de "lugares", este evento no se produciría. El que se produciría sería `onChildChanged`.

6. Veamos el código del evento de modificación de un nodo:

```
@Override public void onChildChanged(DataSnapshot dataSnapshot, String s){
   String key = dataSnapshot.getKey();
   int index = keys.indexOf(key);
   if (index!=-1) {
      items.set(index, dataSnapshot);
      notifyItemChanged(index, dataSnapshot.getValue(Lugar.class));
   }
}
```

Cuando cambie un nodo, averiguamos su clave y buscamos la posición de esta clave en el array `keys`. Si la clave ha sido encontrada, actualizamos los datos del nodo en el array `items`. Notificamos que ha cambiado el elemento de correspondiente e indicamos el lugar que acabamos de recibir.

7. El evento de borrado es manejado de la siguiente forma:

```
@Override public void onChildRemoved(DataSnapshot dataSnapshot) {
   String key = dataSnapshot.getKey();
   int index = keys.indexOf(key);
   if (index!=-1) {
      keys.remove(index);
      items.remove(index);
      notifyItemRemoved(index);
   }
}
```

8. Los métodos de mover un nodo y cancelar, no serán implementados.

```
@Override public void onChildMoved(DataSnapshot dataSnapshot, String s) {}
@Override public void onCancelled(DatabaseError databaseError) {}
```

9. Nos queda activar/desactivar los escuchadores de eventos que hemos definido en la interfaz `ChildEventListener`. Añade los siguientes métodos:

```
public void startListening(){
   keys = new ArrayList<String>();
   items = new ArrayList<DataSnapshot>();
   reference.addChildEventListener(this);
}
```

```
public void stopListening(){
   reference.removeEventListener(this);
}
```

Cada vez que activemos el escuchador recibiremos de nuevo todos los nodos de los lugares. Si no inicializas las estructuras de datos, obtendríamos una nueva copia de la lista de lugares a continuación de la que ya tenemos.

Estos métodos serán llamados cuando la actividad pase a estado activo y dejará de tener sentido recibir actualizaciones cuando deje de estarlo (`onResume()/onPause()`). Si queremos recibir notificaciones mientras estemos visibles, usaremos `onStart()/onStop()`. Para recibir notificaciones incluso en segundo plano, puedes utilizar `onCreate() /onDestroy()`.

10. En `SelectorFragment` inicializa `adaptador` con la nueva clase.
11. Ejecuta la aplicación y verifica el resultado.

Práctica: Almacenar puntuación de cada usuario (Realtime Database)

Trata de implementar la posibilidad de que cada usuario pueda realizar una puntuación del lugar de forma individual. Queremos que trabaje de la siguiente forma, cuando las estrellas se muestren en el `RecycerView` representará la puntuación media de un lugar. Cuando se pulse sobre un lugar para mostrar el detalle, las estrellas representarán la votación del usuario. Si el usuario no ha hecho ninguna puntuación se mostrará el texto “por favor, puntúa este lugar”.

Existen dos alternativas para organizar la estructura de datos. Piensa cual será más eficiente.

2.3. Cloud Firestore

Cloud Firestore es la nueva propuesta de bases de datos en tiempo real de Google. Comparte muchas características con Realtime Database, pero aporta nuevas ventajas:

- Nuevo modelo de datos más intuitivo orientado a colecciones y documentos.
- Consultas más ricas y rápidas.
- Realtime Database no permitía más de 100 000 conexiones simultáneas. En caso de superarlas, teníamos que fragmentar en varias bases de datos y ubicarlas diferentes servidores. Por el contrario, Cloud Firestore resulta mucho más escalable. En caso de necesitar varios servidores el proceso de fragmentación es automático, no tenemos que hacer nada.
- Cuando deje de estar en fase beta, permitirá la replicación automática de los datos a varios centros de datos de distintas regiones.
- En la versión anterior, el soporte para trabajar offline no era posible en clientes Web. Cloud Firestore ya lo permite.

Como desventajas cabe señalar:

- Se encuentra en fase beta.
- Mayor latencia.

2.3.1. Modelo de datos

Firestore nos propone una forma de trabajar con los datos algo más estructurada que en Realtime Database, donde nuestra base de datos podía ser vista como un gran fichero JSON. No obstante, el grado de estructuración no es tan grande como en SQL, donde los datos se almacenan siguiendo un esquema de tablas y columnas.

De forma resumida podemos decir que en Firestore almacenamos **datos** en **documentos**, que se organizan en **colecciones**.


```
lugares
  upv
nombre: Universidad Pol
posicion:
  latitud: 38.94
  logitud: -0.1906
tipo: EDUCACION
  la_vital
nombre: La Vital
posicion: …
```

La forma en la que podemos visualizar los datos desde la consola también se estructura en tres columnas. De izquierda a derecha: colecciones, documentos y datos.

Veamos con más detalle cada uno de estos tres elementos.

2.3.1.1. Los Datos

Solo pueden existir dentro de un documento. Disponemos de nueve tipos simples: booleanos, números, strings, puntos geográficos, bloques binarios y marcas de tiempo. Y dos tipos estructurados: matrices (que son listas de un tipo simple) y mapas (que representan una agrupación de datos, es decir, lo que entendemos por objeto).

Una matriz no puede contener otra matriz en su interior, solo un tipo simple. Por lo tanto, no puede haber matrices de más de una dimensión.

Ejemplo de mapa sería guardar un `GeoPunto` compuesto por longitud y latitud. Un mapa puede contener otros mapas y así, de forma indefinida. En la siguiente figura se guarda dentro del documento `upv` el mapa `posicion` y los tipos simples `nombre` y `tipo`.

```
upv
nombre: Universidad Pol
posicion:
   latitud: 38.94
   logitud: -0.1906
tipo: EDUCACION
```

La siguiente tabla muestra información adicional sobre los datos:

Tipo de datos	Orden	Notas
Nulo	Ninguno	—
Booleano	false < true	—
Fecha y hora	Cronológico	precisión máxima es de microsegundos
Entero	Numérico	64 bits, con signo
Número de coma flotante	Numérico	Precisión doble de 64 bits, IEEE 754
Punto geográfico	por latitud, luego por longitud	—
String de texto	Orden de bytes con codificación UTF-8	De 89 bytes a 1 MiB Consultas solo en los primeros 1500 bytes
Bytes	Orden de bytes	De 89 bytes a 1 MiB Consultas solo en los primeros 1500 bytes
Referencia	colección, id docum., colección, id docum…	projects/[PROJECT_ID]/databases/[DATABASE_ID]/documents/[DOCUMENT_PATH]
Matriz	Ninguno	No puede contener otro valor de matriz
Mapa	Ninguno	Objeto incorporado a documento Si indexado, podemos hacer consultas por subcampos

2.3.1.2. Los documentos

Los documentos son registros livianos que contienen campos con valores asignados. Se trata de la unidad mínima para descargar datos.

Todo documento ha de tener un nombre. Este nombre se utiliza como clave y suele crearse a partir de una marca temporal autocreada.

La estructura interna de un documento es similar a un JSON. Aunque permite almacenar tipos adicionales como puntos geográficos y bloques binarios. Además, tienen la limitación de un tamaño máximo de 1 MB.

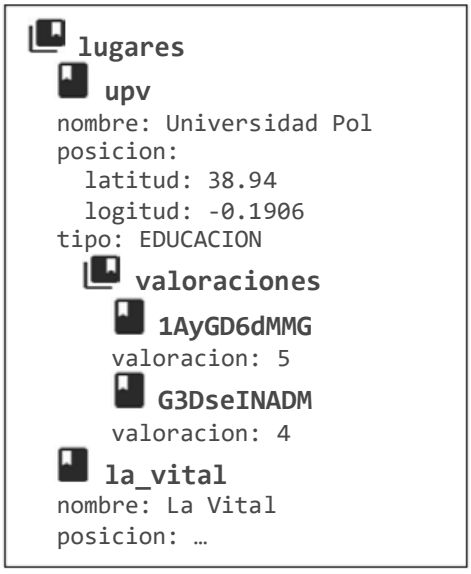
2.3.1.3. Las colecciones

Todo documento ha de almacenarse dentro de una colección. Las colecciones pueden contener documentos de diferentes tipos, aunque es recomendable que todos sean iguales. Por ejemplo, una colección para lugares, otra para usuarios, etc.

No puede contener dos documentos con mismo nombre. En caso de intentar crear un documento cuyo nombre ya exista, daría un error.

No es necesario crear las colecciones, se realizará automáticamente cuando se cree el primer documento. Tampoco hace falta borrarlas, al eliminar el último documento la colección será eliminada.

Una colección no puede contener otras colecciones, solo puede contener documentos. Pero un documento sí que puede contener una colección (en este caso, diremos que es una subcolección). Una subcolección puede contener a su vez documentos, y estos, subcolecciones. Podemos repetir este proceso hasta un anidamiento de 100 niveles. Las colecciones en la raíz ofrecen más flexibilidad y escalabilidad. Además, podemos hacer consultas.



Cuando trabajamos con subcolecciones hemos de tener cuidado de no borrar el documento al que pertenecen sin haberlas eliminado antes El sistema no las elimina automáticamente, quedarían ocupando memoria sin poder tener acceso a ellas.

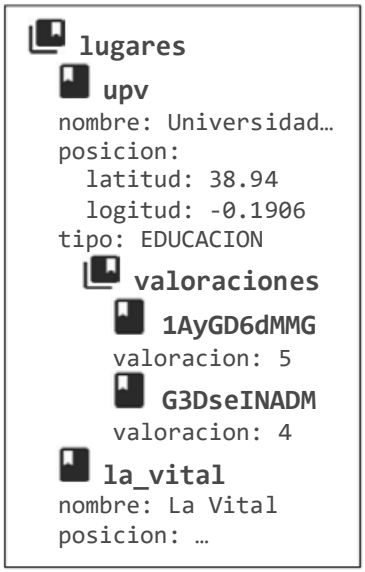
Cada colección o documento se identifica de forma única mediante una referencia según su posición. Veamos algunos ejemplos:

```
CollectionReference cr = db.collection("lugares");
DocumentReference dr = db.collection("lugares").document("upv");
DocumentReference dr = db.document("lugares/upv");
CollectionReference cr = db.collection("lugares/upv/valoraciones");
DocumentReference dr = db.collection("rooms").document("roomA")
        .collection("messages").document("message1");
```

2.3.1.4. ¿Cómo estructurar los datos?

Para organizar los datos en la base de datos será necesario utilizar una estrategia adecuada. Esta decisión es muy importante, de ella dependerá la facilidad que luego tendremos para acceder a los datos.

En este apartado estudiaremos tres posibles alternativas.



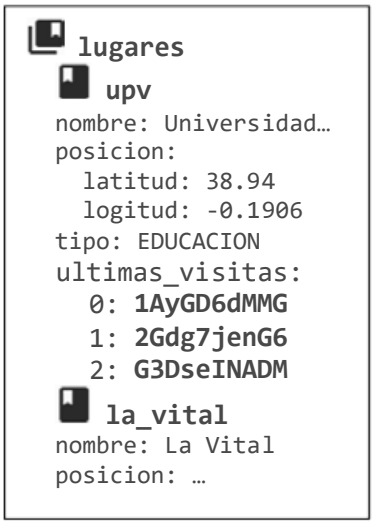
Subcolecciones:

Ejemplo: Como se muestra a la derecha, podemos crear en cada lugar una colección llamada `valoraciones`. En ella almacenaríamos documentos cuyos nombres serían los códigos de usuario que han valorado el lugar.

Ventajas: A medida que crecen las subcolecciones, el tamaño del documento principal no cambia, dado que, cuando descargamos un documento, no se descargan sus subcolecciones.

Limitaciones: No se puede realizar consultas compuestas en subcolecciones.

Datos anidados en documentos:

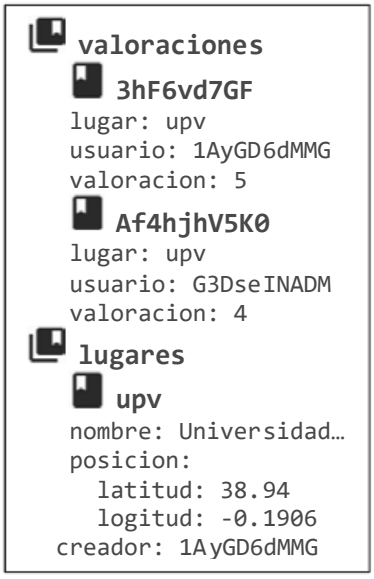


Ejemplo: Como se muestra a la derecha, podemos crear un mapa dentro de cada lugar para almacenar los usuarios que se encuentran en este momento en el lugar o los tres últimos usuarios que lo visitaron.

Ventajas: Si tenemos listas simples y fijas (o que no pueden crecer demasiado) podemos almacenarlas directamente en el documento. Obtendremos la información en la misma lectura con la que obtemos el documento sin apenas penalizar el tamaño de este. Además, optimiza la estructura de datos, no necesitamos gastar recursos en crear nuevas colecciones.

Limitaciones: No podemos hacer consultas en listas anidadas. Menos escalable. Si los datos crecen con el tiempo, el documento crece, por lo que el tiempo de recuperación es lento.

Colecciones de nivel de raíz:



Ejemplo: Para almacenar cada valoración que un usuario hace de un lugar, podemos crear una colección en la raíz, tal y como se muestra a la derecha.

Otro posible ejemplo es almacenar el campo `creador` en cada lugar con el `id` de usuario que lo creó. En la raíz, crearíamos la colección `usuarios` con documentos identificados por este `id`.

Ventajas: Las colecciones en la raíz ofrecen más flexibilidad y escalabilidad. Podemos hacer consultas con índices compuestos. El segundo ejemplo permite que los datos no se dupliquen. Los datos del usuario están solo en la colección raíz, siempre que necesitemos indicar un usuario, pondremos simplemente su `id`.

Limitaciones: Hay que establecer relaciones por clave, que nos obliga a hacer nuevos accesos.

2.3.2. Trabajar con bases de datos

Vamos a realizar una serie de pequeños ejercicios para ir familiarizándonos con el trabajo con bases de datos usando Cloud Firestore.

Ejercicio: Crear la base de datos Firestore y añadir datos

1. En la consola de Firebase, entra en el proyecto “Mis Lugares” y selecciona el apartado DATABASE. Abre el desplegable que se muestra a la derecha de DATABASE y selecciona Cloud Firestore:

2. Te preguntará que tipo de reglas de seguridad quieres utilizar. Selecciona:

NOTA: Recuerda cambiar estas reglas una vez tu aplicación contenga datos reales.

3. A diferencia de Realtime Database, no dispones de una opción directa para importar datos desde JSON. Para realizar esta acción, puedes crear un script de importación [1]. Nosotros iniciaremos los datos en un próximo ejercicio.

4. En el proyecto, añade la siguiente dependencia:

```
implementation 'com.google.firebase:firebase-firestore:11.8.0'
```

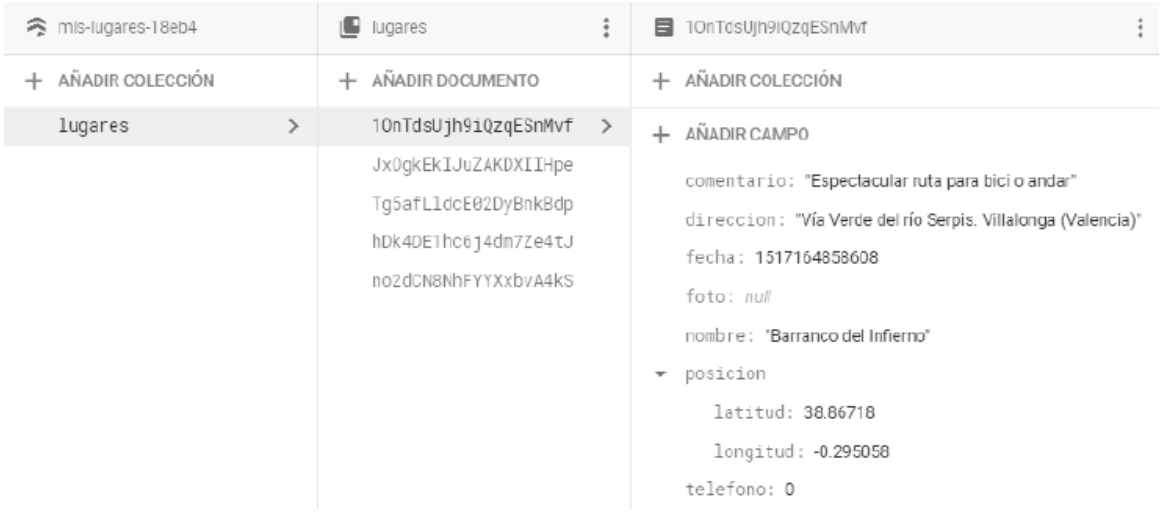
5. En `MainActivity`, dentro de `onCreate()`, añade el siguiente código:

```
FirebaseFirestore db = FirebaseFirestore.getInstance();
for (Lugar lugar: LugaresVector.ejemploLugares()){
   db.collection("lugares").add(lugar);
}
```

Comenzamos obteniendo una instancia de la base de datos. El método `ejemploLugares()` nos da un listado de objetos de tipo Lugar. Cada uno de estos objetos es añadido a la colección `lugares`.

[1] https://hackernoon.com/filling-cloud-firestore-with-data-3f67d26bd66e

6. Ejecuta la aplicación y accede a la consola Firebase. Verifica el resultado:



7. Faltarían los URL en `foto`. Puedes introducirlos a maño desde la consola. En el fichero JSON utilizado en el apartado anterior, tienes URL para muchos de los lugares.

8. Comenta el código anterior. De lo contrario, cada vez que entres en la aplicación, se añadirán nuevos elementos a la base de datos.

2.3.3. Definición de POJO

Cuando trabajemos con Firestore, vamos a utilizar POJO, de la misma forma que lo hacíamos en Realtime Database. Si no has saltado este apartado, es importante que leas el punto de POJO y hagas los ejercicios.

Los requisitos que necesitamos son los mismos, aunque en Firestore aparecen nuevos tipos de datos que podemos almacenar de forma nativa.

Ejercicio: Guardar usuario en Firestore

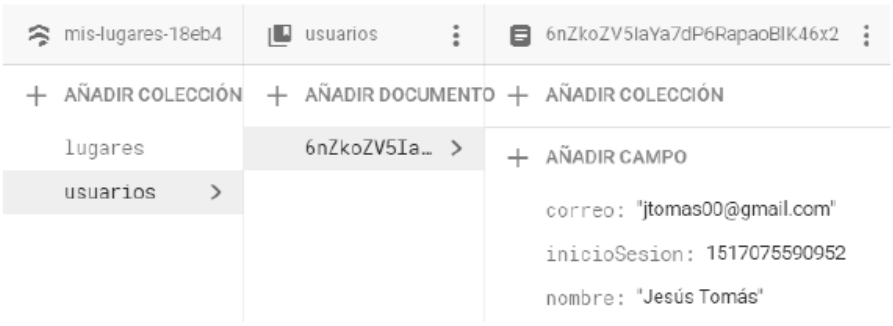
En el último ejercicio de la sección de POJO, es necesario que realices un pequeño ajuste para que funcione en Firestore.

1. Si no lo has realizado ya, sigue los pasos del ejercicio Guardar información del usuario.

2. En el siguiente método, reemplaza las líneas subrayadas:

```
static void guardarUsuario(final FirebaseUser user) {
   Usuario usuario = new Usuario(user.getDisplayName(), user.getEmail());
   FirebaseFirestore db = FirebaseFirestore.getInstance();
   db.collection("usuarios").document(user.getUid()).set(usuario);
}
```

3. Ejecuta la aplicación y verifica en la consola que se crea el usuario:



2.3.1. Trabajar con FirebaseUI

Vamos a utilizar la biblioteca FirebaseUI, que, al igual que pasaba en autentificación, nos va a simplificar ciertas tareas relacionadas con la interfaz de usuario. En concreto, FirebaseUI para Cloud Firestore nos ofrece `FirestoreRecyclerAdapter`, con el que podremos visualizar información de la base de datos en un `RecyclerView`.

Con la información almacenada de forma local, se hizo uso de un `RecyclerView` y su respectivo `Adapter` para mostrar los datos. Gracias a la biblioteca FirebaseUI, cambiarlo para que los datos sean leídos de una base de datos Firestore va a ser muy sencillo. Simplemente, hemos de cambiar el `Adapter` por `FirestoreRecyclerAdapter` e implementar un par de métodos.

Este adaptador tiene una característica muy interesante: cada vez que se produzca un cambio en la base de datos remota, se realizará la correspondiente sincronización de forma totalmente automática.

Ejercicio: Mostrar datos con FirestoreRecyclerAdapter

NOTA: Este ejercicio es muy parecido a Mostrar datos con FirebaseRecyclerAdapter; para resaltar este hecho, se han remarcado las diferencias.

1. Añade la dependencia:

```
implementation 'com.firebaseui:firebase-ui-firestore:3.1.3'
```

2. Crea la siguiente clase:

```
public class AdaptadorLugaresFirestoreUI extends
            FirestoreRecyclerAdapter<Lugar, AdaptadorLugares.ViewHolder> {
   protected View.OnClickListener onClickListener;

   public AdaptadorLugaresFirestoreUI(
                        @NonNull FirestoreRecyclerOptions<Lugar> options) {
      super(options);
   }

   @Override public AdaptadorLugares.ViewHolder onCreateViewHolder(
                                          ViewGroup parent, int viewType) {
```

```
        View view = LayoutInflater.from(parent.getContext())
                            .inflate(R.layout.elemento_lista, parent, false);
        return new AdaptadorLugares.ViewHolder(view);
    }

    @Override protected void onBindViewHolder(@NonNull AdaptadorLugares
                    .ViewHolder holder, int position, @NonNull Lugar lugar) {
        AdaptadorLugares.personalizaVista(holder, lugar);
        holder.itemView.setOnClickListener(onClickListener);
    }

    public void setOnItemClickListener(View.OnClickListener onClick) {
        onClickListener = onClick;
    }

    public String getKey(int pos) {
        return super.getSnapshots().getSnapshot(pos).getId();
    }
}
```

Observa cómo se extiende de `FirestoreRecyclerAdapter` indicando la clase del modelo de datos (POJO) y el `ViewHolder` que personalizará la lista. No es necesario definir un nuevo `holder`, utilizamos el definido en `AdaptadorLugares`.

A continuación, se incluye el constructor, que simplemente se limita a llamar al `super`. El parámetro opciones se utilizará para incluir una consulta sobre la base de datos. Lo veremos más adelante.

El método `onCreateViewHolder()` es llamado para crear los `holder`. Simplemente creamos una vista inflando el layout y la convertimos en `ViewHolder` usando el método que ya tenemos en `AdaptadorLugares`.

El método `onBindViewHolder()` es llamado cada vez que se descarga un nuevo lugar y hemos de mostrarlo en el `holder` correspondiente. Como esta tarea ya la hacíamos en `AdaptadorLugares`, nos limitamos a llamar al método `personalizaVista`.

También queremos tener la posibilidad de que los elementos listados dispongan de un escuchador `onClick`. Esto se realiza de la misma forma que en un `RecyclerView` normal.

El último método nos permite obtener la clave de un lugar en una determinada posición del `RecyclerView`.

3. Para poder llamar a `personalizaVista` es necesario que sea estático. Añade `static` al método en la clase `AdaptadorLugares`:

```
public static void personalizaVista(ViewHolder holder, Lugar lugar) {
```

4. En la clase `SelectorFragment` vamos a indicar que se use el nuevo adaptador, de momento sin eliminar la definición del anterior. Añade:

```
public static AdaptadorLugaresFirestoreUI adaptador2;
```

5. Añade en `onCreate` el siguiente código:

```
//recyclerView.setAdapter(adaptador);
Query query = FirebaseFirestore.getInstance()
        .collection("lugares")
        .limit(50);
FirestoreRecyclerOptions<Lugar> opciones = new FirestoreRecyclerOptions
        .Builder<Lugar>().setQuery(query, Lugar.class).build();
adaptador2 = new AdaptadorLugaresFirestoreUI(opciones);
recyclerView.setAdapter(adaptador2);
```

Utilizamos la clase `Query` para indicar la consulta a visualizar. En el ejemplo se indica que se muestren solo 50 elementos del nodo `lugares`. Con esta consulta se crea el parámetro `opciones`, que, como acabamos de ver, se utiliza en el constructor del adaptador.

6. Para que el adaptador reciba actualizaciones desde la base de datos es necesario llamar al método `startListening()`. Llamarás a `stopListening()` para dejar de recibirlas. Se recomienda hacerlo en los siguientes métodos:

```
@Override public void onStart() {
   super.onStart();
   adaptador2.startListening();
}
@Override public void onStop() {
   super.onStop();
   adaptador2.stopListening();
}
```

No obstante, dado que queremos seguir recibiendo notificaciones tras arrancar otras actividades y que los adaptadores sigan activos, llamarás a estos métodos en:

```
@Override public void onActivityCreated(Bundle state) {
   …
   adaptador2.startListening();
}
@Override public void onDestroy() {
   super.onDestroy();
   adaptador2.stopListening();
}
```

7. Verifica el resultado. En la actividad principal ha de mostrarse el listado de los lugares según la información almacenada en Firestore. Sin embargo, al pulsar sobre un lugar no ha de pasar nada. Es debido a que no hemos usado el método `setOnClickListener()` del nuevo adaptador. Lo haremos en el próximo ejercicio.

8. Además, es posible que, con este nuevo adaptador, el alto del `RecyclerView` quede demasiado pequeño para mostrar todos los elementos. En tal caso, abre el layout `elemento_lista` y modifica el siguiente atributo:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
    android:layout_height="?android:attr/listPreferredItemHeightLarge"
    android:padding="8dp">
```

9. Accede a la consola de Firebase, sección Database / Firestore. Si modificas, creas o borras un lugar, podrás observar que los datos son actualizados en el `RecyclerView` de forma automática. Esta es una de las grandes ventajas que nos ofrece `FirestoreRecyclerAdapter` dentro de la librería `FirebaseUI`. También resulta posible realizar este proceso de una forma manual, sin la ayuda de `FirebaseUI`. De esta forma, podremos escribir nuestros propios escuchadores de eventosy tendremos mucho más control sobre el proceso. Lo haremos en un próximo ejercicio.

2.3.2. Interfaz CRUD asíncrona para Firestore

En caso de que no lo hayas hecho ya, lee el apartado Interfaz CRUD asíncrona para Realtime Database.

Una vez definida la interfaz asíncrona para lugares, ya podemos implementar una clase que realice estas operaciones en Firestore:

```
public class LugaresFirestore implements LugaresAsinc {
   private CollectionReference lugares;

   public LugaresFirestore() {
      FirebaseFirestore db = FirebaseFirestore.getInstance();
      lugares = db.collection("lugares");
   }

   public void elemento(String id, final EscuchadorElemento escuchador) {
      lugares.document(id).get().addOnCompleteListener(
         new OnCompleteListener<DocumentSnapshot>() {
            @Override
            public void onComplete(@NonNull Task<DocumentSnapshot> task) {
               if (task.isSuccessful()) {
                  Lugar lugar = task.getResult().toObject(Lugar.class);
                  escuchador.onRespuesta(lugar);
               } else {
                  Log.e("Firebase", "Error al leer", task.getException());
                  escuchador.onRespuesta(null);
               }
            }
         });
   }

   public void anyade(Lugar lugar) {
      lugares.document().set(lugar);      //o lugares.add(lugar);
   }

   public String nuevo() {
      return lugares.document().getId();
   }
```

```
    public void borrar(String id) {
        lugares.document(id).delete();
    }

    public void actualiza(String id, Lugar lugar) {
        lugares.document(id).set(lugar);
    }

    public void tamanyo(final EscuchadorTamanyo escuchador) {
        lugares.get().addOnCompleteListener(
            new OnCompleteListener<QuerySnapshot>() {
                @Override
                public void onComplete(@NonNull Task<QuerySnapshot> task) {
                    if (task.isSuccessful()) {
                        escuchador.onRespuesta(task.getResult().size());
                    } else {
                        Log.e("Firebase","Error en tamanyo",task.getException());
                        escuchador.onRespuesta(-1);
                    }
                }
            });
    }

}
```

El atributo `lugares` es una referencia a la colección, donde guardaremos los lugares. Este atributo es calculado en el constructor; así, no es necesario obtenerlo en cada operación con este nodo.

El método `elemento()` es uno de los más extensos. Partiendo de `lugares` pedimos que obtenga el documento hijo con clave `id`. Usamos `addOnCompleteListener()` para indicar el escuchador que recibirá el documento. Solo hay un método a implementar, `onComplete()`. Miramos si la operación ha podido realizarse. En caso afirmativo, obtenemos el lugar y se lo pasamos al escuchador que nos han indicado en los parámetros. En caso negativo, devolveremos `null` al escuchador.

El método `anyade()` se limita a crear un nuevo documento (`document()`) en la colección `lugares` y asignarle el objeto que nos han pasado (`set()`). Sería equivalente utilizar `lugares.add(lugar)`. El método, `nuevo()` crea un nuevo documento en la colección `lugares`, pero sin asignarle un valor. Obtenemos la clave creada para este método (`getId()`), que es devuelta de forma síncrona.

> **NOTA:** Esta operación puede ser síncrona, dado que la clave es creada desde el cliente antes de ser enviada a Firebase.

Los métodos `borrar()` y `actualiza()` también resultan sencillos. Accedemos al hijo según la clave que nos han indicado y modificamos su valor o lo borramos.

Finalmente `tamanyo()` tiene un funcionamiento similar a `elemento()`, al usar también una interfaz asíncrona. Aunque existen tres diferencias. Ahora pedimos que se descargue toda la colección con todos los lugares, en lugar de pedir solo

un lugar. Vamos a contar el número de elementos (`size()`). En caso de que se produzca un error en la consulta, devolveremos en valor –1.[2]

Ejercicio: Operaciones CRUD con Firestore

En caso de que no hayas realizado el ejercicio Operaciones CRUD con Firebase, hazlo. Solo tienes que modificar la siguiente línea para utilizar Firestore:

1. En el método `onCreate()` de `MainActivity` cambia:

```
lugares = new LugaresFirebaseFirestore();
```

Práctica: Almacenar usuario que creó cada lugar en Firestore

¿Podrías almacenar el usuario que creó cada lugar? Los lugares creados al importar no tienen usuario, puedes resolverlo como estimes oportuno. Llama al nuevo campo `creador`. A la hora de borrar un lugar, sería interesante verificar que el usuario que creó el lugar es el mismo que intenta borrarlo.

2.3.3. Creación de un adaptador usando el SDK

En este apartado vamos a realizar la misma tarea que en el apartado Trabajando con FirebaseUI, aunque ahora usando el SDK de Cloud Firestore. Igual como ocurría en autentificación, trabajar con el SDK va a ser más laborioso, pero vamos a tener más control sobre cómo se realizan las cosas. Hay algunos pasos que son iguales, como la definición de las clases POJO.

Ejercicio: AdaptadorLugares basado en Firestore SDK

1. Crea la siguiente clase:

```
public class AdaptadorLugaresFirestore
              extends RecyclerView.Adapter<AdaptadorLugares.ViewHolder>
              implements EventListener<QuerySnapshot> {
   public static final String TAG = "Mislugares";
   private Query query;
   private List<DocumentSnapshot> items;
   private ListenerRegistration registration;
   private LayoutInflater inflador;
   private View.OnClickListener onClickListener;
```

[2] Descargar toda la colección para contar sus elementos no parece muy eficiente. Para otra alternativa basada en Functions, véase:
https://stackoverflow.com/questions/46554091/firebase-firestore-collection-count

Observa como esta vez extendemos de un `RecyclerView` normal. Como en la anterior ocasión, aprovecharemos el `ViewHolder` creado en `AdaptadorLugares`. Además, implementamos la interfaz `EventListener`, para el manejo de los eventos de la base de datos. Consta del método antes descrito.

Cuando trabajábamos con FirebaseUI, la biblioteca se encargaba de descargar los datos y almacenarlos. Al trabajar con SDK, vamos a tener que definir una estructura para almacenar la lista de lugares. En `items` se guardará la lista de lugares descargada. Es de tipo `DocumentSnapshot`, en vez de `Lugar`, para que guarde la referencia al documento Firestore y la clave de cada uno de los lugares.

2. Añade los siguientes métodos:

```
public AdaptadorLugaresFirestore(Context contexto, Query query){
   items = new ArrayList<DocumentSnapshot>();
   reference = ref;
   this.query = query;
   inflador = (LayoutInflater) contexto
                     .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
}

@Override public AdaptadorLugares.ViewHolder onCreateViewHolder(
                                          ViewGroup parent, int viewType) {
   View v = inflador.inflate(R.layout.elemento_lista, null);
   v.setOnClickListener(onClickListener);
   return new AdaptadorLugares.ViewHolder(v);
}

@Override public void onBindViewHolder(AdaptadorLugares.ViewHolder holder,
                                                              int posicion) {
   Lugar lugar = getItem(posicion);
   personalizaVista(holder, lugar);
}
```

El constructor se limita a inicializar las variables. `onCreateViewHolder` es idéntico al usado en otros adaptadores. En el último método, obtenemos el `Lugar` en la posición que nos indican y lo utilizamos para personalizar el `ViewHolder` usando un método ya definido en `AdaptadorLugares`.

3. Añade los métodos:

```
public Lugar getItem(int pos) {
   return items.get(pos).toObject(Lugar.class);
}

public String getKey(int pos) {
   return items.get(pos).getId(); }

@Override public int getItemCount() {return items.size();}

public void setOnItemClickListener(View.OnClickListener onClick){
   onClickListener = onClick;
}
```

Los métodos `getItem()` y `getKey()` serán útiles para obtener el lugar en una determinada posición o la clave de este lugar. Los dos siguientes métodos son para obtener número de elementos y para indicar el escuchador del evento `onItemClick`.

4. A diferencia de una base de datos tradicional, al trabajar con tiempo real, en vez de realizar consultas, esperamos por eventos de actualización. Es importante recordar que los eventos usan una `Query` asociada a una colección. Esta información se ha pasado en el constructor. Nos queda activar/desactivar el escuchador de eventos que vamos a definir para actuar ante los cambios en la colección. Añade los siguientes métodos:

```
public void startListening(){
   items = new ArrayList<DocumentSnapshot>();
   registration = query.addSnapshotListener(this);
}

public void stopListening(){
   registration.remove();
}
```

Cada vez que activemos el escuchador recibiremos de nuevo todos los nodos de la `Query`. Si no inicializáramos las estructuras de datos, obtendríamos una nueva copia de la lista de lugares a continuación de la que ya tenemos. Observa cómo se indica que somos nosotros quien vamos a recibir los eventos, por lo que hemos implementado la interfaz `EventListener<QuerySnapshot>`. Además, guardamos el registro del escuchador en una variable para poder eliminarlo en `stopListener()`.

Estos métodos han de ser llamados cuando la actividad pase a estado activo, y dejará de tener sentido recibir actualizaciones cuando deje de estarlo (`onResume()/onPause()`). Si queremos recibir notificaciones mientras estemos visibles, usaremos `onStart()/onStop()`. Para recibir notificaciones, incluso en segundo plano, puedes utilizar `onCreate() /onDestroy()`.

5. Añade el siguiente método para implementar `EventListener<QuerySnapshot>`:

```
@Override
public void onEvent(QuerySnapshot snapshots, FirebaseFirestoreException e){
   if (e != null) {
      Log.w(TAG, "error al recibir evento", e);
      return;
   }
   for (DocumentChange dc : snapshots.getDocumentChanges()) {
      int pos = dc.getNewIndex();
      int oldPos = dc.getOldIndex();
      switch (dc.getType()) {
         case ADDED:
            items.add(pos, dc.getDocument());
            notifyItemInserted(pos);
            break;
         case REMOVED:
            items.remove(oldPos);
```

```
                notifyItemRemoved(oldPos);
                break;
            case MODIFIED:
                items.remove(oldPos);
                items.add(pos,dc.getDocument());
                notifyItemRangeChanged(min(pos,oldPos),abs(pos-oldPos)+1);
                break;
            default:
                Log.w(TAG, "Tipo de cambio desconocido", e);
        }
    }
}
```

Notarás una gran diferencia con la forma de recoger los eventos en Realtime Database. Ahora, se pueden recibir varios eventos juntos de diferentes tipos. Esto permite un procesamiento más eficiente. Tras verificar que no haya habido un error, vamos a realizar un bucle para tratar cada evento de forma individual. Utilizaremos un objeto de tipo `DocumentChange` que nos informará del documento que ha cambiado (`getDocument()`), el tipo de cambio (`getType()`), la posición que ocupa ahora el documento (`getNewIndex()`) y la posición que ocupaba antes del cambio (`getOldIndex()`).

Según el tipo de cambio, se realizarán acciones diferentes: Si ha sido añadido (`ADD`) lo insertaremos en la lista local en su posición. Avisaremos al `RecyclerView` de la inserción para que lo repinte. Este evento también se producirá al activar el escuchador por cada uno de los nodos que ya existan. Por lo tanto, vamos a recibir todos los nodos que ya existen. Si se ha borrado un documento (`REMOVED`), lo borramos de la lista local y lo notificamos al `RecyclerView`. El último caso (`MODIFIED`) hay que estudiarlo con atención. Si `pos` y `oldPos` coinciden, se ha modificado un documento sin que haya cambiado de posición. Pero si no coinciden, además de modificarse, ha cambiado de posición. Esto es posible en una consulta ordenada cuando se modifique el valor por el que estamos ordenando. Para sincronizar este cambio con la lista local, vamos a eliminar el documento donde estaba antes e insertaremos el cambiado en su nueva posición. Todos los elementos entre `pos` y `oldPos` habrán cambiado de posición, por lo que habrá que notificárselo al `RecyclerView`.

6. Modifica la inicialización de `adaptador` en `SelectorFragment`. Tendrás que comentar la variable `options`, dado que ahora no es necesaria. No elimines el código
7. Ejecuta la aplicación y verifica el resultado. En un próximo ejercicio se introducirán filtros y ordenación en la consulta. Podrás verificar que, aunque cambien los lugares, estos siempre son mostrados según los criterios indicados.

2.3.4. Realizar consultas

Como hemos visto, podemos leer todos los documentos de una colección o ser notificados de cualquier actualización. Para esto utilizábamos:

```
FirebaseFirestore db = FirebaseFirestore.getInstance();
CollectionReference lugares = db.collection("lugares");

lugares.get().addOnCompleteListener(escuchador)
ó
lugares.addSnapshotListener(escuchador)
```

En algunos casos no estamos interesados en obtener todos los documentos de una colección, sino solo los que cumplen una determinada condición. Veamos algunos ejemplos:

```
Query query = lugares.whereEqualTo("tipo", "BAR");
```

La clase `CollectionReference` desciende de `Query`, por lo que podemos usar ambas clases de forma similar:

```
query.get().addOnCompleteListener(escuchador)
ó
query.addSnapshotListener(escuchador)
```

El resultado es obvio, descargaríamos una colección con los lugares cuyo `tipo` es `BAR`. Existen 5 variantes del método `where()`, según queramos comprobar: ==, <, <=, > o >=.

```
lugares.whereEqualTo("tipo", "BAR");
lugares.whereLessThan("fecha", 1514967298608);
lugares.whereGreaterThanOrEqualTo("valoracion", 4);
```

También puedes concatenar varios `where()`. El resultado equivale a usar el operador lógico `and`:

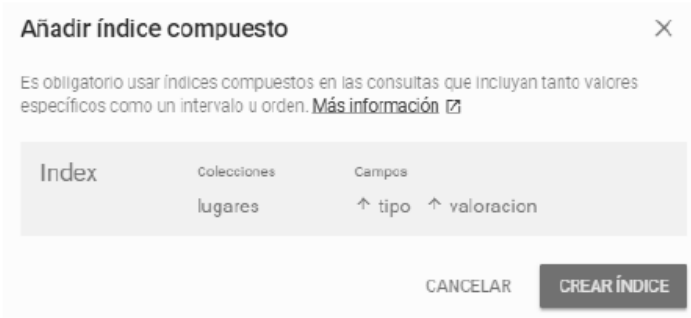
```
lugares.whereEqualTo("tipo", "BAR")
       .whereGreaterThanOrEqualTo("valoracion", 4);
```

Si escribimos la consulta anterior, comprobaremos que no se muestra ningún resultado. Si miramos en el Logcat, aparece el siguiente Warning:

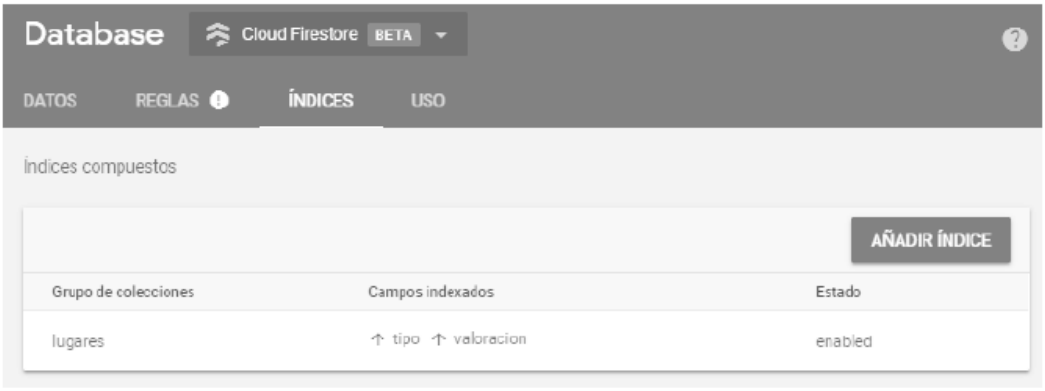
```
…FAILED_PRECONDITION: The query requires an index. You can create it here:
https://console.firebase.google.com/project/mis-lugares-18eb4/...
```

NO SE PERMITE REALIZAR UNA CONSULTA SOBRE DOS CAMPOS SIN HABER PREPARADO PRIMERO EL ÍNDICE CORRESPONDIENTE.

Si pulsamos sobre el link, nos permite crear el índice automáticamente:



Tras unos minutos, el nuevo índice es creado. Podemos verificarlo en la consola de Firebase sección Database / Cloud Firestore / ÍNDICES:



NOTA: Para realizar consultas por un solo campo no es necesario crear el índice. De forma automática se crea un índice para cada campo individual.

Podemos añadir nuevos índices de forma manual pulsando el botón correspondiente. Debemos seleccionar un orden ascendente o descendente en cada campo, incluso si no necesitamos el campo para ordenar. Nuestra elección no afecta el comportamiento en las consultas de igualdad.

Hay otra regla que tienes que tener en cuenta:

NO SE PERMITE REALIZAR UNA CONSULTA DE TIPO < O > SIMULTÁNEAMENTE SOBRE MÁS DE UN CAMPO DISTINTO.

Intenta realizar la siguiente consulta:

```
lugares.whereGreaterThan("tipo", "BAR")
       .whereGreaterThanOrEqualTo("valoracion", 4);
```

En el Logcat aparecerá el siguiente error:

```
All where filters other than whereEqualTo() must be on the same field. But
you have filters on 'tipo' and 'valoracion'
```

Sin embargo, la siguiente es válida:

```
lugares.whereEqualTo("tipo", "BAR")
       .whereGreaterThanOrEqualTo("valoracion", 2)
       .whereLessThan("valoracion", 4.5);
```

Podríamos añadir nuevas condiciones `whereEqualTo()` para otros campos siempre que creemos el índice correspondiente.

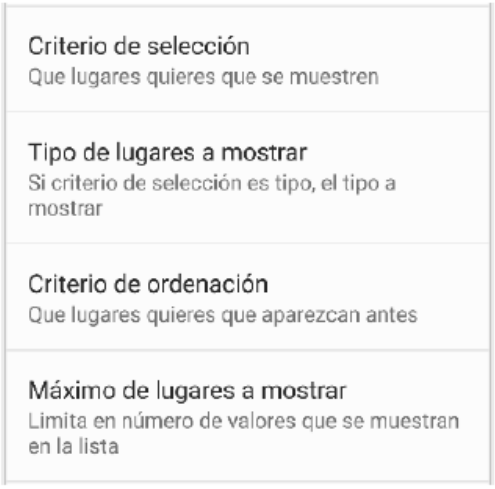
Ejercicio: Filtrar y ordenar la lista de lugares

Vamos a poner en práctica todo lo aprendido es este punto, para que el `RecyclerView` de "Mis Lugares" muestre la información que nos interese en el orden adecuado.

1. Añade las siguientes referencias a res/xml/preferencias.xml:

```
<ListPreference
    android:key="seleccion"
    android:title="Criterio de selección"
    android:summary="Que lugares quieres que se muestren"
    android:entries="@array/tiposSeleccion"
    android:entryValues="@array/tiposSeleccionValores"
    android:defaultValue="1"/>
<EditTextPreference
    android:key="tipo_seleccion"
    android:title="Tipo de lugares a mostrar"
    android:summary="Si criterio de selección es tipo, el tipo a mostrar"
    android:inputType="textCapCharacters"
    android:defaultValue="BAR"/>
<ListPreference
    android:key="orden"
    android:title="Criterio de ordenación"
    android:summary="Que lugares quieres que aparezcan antes"
    android:entries="@array/tiposOrden"
    android:entryValues="@array/tiposOrdenValores"
    android:defaultValue="1"/>
```

Asegúrate de que también tienes la opción **Máximo de lugares a mostrar**. El resultado final ha de ser similar a:



2. Crea la siguiente clase para acceder de forma sencilla al valor de las preferencias:

```
public class Preferencias {
   private static final Preferencias INSTANCIA = new Preferencias();
   private SharedPreferences pref;
   public final static int SELECCION_TODOS =0;
   public final static int SELECCION_MIOS =1;
   public final static int SELECCION_TIPO =2;

   public static Preferencias getInstance() { return INSTANCIA;}

   private Preferencias() {}
```

```
   public void inicializa(Context contexto){
      pref = PreferenceManager.getDefaultSharedPreferences(contexto);
   }

   public int criterioSeleccion() {
      return parseInt(pref.getString("seleccion", "0"));
   }

   public String tipoSeleccion() {
      return (pref.getString("tipo_seleccion", "BAR"));
   }

   public String criterioOrdenacion() {
      return (pref.getString("orden", "valoracion"));
   }

   public int maximoMostrar() {
      return parseInt(pref.getString("maximo", "50"));
   }
}
```

3. Reemplaza el fichero res/values/arrays.xml por el siguiente:

```
<resources>
   <string-array name="tiposOrden">
      <item>Valoración</item>   <item>Fecha</item>   <item>Tipo</item>
   </string-array>
   <string-array name="tiposOrdenValores">
      <item>valoracion</item>   <item>fecha</item>   <item>tipo</item>
   </string-array>
   <string-array name="tiposSeleccion">
      <item>Todos</item> <item>Creados por mi</item> <item>Por tipo</item>
   </string-array>
   <string-array name="tiposSeleccionValores">
      <item>0</item>     <item>1</item>              <item>2</item>
   </string-array>
</resources>
```

4. En la clase `SelectorFragment` crea el método estático `ponerAdaptador()` y mueve el código tal y como se indica:

```
@Override public void onActivityCreated(Bundle state) {
   super.onActivityCreated(state);
   RecyclerView.LayoutManager layoutManager =
               new LinearLayoutManager(getContext());
   recyclerView.setLayoutManager(layoutManager);
   // Mover el resto del código al siguiente método
   context = getContext();
   ponerAdaptador();
}

public static void ponerAdaptador() {
      Preferencias pref = Preferencias.getInstance();
      pref.inicializa(context);
   // Poner el código aquí
}
```

Esto nos permitirá cambiar el adaptador, cuando nos interese.

5. Añade en la declaración de variables el código subrayado:

```
private static RecyclerView recyclerView;
private static Context context;
```

6. Añade en la clase `PreferenciasActivity` el siguiente método:

```
@Override public void onDestroy() {
   super.onDestroy();
   SelectorFragment.ponerAdaptador();
}
```

7. De esta forma, cuando un usuario salga de las preferencias, se volverá a obtener el adaptador.

8. Una vez preparada la configuración, la aplicas dentro de `ponerAdaptador()`:

```
Query query = FirebaseFirestore.getInstance()
                  .collection("lugares")
                  .orderBy(pref.criterioOrdenacion())
                  .limit(pref.maximoMostrar());
switch (pref.criterioSeleccion()){
   case SELECCION_MIOS:
      query = query.whereEqualTo("creador",
                     FirebaseAuth.getInstance().getCurrentUser().getUid());
      break;
   case SELECCION_TIPO:
      query = query.whereEqualTo("tipo", pref.tipoSeleccion());
      break;
}
```

9. En caso de usar Realtime Database con FirebaseUI, el código sería:

```
com.google.firebase.database.Query query = FirebaseDatabase.getInstance()
        .getReference().child("lugares")
        .limitToLast(pref.maximoMostrar());
switch (pref.criterioSeleccion()) {
   case SELECCION_MIOS:
      query = query.orderByChild("creador")
          .equalTo(FirebaseAuth.getInstance().getCurrentUser().getUid());
      break;
   case SELECCION_TIPO:
      query = query.orderByChild("tipo").equalTo(pref.tipoSeleccion());
      break;
   default:
      query = query.orderByChild(pref.criterioOrdenacion());
      break;
}
```

NOTA: Reemplaza "creador" por el nombre que le diste a este campo.

En este caso solo podemos usar `equalTo()` si hemos ordenado por este campo. Por lo tanto, algunas configuraciones no van a ser posibles. Para

resolverlo, aplicaremos el criterio de ordenación solo si se ha escogido TODOS como criterio de selección.

En caso de Realtime Database con adaptador personalizado, no se ha previsto la posibilidad de hacer consultas.

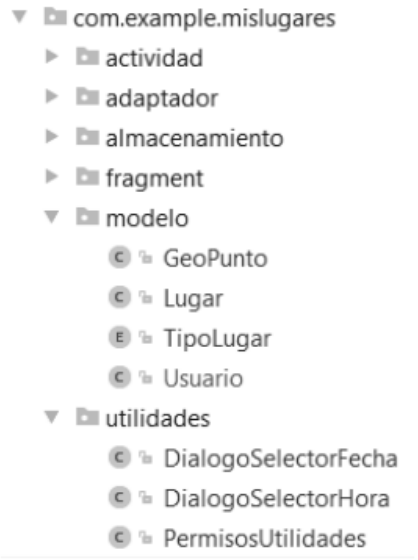
NOTA: En un próximo ejercicio se sugiere como poder seleccionar desde las preferencias los tipos base de datos y los adaptadores.

2.3.5. Organizar y seleccionar las clases

Ejercicio: Organizar las clases en paquetes

El número de clases del proyecto es muy elevado, por lo que resulta complicado localizarlas. Para resolver este problema, resulta frecuente organizar las clases en diferentes paquetes. Podemos usar diferentes criterios, por ejemplo por módulos del proyecto (como autentificación, visualización, mapas…) Otro criterio podría ser por entidades (como lugares, usuarios…) En este ejercicio utilizaremos como criterio la función de la clase (como actividades, adaptadores…).

En concreto, la organización propuesta se muestra a continuación:



Pero eres libre de usar nombres en inglés o definir tu propio criterio.

En el paquete `modelo` suelen ponerse las clases que representan la lógica interna de la aplicación (también conocida como lógica de negocio). Las clases de este paquete se conocen como POJO (Plain Old J ava Object) al tratarse de clases Java puras. No es conveniente que en estas clases se utilicen API externos. Si abres las cuatro clases definidas en este paquete, podrás comprobar que ninguna necesita un `import`.

En el paquete `almacenamiento` estarían las clases encargadas de guardar de forma permanente o acceder a ellos. Suelen representar bases de datos, servicios

Web, preferencias, ficheros JSON/XML… También es conocida como capa de datos. Otros nombres frecuentes para este paquete serían `datos`, `storage` o `persistence`.

1. Pulsa con el botón derecho sobre `com.example.mislugares` y selecciona New / Package. Escribe `actividad`. Se creará `com.example.mislugares.actividad`.
2. Arrastra todas las clases terminadas en `Activity` a este paquete. El proceso de refactorización se realiza de forma bastante automática. Aunque en algunos casos tendrás que realizar algún pequeño ajuste, como hacer público algún método.
3. Repite este proceso para adaptadores, fragments…

Ejercicio: Permitir simultáneamente varias configuraciones

Queremos poder utilizar en la aplicación bases de datos Realtime Database y Cloud Firestore, de forma que desde una preferencia, podamos cambiar la configuración. Además, para cada una de estas dos alternativas hemos creado dos posibles adaptadores, que queremos también poder seleccionar desde una preferencia. Para conseguirlo sigue los siguientes pasos:

1. Añade las siguientes referencias a res/xml/preferencias.xml:

```
<CheckBoxPreference
   android:key="firestore"
   android:title="Usar Firestore"
   android:summary="Usar Cloud Firestore en lugar de Realtime Database"/>
<CheckBoxPreference
   android:key="firebaseUI"
   android:title="Usar FirebaseUI"
   android:summary="Utilizar FirebaseUI en lugar de directamente el SDK"/>
```

2. Añade a la clase `Preferencias` los siguientes métodos:

```
public boolean usarFirestore() {
   return (pref.getBoolean("firestore", true));
}

public boolean usarFirebaseUI() {
   return (pref.getBoolean("firebaseUI", true));
}
```

3. En `onCreate()` de `MainActivity` inicializa lugares de la siguiente forma:

```
Preferencias pref = Preferencias.getInstance();
pref.inicializa(this);
if (pref.usarFirestore()) {
   lugares = new LugaresFirestore();
} else {
   lugares = new LugaresFirebase();
}
```

4. Seleccionar el adaptador va a ser algo más complicado. En `SelectorFragment`, declara adaptador de la siguiente forma:

```
public static RecyclerView.Adapter adaptador;
```

5. Tras este cambio verás que aparecen muchos errores, dado que métodos como `getKey()`, `startListening()` no pertenecen a `RecyclerView.Adapter`. El problema es que en Java no existe la herencia múltiple.

   Para resolverlo, crea la siguiente interfaz:

```
public interface AdaptadorLugaresInterface {
   public String getKey(int pos);
   public Lugar getItem(int pos);
   public void setOnItemClickListener(View.OnClickListener onClick);
   public void startListening();
   public void stopListening();
}
```

   En ella hemos añadido todos los métodos que necesitamos y no están en `RecyclerView.Adapter`.

6. Haz que todos los adaptadores para Firebase que has creado implementen esta interfaz.

7. Añade el siguiente método en `SelectorFragment`:

```
public static AdaptadorLugaresInterface getAdaptador() {
   return (AdaptadorLugaresInterface) adaptador;
}
```

8. En todos los sitios donde se produzca el error reemplaza `adaptador` por `getAdaptador()`.

9. En el método `ponerAdaptador()` de `SelectorFragment`, reemplaza la inicialización de `adaptador` por:

```
if (pref.usarFirestore()) {
   com.google.firebase.firestore.Query query =
                                       FirebaseFirestore.getInstance()
        .collection("lugares")
        .orderBy(pref.criterioOrdenacion())
        .limit(pref.maximoMostrar());
   switch (pref.criterioSeleccion()){
      case SELECCION_MIOS:
         query = query.whereEqualTo("creador", FirebaseAuth.getInstance()
                     .getCurrentUser().getUid());
```

```
            break;
         case SELECCION_TIPO:
            query = query.whereEqualTo("tipo", pref.tipoSeleccion());
            break;
      }
      if (pref.usarFirebaseUI()) {
         FirestoreRecyclerOptions<Lugar> options =
              new FirestoreRecyclerOptions.Builder<Lugar>()
                       .setQuery(query, Lugar.class)
                       .build();
         adaptador = new AdaptadorLugaresFirestoreUI(options);
      } else {
         adaptador = new AdaptadorLugaresFirestore(context, query);
      }
   } else {
      if (pref.usarFirebaseUI()) {
         com.google.firebase.database.Query query = FirebaseDatabase
               .getInstance().getReference().child("lugares")
               .limitToLast(pref.maximoMostrar());
         switch (pref.criterioSeleccion()) {
            case SELECCION_MIOS:
               query = query.orderByChild("creador").equalTo(
                       FirebaseAuth.getInstance().getCurrentUser().getUid());
               break;
            case SELECCION_TIPO:
               query = query.orderByChild("tipo")
                                                .equalTo(pref.tipoSeleccion());
               break;
            default:
               query = query.orderByChild(pref.criterioOrdenacion());
               break;
         }
         FirebaseRecyclerOptions<Lugar> options =
                       new FirebaseRecyclerOptions.Builder<Lugar>()
                               .setQuery(query, Lugar.class)
                               .build();
         adaptador = new AdaptadorLugaresFirebaseUI(options);
      } else {
         adaptador = new AdaptadorLugaresFirebase(context, FirebaseDatabase
                                    .getInstance().getReference("lugares"));
      }
   }
   getAdaptador().setOnItemClickListener(new View.OnClickListener() {
      @Override public void onClick(View v) {
         ((MainActivity) context).muestraLugar(
                 recyclerView.getChildAdapterPosition(v));
      }
   });
   getAdaptador().startListening();
   recyclerView.setAdapter(adaptador);
```

En caso de que no hayas implementado algún adaptador, elimina el código oportuno.

10. Añade en la clase `PreferenciasActivity` el siguiente código:

```
@Override public void onDestroy() {
   super.onDestroy();
   Preferencias pref = Preferencias.getInstance();
   if (pref.usarFirestore()) {
      MainActivity.lugares = new LugaresFirestore();
   } else {
      MainActivity.lugares = new LugaresFirebase();
   }
   SelectorFragment.ponerAdaptador();
}
```

11. Modifica el método `Usuarios.guardarUsuario()` para que el usuario se guarde según la base de datos seleccionada.

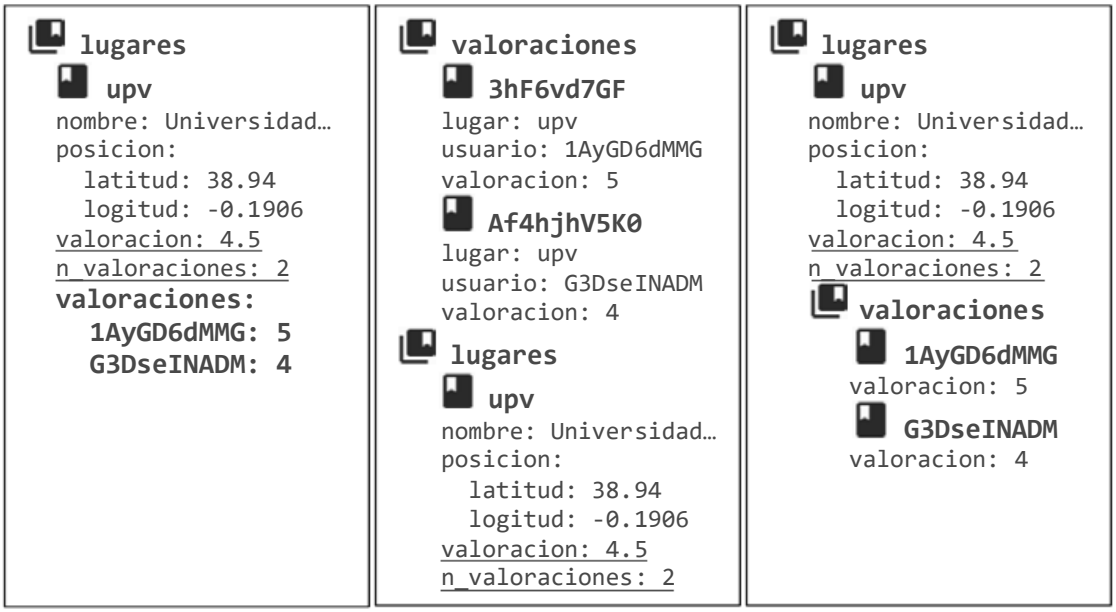
12. Verifica el funcionamiento.

2.3.6. Trabajar con diferentes colecciones

En este apartado vamos a tratar de realizar acciones algo más complejas, donde tengamos que trabajar con varias colecciones. En concreto queremos registrar las valoraciones que cada usuario hace de un lugar.

La forma en la queremos gestionar las valoraciones es la siguiente. En el `RecyclerView` se mostrará la valoración media del lugar, pero cuando un usuario entre en la actividad para ver los detalles del lugar, se mostrará la puntuación realizada por él (o se indicará que no ha valorado). Desde esta actividad se podrá valorar o cambiar la valoración.

Guardaremos en cada documento de la colección `lugares` la valoración media calculada hasta el momento (campo `valoracion`) y el número de valoraciones realizadas (campo `n_valoraciones`). Este último nos será útil para realizar los cálculos. Para almacenar las valoraciones de cada usuario, podemos plantear varias alternativas:



a) b) c)

Práctica: Elegir estructura de almacenamiento

Teniendo en cuenta que queremos acceder a los datos según lo que acabado de describir. ¿Cuál sería la estructura de almacenamiento más eficiente? ¿En qué circunstancias sería más interesante usar cada alternativa?

a) Es la estructura que menos tamaño requiere, pero presenta dos inconvenientes: no podemos hacer búsquedas y el tamaño del documento `lugares` aumenta al aumentar las valoraciones. Interesante si el número de valoraciones se mantiene siempre en un valor acotado.

b) Es posiblemente la estructura de almacenamiento más versátil, al permitir hacer búsquedas por cualesquiera de sus campos. Como inconveniente estaría que almacenamos más información que en a) y c). Interesante si queremos obtener rápidamente todas las valoraciones de un usuario; si queremos hacer consultas sobre estos campos (ej. obtén todos los lugares que un usuario ha valorado con 5) o si queremos conocer el orden cronológico en que se realizan las valoraciones (usaríamos la clave que siempre se genera de forma cronológica).

c) No presenta el problema del aumento de tamaño del documento que hemos visto en a). Además, ocupa algo menos que b). Como permite realizar todas las operaciones que necesitamos, sería la opción escogida.

Ejercicio: Añadir valoraciones de usuarios

Vamos a tratar de implementar el esquema de almacenamiento descrito en la propuesta c) de la práctica anterior. Recuerda que en el `RecyclerView` se mostrará la valoración media del lugar. Pero cuando un usuario entre en los detalles del lugar, se mostrará la puntuación de este usuario y se le permitirá cambiarla.

1. Abre la clase `VistaLugarFragment`. En el método `actualizarVistas()` reemplaza las acciones a realizar cuando cambia el `RaitingBar`:

```
valoracion.setOnRatingBarChangeListener(
        new RatingBar.OnRatingBarChangeListener() {
           @Override
           public void onRatingChanged(RatingBar ratingBar,
                                       float valor, boolean fromUser) {
              String _id = getAdaptador().getKey((int) id);
              String usuario = FirebaseAuth.getInstance().getUid();
              guardarValoracion(_id, usuario, ((double) valor));
              lugar.setValoracion(valor);
              actualizaLugar();
           }
        });
```

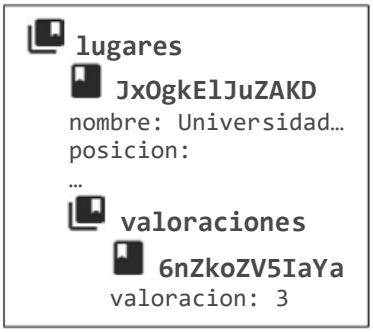
En lugar de cambiar la valoración del lugar, vamos a obtener el identificador de lugar y usuario, y vamos a llamar a un nuevo método que almacenará la valoración.

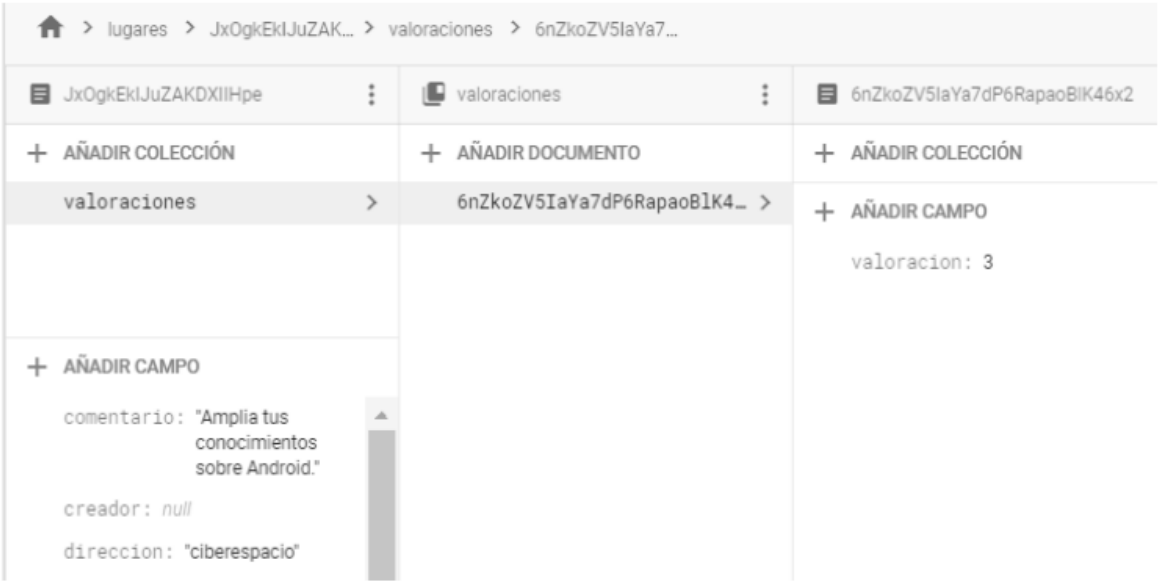
2. Añade la siguiente clase:

```
public class ValoracionesFirestore {
   public static void guardarValoracion(String lugar, String usuario,
                                                          Double valor) {
      FirebaseFirestore db = FirebaseFirestore.getInstance();
      Map<String, Object> datos = new HashMap<>();
      datos.put("valoracion", valor);
      db.collection("lugares").document(lugar).collection("valoraciones")
             .document(usuario).set(datos);
   }
}
```

Dada la simplicidad de los datos almacenados, en este caso no hemos creado un POJO. Para indicar los datos, usamos un `Map`.

3. Ejecuta y realiza una valoración. Abre la consola de Firebase y verifica el resultado:





Si entras a ver el lugar anterior, podrás comprobar que la valoración se sigue obteniendo del lugar. Vamos a leer la valoración del usuario para mostrarla en esta actividad. En la clase `ValoracionesFirestore` añade este código:

```
public interface EscuchadorValoracion {
   void onRespuesta(Double valor);
   void onNoExiste();
   void onError(Exception e);
}

public static void leerValoracion(String lugar, String usuario,
                                  final EscuchadorValoracion escuchador) {
   FirebaseFirestore db = FirebaseFirestore.getInstance();
   db.collection("lugares").document(lugar)
      .collection("valoraciones").document(usuario).get()
      .addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
         @Override
         public void onComplete(@NonNull Task<DocumentSnapshot> task) {
            if (task.isSuccessful()) {
               if (!task.getResult().exists()) {
                  escuchador.onNoExiste();
               } else {
                  double val = task.getResult().getDouble("valoracion");
                  escuchador.onRespuesta(val);
               }
            } else {
               Log.e("Mis Lugares", "No se puede leer valoraciones",
                                                    task.getException());
               escuchador.onError(task.getException());
            }
         }
      });
}
```

Este método permite leer la valoración que un usuario ha hecho sobre un lugar. El resultado se devuelve de forma asíncrona utilizando la interfaz `EscuchadorValoracion`. Se llama a uno de los tres métodos según el usuario haya realizado una valoración, todavía no la haya realizado o se produzca un error.

4. En la clase `VistaLugarFragment`, en el método `actualizarVistas()`, reemplaza las acciones para inicializar el `RaitingBar` por:

```
final RatingBar valoracion = (RatingBar) v.findViewById(R.id.valoracion);
final String _id = getAdaptador().getKey((int) id);
final String usuario = FirebaseAuth.getInstance().getUid();
leerValoracion(_id, usuario, new ValoracionesFirestore
                                                .EscuchadorValoracion() {
   @Override public void onNoExiste() {
      this.onRespuesta(0.0);
   }
   @Override public void onRespuesta(Double valor) {
      valoracion.setOnRatingBarChangeListener(null);
```

```
        valoracion.setRating(valor.floatValue());
        valoracion.setOnRatingBarChangeListener(
                new RatingBar.OnRatingBarChangeListener() {
                    @Override
                    public void onRatingChanged(RatingBar ratingBar,
                                                float valor, boolean fromUser) {
                        guardarValoracion(_id, usuario, (double) valor);
                    }
                });
    }
    @Override public void onError(Exception e) {}
});
```

Observa cómo ponemos `final` en las variables para poder acceder desde los objetos anónimos. Miramos si el usuario actual ya ha valorado el lugar. Si no es así, ponemos la valoración a 0 y llamamos a `onRespuesta()` para que se muestre este valor. Sería interesante mostrar un `TextView` diciendo "Por favor, valora este lugar". Si ya hay una valoración la ponemos en el `RaitingBar` y activamos un escuchador de forma similar a como lo hacíamos en el punto anterior.

5. Ejecuta la aplicación y verifica que ahora en `VistaLugarFragment`, se muestra la valoración hecha por el usuario.

Ejercicio: Calcular la valoración media de los usuarios

En la versión actual, en el `RecyclerView` se muestra la valoración inicial del lugar. Esta valoración podría ser calculada periódicamente desde el servidor según la media de las valoraciones de los usuarios. En este ejercicio trataremos que esta media se recalcule cada vez que un usuario realiza una valoración.

1. Abre la clase `Lugar` y añade la propiedad `n_valoraciones`:

```
public class Lugar {
    …
    private float valoracion;
    private long n_valoraciones;
```

2. En el constructor `Lugar()`, inicialízala a cero.
3. Añade los getters y setters correspondientes (Generate… / getters and setter).
4. En la clase `ValoracionesFirestore` añade este código:

```
public static void guardarValoracionYRecalcular(final String lugar,
                            final String usuario, final float nuevaVal) {
    leerValoracion(lugar, usuario, new EscuchadorValoracion() {
        @Override public void onRespuesta(Double viejaVal) {
            actualizarValoracionMedia(lugar, usuario, viejaVal, nuevaVal);
        }
        @Override public void onNoExiste() {
            actualizarValoracionMedia(lugar, usuario, Double.NaN,nuevaVal);
        }
```

```
      @Override public void onError(Exception e) {}
   });
}

private static void actualizarValoracionMedia(final String lugar,
      final String usuario, final double viejaVal, final double nuevaVal){
   FirebaseFirestore db = FirebaseFirestore.getInstance();
   final DocumentReference refLugar=db.collection("lugares").document(lugar);
   refLugar.get().addOnCompleteListener(
      new OnCompleteListener<DocumentSnapshot>() {
         @Override
         public void onComplete(@NonNull Task<DocumentSnapshot> task) {
            if (task.isSuccessful() && task.getResult().exists()) {
               double media = task.getResult().getDouble("valoracion");
               long nValor = task.getResult().getLong("n_valoraciones");
               double nuevaMedia = nuevaMedia(media, nValor, viejaVal,
                                                            nuevaVal);
               if (Double.isNaN(viejaVal)) nValor++;
               refLugar.update("valoracion",nuevaMedia, "n_valoraciones",
                                                              nValor);
               guardarValoracion(lugar, usuario, nuevaVal);
            } else {
               Log.e("Mis Lugares", "ERROR al leer", task.getException());
            }
         }
      });
}

private static Double nuevaMedia(double media, long nValoraciones,
                                       double viejaVal, double nuevaVal){
   if (nValoraciones==0){ //No existe ninguna valoración
      return nuevaVal;
   } else if (Double.isNaN(viejaVal)){ //No existe valoración anterior
      return (nValoraciones*media + nuevaVal) / (nValoraciones+1);
   } else {   //El usuario cambia su valoración
      return (nValoraciones*media - viejaVal + nuevaVal)/ nValoraciones;
   }
}
```

El primer método comienza mirando si ya hay una valoración. Según el resultado, llama a `actualizarValoracionMedia()` indicando en `viejaVal` la valoración o, de no haberla, `NaN`. Se trata de un valor especial que puede tomar un `double`.

El nuevo método lee de nuevo el lugar obteniendo los valores de los campos `valoracion` y `n_valoraciones`. Con esta información, más los valores la nueva y vieja valoración del usuario, llamamos a `nuevaMedia()`. Existen tres alternativas para calcular la nueva valoración media. Si analizas cada caso por separado, no resulta muy complejo de entender.

Finalmente, se guardan los nuevos valores para los campos `valoracion` y `n_valoraciones` del lugar y la puntuación del usuario en la nueva colección.

5. En la clase `VistaLugarFragment`, reemplaza la llamada a `guardarValoracion()` por `guardarValoracionYRecalcular()`.

6. Ejecuta la aplicación y verifica que la valoración mostrada en la lista corresponde a la media. Si lo haces en un lugar donde el campo `n_valoraciones` no exista, es posible que se produzca un error. En este caso introduce el nuevo campo desde la consola manualmente con valor 0. Tendrás que usar varios usuarios para obtener la media de varias valoraciones.

2.3.7. Operaciones atómicas

Si analizamos con cuidado el ejercicio realizado en el punto anterior, observaremos que en ciertas ocasiones podría ejecutarse incorrectamente. Supongamos que dos usuarios van a puntuar el mismo lugar de forma casi simultánea. Ambos leen en número de valoraciones y obtienen el valor 10. Ambos incrementan este valor en uno y lo escriben. Al final el valor guardado sería 11, cuando realmente tendría que ser 12.

Para resolver este problema, se utilizan las operaciones atómicas. Cloud Firestore dispone de dos tipos de operaciones atómicas: transacciones y escrituras por lotes.

2.3.7.1. Transacciones

Una transacción es un conjunto de operaciones de lectura y escritura en uno o más documentos que se ejecuta como una operación atómica. Las transacciones nunca aplican escrituras de forma parcial. Todas las escrituras se ejecutan al final de una transacción correcta. Veamos un ejemplo:

Ejercicio: Calcular la valoración media con una transacción

Como acabamos de comentar, el ejercicio anterior podría realizar mal las operaciones en caso de varias valoraciones simultáneas. Para resolver el problema puedes utilizar una transacción.

1. Para conseguir que el método ejercicio `actualizarValoracionMedia()` anterior se ejecute como una transacción, reemplaza el código por el siguiente.

```
private static void actualizarValoracionMedia(final String lugar, final
           String usuario, final double viejaVal, final double nuevaVal) {
   FirebaseFirestore db = FirebaseFirestore.getInstance();
   final DocumentReference ref = db.collection("lugares").document(lugar);
   db.runTransaction(new Transaction.Function<Void>() {
      @Override
      public Void apply(Transaction transaction)
                                    throws FirebaseFirestoreException {
         DocumentSnapshot snapshot = transaction.get(refLugar);
         double media = snapshot.getDouble("valoracion");
         long nValoraciones = snapshot.getLong("n_valoraciones");
         double nuevaMedia =nuevaMedia(media,nValoraciones,viejaVal,nuevaVal);
         if (viejaVal == Double.NaN) nValoraciones++;
         transaction.update(refLugar, "valoracion", nuevaMedia,
                                     "n_valoraciones", nValoraciones);
```

```
            //Escribimos valoración
            Map<String, Object> datos = new HashMap<>();
            datos.put("valoracion", nuevaVal);
            transaction.set(db.collection("lugares").document(lugar)
                        .collection("valoraciones").document(usuario), datos);
            return null;
        }
    });
}
```

2. Si lo pruebas en un lugar donde no exista en campo `n_valoraciones`, dará un error. Para evitarlo crea este campo manualmente desde la consola con valor 0.

No modifiques el estado de la aplicación dentro de las funciones de transacción. Esto generará problemas de simultaneidad, debido a que las funciones de transacción pueden ejecutarse varias veces y no se garantiza que se ejecuten en el procesamiento de IU. En vez de esto, pasa la información que necesites desde tus funciones de transacción. El siguiente ejercicio se basa el anterior para mostrarte cómo devolver información desde una transacción.

Ejercicio: Transacción con devolución de resultado

1. Añade el siguiente código:

```
private static void actualizarValoracionMedia(final String lugar, final
          String usuario, final double viejaVal, final double nuevaVal) {
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    final DocumentReference ref = db.collection("lugares").document(lugar);
    db.runTransaction(new Transaction.Function<Double>() {
        @Override
        public Double apply(Transaction transaction)
                                        throws FirebaseFirestoreException {
            …
            return nuevaMedia;
        }
    }).addOnSuccessListener(new OnSuccessListener<Double>() {
        @Override public void onSuccess(Double result) {
            Log.d("Mis Lugares", "Transaccion OK. Nueva media : " + result);
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override public void onFailure(@NonNull Exception e) {
            Log.w("Mis Lugares", "Transaccion errónea.", e);
        }
    });
}
```

2. Ejecuta la aplicación y verifica que la salida de la transacción es mostrada en el Log.

2.3.7.2. Escrituras por lotes

En caso de querer hacer una operación atómica y no necesites leer documentos, Firestore nos ofrece esta operación.

Una escritura por lotes es un conjunto escrituras (añadir, modificar o borrar) en uno o más documentos, ejecutadas de forma atómica. Podemos enviar al servidor hasta 500 operaciones de escritura de forma conjunta. Las escrituras por lotes se pueden lanzar incluso cuando el dispositivo del usuario está sin conexión. Veamos un ejemplo:

```
FirebaseFirestore db = FirebaseFirestore.getInstance();
DocumentReference ref;
WriteBatch lote = db.batch();

ref = db.collection("usuarior").document("nuevoUsuario");
lote.set(ref, new Usuario());

ref = db.collection("usuarios").document("otroUsuario");
lote.update(ref, "correo", "correo@upv.es");

ref = db.collection("usuarios").document("viejoUsuario");
lote.delete(ref);

lote.commit().addOnCompleteListener(new OnCompleteListener<Void>() {
   @Override
   public void onComplete(@NonNull Task<Void> task) {
      // ...
   }
});
```

Como puedes ver, una escritura por lotes resulta muy sencilla de realizar. El código no necesita mayores explicaciones.

2.3.8. Reglas de acceso

Al crear la base de datos permitimos acceso de lectura y escritura a todo el mundo. En una aplicación real esto ocasionaría un importante problema de seguridad. Podemos configurar quién puede acceder a los datos y qué operaciones puede realizar configurando las reglas de acceso. Para ello entra en la pestaña Database / Cloud Firestore / **Reglas**. Podrás comprobar que actualmente permitimos lectura y escritura a todo el mundo:

Un fichero de reglas más realista tendría la siguiente estructura:

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /lugares/* {       //solo para documentos en colección lugares
      allow read;            //permitimos lectura
      allow write: if …      //permitimos escritura si se cumple condición

      match /valuraciones/** //para documentos que cuelgan de valoraciones
        allow …
      }
    }

    match /usuarios/{usuario} { //para documentos en colección usuarios
      allow read, write: if …   //permitimos lectura y escritura si …
    }                           // usaremos $(usuario) en la condición
  }                             // para indicar el nombre del documento

    match /notas/{nota=**} {    //para documentos que cuelgan de notas
      allow …
    }
  }
}
```

Existen dos acciones principales `read` y `write`. Pero podemos especificar otras más concretas:

`get`	Permitimos operación `get` (obtener documento)
`list`	Permitimos `where().get()` (hacer consultas)
`read`	Permitimos `get` y `list`
`create`	Permitimos `set()` y `add()` (añadir)
`update`	Permitimos `set()` y `update()` (modificar)
`delete`	Permitimos `remove()` (borrar)
`write`	Permitimos `create`, `update` y `delete`

Veamos un ejemplo:

```
match /lugares/* {
    allow read, create; //todos: leer doc., consultas y crear
    allow update: if …  //permitimos modificar si …
}                       //resto de acciones prohibidas (delete)
```

La variable `request` contiene información sobre la solicitud que viene desde el cliente. Es un mapa con diferentes campos. Uno es `auth`, con información sobre el usuario que accede, obtenida de Firebase Authentication. Otro campo interesante es `resource`, que contiene información sobre el documento que se envía. En la siguiente tabla se muestran estos y otros campos de `request`.

`auth`: información de Firebase Authentication del usuario que accede:

```
request.auth.uid
request.auth.email
request.auth.email_verified …
```

`resource`: documento que se envía

- `request.resource.data.correo` valor del campo correo
- `request.resource.data.size()` número de campos del docum
- `request.resource.data.keys()` lista con los campos del docum

`time`: cuándo se envía

`query`: consulta que se envía

- `request.limit`
- `Request.orderBy …`

`writeFields`: campos que se pide actualizar

Veamos algunos ejemplos:

```
match /users/{userID}/notas/{nota} {
    allow read: if request.auth.uid != null;
    allow write: if request.auth.uid == userID;
}
```

Cualquier usuario, si está autentificado, puede leer una nota. Solo el usuario al que pertenece la nota puede escribir.

```
match /usuarios/{userID} {
   allow write: if request.resource.data.keys().hasAll(
                           ['nombre', 'correo', 'inicioSesion'])
               && request.resource.data.size() == 3
               && request.resource.data.nombre is string
               && request.resource.data.correo is string
               && request.resource.data.inicioSesion is int;
}
```

Solo permitimos crear o cambiar los datos de un usuario si se escriben tres campos, con los nombres indicados y de los tipos indicados.

```
match /usuarios/{userID} {
   allow write: if request.resource.data.inicioSesion <= request.time;
}
```

El campo inicio de sesión ha de ser anterior a la solicitud.

La variable `resource` contiene información sobre el documento almacenado antes de ser escrito. No confundir con `request.resource`. Veamos algunos ejemplos:

```
match /lugares/{lugarID} {
   allow read, create;
   allow update: if request.resource.data.nombre == resource.data.nombre
                 && resource.resource.data.creador == request.auth.creador
}
```

Todo el mundo puede leer o crear un lugar. Solo se puede actualizar si conservamos el campo `nombre` y `creador`.

```
match /lugares/{lugarID}/valoraciones/{userID} {
   allow read;
   allow write: if request.resource.data.valoracion > resource.data.valoracion
               || userID == request.auth.uid;
}
```

Todo el mundo puede leer una valoración. Cualquiera puede escribir siempre que aumente la `valoracion` o si es el usuario que creó la valoración.

Además de consultar el documento al que se va a acceder (usando `resource`), podemos consultar otros documentos utilizando `get()` o `exist()`.

```
match /lugares/{lugarID}/valoraciones/{userID} {
   allow read;
   allow write: if get(/databases/$(database)/documents
                /lugares/$(lugarID)).data.creador != userID;
}
```

Todo el mundo puede leer una valoración. Se puede escribir siempre que la valoración no la haga el creador del lugar.

```
match /lugares/{lugarID} {
   allow write: if exist(/databases/$(database)/documents
                /lugares/$(lugarID)/administradores/$(request.auth.uid));
}
```

Se puede escribir en un lugar si existe un documento en la subcolección `administradores` con nombre igual al usuario autentificado.

NOTA: get() y exists() solo son para documentos, no para colecciones.

Cuando una condición va a ser escrita varias veces, podemos utilizar una función:

```
match /lugares/{lugarID} {
   function puedeAccion() {
        return (  request.auth.uid == resource.resource.data.creador
               && request.auth.uid == resource.data.creador)
               || get(/databases/$(database)/documents
                /usuarios/$(request.auth.uid)).data.tipo == "admin";
      }
   allow update: if puedeAccion();

   match /{cualquierHijo=**} {
        allow delete: if puedeAccion();
   }
}
```

Permitimos actualizar un lugar si somos el creador y no intentamos cambiar el campo `creador`; o si en la colección `usuarios` verificamos que somos de tipo `admin`. Si cumplimos esta condición, también vamos a poder borrar cualquier documento que cuelgue de este lugar.

Práctica: Configurar las reglas de acceso en "Mis lugares"

Configura las reglas de acceso en la base de datos Firestore de "Mis Lugares" para que se verifique:

Todo el mundo podrá leer o crear un lugar. Solo se podrá actualizar un lugar si somos el creador del lugar y además no cambiamos el campo creador. No obstante, los campos `valoracion` y `nValoraciones` podrán ser actualizados por cualquier usuario, siempre que esté validado. No está permitido borrar lugares. El creador de un lugar no puede valorar sus propios lugares.

Solo podrá leer o escribir en las colecciones `usuario` y una `valoracion` el mismo usuario que creó el documento.

2.3.9. Trabajar con datos sin conexión

En un dispositivo móvil es muy frecuente que el acceso a Internet no esté siempre garantizado. Para resolver este problema, Firestore nos ofrece una interesante característica. Cuando un usuario pierda la conexión a Internet, podrá seguir usando Firestore a través de una caché local. En esta caché están los datos leídos recientemente. Las escrituras también se realizarán en la caché. Cuando el dispositivo vuelva a estar en línea, todos los cambios locales serán sincronizados. Además, esta característica se lleva a cabo de forma transparente al usuario y al programador.

Trabajar sin conexión solo está disponible en Android, iOS y Web. En Android e iOS está activada por defecto. En caso de quererla desactivar, utiliza:

```
FirebaseFirestoreSettings settings =
    new FirebaseFirestoreSettings.Builder()
        .setPersistenceEnabled(true)
        .build();
db.setFirestoreSettings(settings);
```

En ciertas operaciones vas a querer asegurarte que los datos están sincronizados. Para comprobar si los datos son del servidor o la caché, utiliza:

```
snapshot.getMetadata().isFromCache()
```

CAPÍTULO 3.

Mensajes y almacenamiento en la nube

VICENTE CARBONELL

La creación de aplicaciones basadas en la nube es un tema muy a tener en cuenta, nos ofrece una serie de posibilidades que no podemos desarrollar mediante una aplicación que se ejecute de forma local en un dispositivo o que son muy costosas de hacer en un entorno cliente-servidor. En este capítulo nos centraremos en dos de los servicios en la nube más importantes: el envío de mensajes entre aplicaciones y el almacenamiento.

Comenzaremos estudiando Firebase Cloud Messaging, que nos permitirá el envío de mensajes a nuestras aplicaciones. Veremos que estos mensajes podrán ser enviados tanto desde la consola de Firebase como desde un servidor propio.

La segunda parte del capítulo trata sobre el almacenamiento de ficheros en la nube. Veremos dos alternativas: Firebase Storage, que ofrece servicios orientados a las aplicaciones, y Google Drive, más centrado en el usuario final.

Google ofrece diferentes servicios en la nube desde distintas consolas. Podemos crear un proyecto en una de las consolas y será visible desde el resto. Veremos la consola de Firebase, para los servicios Firebase Cloud Messaging y Firebase Storage, y la consola de Google API, para el servicio Google Drive.

Objetivos

- Aprender los principios de Firebase Cloud Messaging (FCM).

- Diferenciar los diferentes tipos de mensajes de FCM y conocer cómo son recibidos en la aplicación móvil.
- Enviar mensajes desde la consola y desde un servidor propio.
- Almacenar datos en la nube con Firebase Storage.
- Introducción al uso de Google Drive en Android.

3.1. Mensajes en la nube

En algunas ocasiones, nuestras aplicaciones móviles necesitan contar con la capacidad de notificar al usuario, o al terminal, determinados eventos que ocurren fuera del dispositivo. Normalmente, el origen es una aplicación web o servicio online alojado en un servidor externo. Como ejemplo de esto podríamos pensar en los mensajes que nos muestra nuestro móvil cuando se recibe un nuevo mensaje de WhatsApp o un correo electrónico con la aplicación de Gmail.

El destinatario de un mensaje puede ser el usuario final, en este caso se utilizará una notificación de la barra del sistema. A este tipo de mensajes los llamaremos mensajes de notificación. En otras ocasiones nos va a interesar que la información llegue solo a la aplicación, (por ejemplo, avisar que tiene que sincronizar la base de datos). A este tipo de mensajes los llamaremos mensajes de datos.

Para implementar los mensajes en la nube se han propuesto varias soluciones. Una primera alternativa consiste en mantener abierta una conexión permanente con el servidor, de forma que este le pudiera comunicar inmediatamente cualquier nuevo evento a nuestra aplicación. Esta técnica, aunque es viable, no es efectiva.

Una segunda solución utilizada habitualmente sería hacer que nuestra aplicación móvil revise de forma periódica, en el servidor, si existe alguna novedad que notificar al usuario. Esto se denomina polling, y requiere muchos menos recursos que la opción anterior. Sin embargo, tiene un inconveniente, que puede ser importante según el objetivo de nuestra aplicación: cualquier evento que se produzca en el servidor no se notificará al usuario hasta la próxima consulta al servidor que haga la aplicación cliente, que podría ser mucho tiempo después y carecer, así, de interés.

La tercera solución serían los mensajes o notificaciones push. En esta solución es el servidor el que inicia el proceso de notificación, por lo que puede realizarla en el mismo momento que se produce el evento; el cliente se limita a "esperar" los mensajes sin tener que realizar consultas periódicamente.

En la arquitectura de Google, todo esto se consigue introduciendo un nuevo actor en el proceso: un **servidor de mensajería push o cloud to device** (que se traduciría en algo así como "mensajes de la nube al dispositivo"), y que se situaría entre una aplicación servidor y una aplicación cliente. Este servidor intermedio se encargará de recibir las notificaciones enviadas desde una aplicación servidor y de hacerlas llegar a la aplicación móvil. La comunicación se realiza mediante un

"protocolo" bien definido de registros y autorizaciones entre los distintos actores que participan en el proceso.

Un uso frecuente de la mensajería en la nube es el de una empresa que posee un sitio web y una aplicación móvil. Desde su sitio web envía noticias, novedades de producto, etc., que serán recibidas en los dispositivos móviles. Este es el supuesto que vamos a desarrollar en los siguientes puntos.

3.1.1. Firebase Cloud Messaging

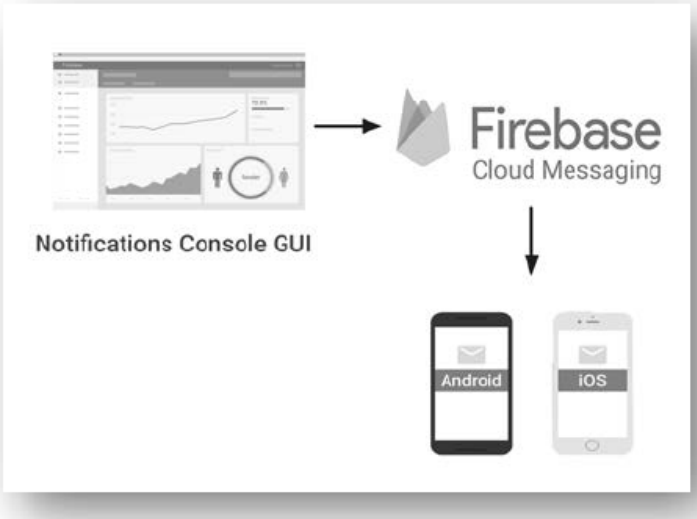
El servicio de notificaciones push de Google recibió en sus comienzos las siglas **C2DM** (Cloud to Device Messaging); posteriormente modificó su nombre a **GCM** (Google Cloud Messaging); y recientemente lo han cambiado a Firebase Cloud Messaging.

Firebase Cloud Messaging (FCM) es un servicio que ayuda a los desarrolladores a enviar datos, desde sus servidores o desde la consola de Firebase, a sus aplicaciones en dispositivos Android e iOS, e incluso para recibir mensajes desde los dispositivos en la misma conexión. El servicio FCM se encarga de todos los aspectos de la cola de mensajes y de la entrega a la aplicación de destino Android que se ejecuta en el dispositivo de destino. Es gratis y sin cuotas, no importa lo grandes que sean nuestras necesidades de mensajería.

Distinguiremos tres actores en los mensajes en la nube en FCM:

- **Servicio FCM**: Será el encargado de registrar el dispositivo para que pueda recibir notificaciones de nuestro proyecto: proporcionará un identificador de registro y activará el servicio. Desregistrar el dispositivo: no le enviará más notificaciones. Hará de intermediario en el envío de mensajes: enviará y gestionará la cola de mensajes. Este actor es gestionado por Google. Google proporciona servidores de conexión para HTTP y XMPP (Protocolo Extensible de Mensajería y Comunicación de presencia).
- **Aplicación Android:** Registrará y desregistrará el dispositivo en FCM y recibirá las notificaciones.
- **Aplicación servidor:** Enviará los mensajes. Necesita mantener un almacenamiento con los identificadores de los dispositivos registrados para el proyecto. Distinguiremos dos tipos de aplicaciones de servidor:
  - **Firebase Notifications:** Los mensajes e identificadores se gestionan desde la consola de Firebase del proyecto.
  - **Servidor propio:** Tendremos que crear una aplicación en un servidor, en el lenguaje de programación que deseemos, que tendrá que mantener los identificadores de los usuarios y gestionar los mensajes.

En la siguiente imagen se muestra cómo funciona Firebase Notifications:



Firebase Notifications es un servicio gratuito en la consola de Firebase que permite enviar notificaciones a los usuarios que utilicen la aplicación. Firebase Notifications está basado en Firebase Cloud Messaging y en el SDK Firebase Cloud Messaging. Ofrece una opción para desarrolladores y organizaciones que buscan una plataforma de mensajería flexible que requiere una codificación mínima para comenzar y una consola gráfica para enviar mensajes.

Podemos enviar dos tipos de mensajes: mensajes de notificación (visibles para el usuario en la barra de notificaciones del dispositivo) o mensajes de datos (no visibles para el usuario, con información extra a procesar por la aplicación). También es posible combinar en un mensaje los dos tipos.

Para ayudarnos a decidir entre desarrollar un servidor propio o utilizar Firebase Notifications para el envío de mensajes, tenemos que tener en cuenta varias cosas, pero una de las más importantes es la integración de FCM con Firebase Analytics. Permite enviar las notificaciones a usuarios que cumplen una serie de requisitos. Al hacer esto podemos enviar la notificación al usuario exacto que queramos que la reciba. También indicar que Firebase Notifications no permite enviar mensajes con datos exclusivamente,es decir, sin notificación (posteriormente ampliaremos este concepto).

Entre las principales funciones de Firebase Notifications se encuentran:

- **Análisis de notificaciones**: Analiza la recepción de los mensajes que enviamos y localiza embudos.
- **Perfilamiento versátil de mensajes**: Filtra a qué usuarios enviar los mensajes mediante el uso de segmentos de Google Analytics, suscripción a temas o dispositivos individuales.
- **Programación flexible de mensajes**: Envía notificaciones de inmediato o a una hora programada.

Usa Firebase Notifications, o bien desarrolla una aplicación de servidor, para redactar y enviar notificaciones. Firebase Cloud Messaging (FCM) se ocupa del enrutamiento y la entrega a los dispositivos de destino.

Respecto a la aplicación cliente, FCM es compatible con aplicaciones Android, iOS y Web. La aplicación cliente será la encargada de recibir los mensajes del servidor y procesarlos.

Actualmente FCM ofrece soporte para dos tipos de conexión: HTTP y XMPP. Puedes utilizarlas por separado o combinadas. Para desarrollar una aplicación FCM debemos elegir un tipo de conexión u otra. Va a depender, entre otras cosas de:

- **HTTP:** protocolo de transferencia de hipertexto. Su uso principal es la comunicación del servidor al dispositivo. No se abre un canal de comunicación. Es más sencillo de programar.
- **XMPP:** protocolo extensible de mensajería y comunicación de presencia. El uso principal es la comunicación del servidor al dispositivo y de dispositivo a dispositivo. Se abre un canal de comunicación. Es más rápido que HTTP. Se necesita un servidor XMPP.

Seguidamente vamos a implementar una aplicación básica FCM que utilizará una conexión HTTP. Enviaremos mensajes a los dispositivos que tengan instalada nuestra aplicación Android. Nos centraremos en la parte del dispositivo. La parte del servidor estará completamente implementada en Firebase Notifications. Además, veremos el código fuente básico en PHP y mySQL con los mínimos datos necesarios para que puedas implementar por tu cuenta un servidor personalizado para enviar notificaciones. Se han elegido PHP y mySQL, ya que son un binomio habitual para crear páginas web o integrarlas en una existente.

3.1.2. Firebase Messaging en Android

En este capítulo y los siguientes, vamos a crear una aplicación llamada “Eventos”, que será capaz de recibir, información de eventos que se produzcan en un cierto ámbito: una ciudad, un grupo de amigos... Para ello partiremos de una aplicación conectada a una base de datos Firestore, en la quepodremos consultar los eventos de la aplicación. Luego la iremos desarrollando y le añadiremos nuevas funcionalidades.

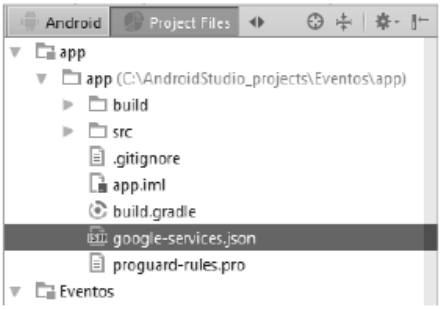
Primero, veremos cómo crear rápidamente una aplicación Android capaz de recibir notificaciones. Para que funcionen los mensajes de notificación, solamente es necesario que creemos un proyecto desde la consola de Firebase, agreguemos la referencia a dicho proyecto en Android y añadamos la librería `firebase-messaging` junto con los servicios de Google Play Services.

Ejercicio: Crear app Android que pueda recibir notificaciones

1. Entra en la consola de Firebase con una cuenta de Google.
2. Crea un nuevo proyecto llamado “Eventos”. Elige tu país. Pulsa en el botón **Crear proyecto** para finalizar.
3. Pulsa en el botón **Agrega Firebase a tu app para Android**.
4. Introduce `org.example.eventos` en el campo Nombre del paquete y pulsa en el botón **Registrar app**.
5. Pulsa en el botón **Descargar google-services.json**. Se descargará el fichero de configuración.
6. Crea un nuevo proyecto en Android Studio con los siguientes datos:

```
Aplication name: Eventos
Company Domain: example.org
Target Android Devices: Phone and Tablet
Minimum Required SDK: API 14 (4.0)
Add an activity to Mobile: Basic Activity
Activity name: ActividadPrincipal
Layout name: actividad_principal
```

7. En Android Studio cambia la vista del proyecto a PROJECT FILES y copia dentro de la carpeta app el fichero google-services.json que has descargado desde la consola de Firebase anteriormente.



8. Introduce la siguiente dependencia en el archivo `build.gradle (Project: Eventos)`:

```
classpath 'com.google.gms:google-services:3.1.0'
```

9. Introduce al final del fichero `build.gradle (Module: app)`:

```
apply plugin: 'com.google.gms.google-services'
```

10. Añade al fichero `build.gradle (Module: app)` la siguiente dependencia:

```
implementation 'com.google.firebase:firebase-messaging:11.8.0'
```

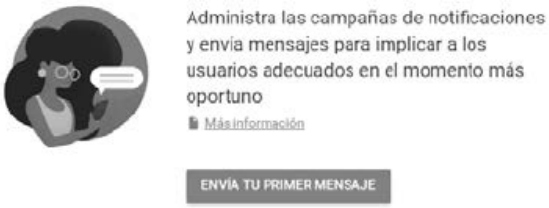
11. Instala y ejecuta la aplicación en un dispositivo.

Ahora la aplicación ya es capaz de recibir mensajes de notificación enviadas desde la consola de Firebase. Hemos creado un proyecto en Firebase al que hemos añadido una aplicación Android, que es la que recibirá las notificaciones. En Android Studio, hemos añadido el fichero de configuración obtenido desde Firebase y una serie de dependencias en los ficheros graddle del proyecto. Las dependencias que hemos añadido nos permiten usar Google Play Services y el servicio de notificaciones de Firebase.

Vamos a probar que es capaz de recibir notificaciones enviadas desde la consola de Firebase.

Ejercicio: Enviar mensajes de notificación desde la consola de Firebase

1. En la consola de Firebase abre el proyecto “Eventos”. Selecciona el menú **NOTIFICATIONS**.
2. Pulsa sobre el botón **Envía tu primer mensaje**.

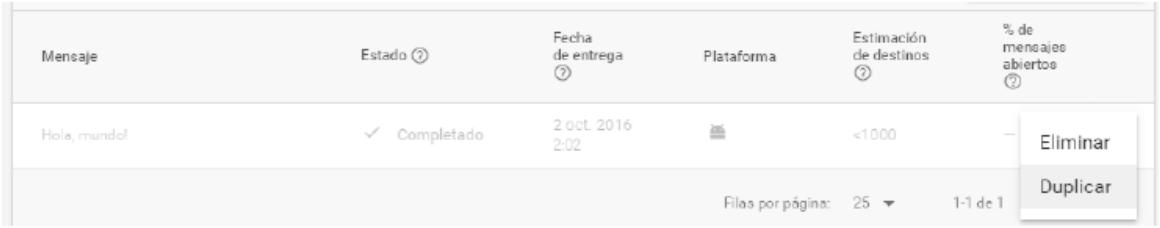


En la siguiente pantalla vamos a introducir los datos del mensaje y lo vamos a enviar a la aplicación.

3. Introduce en el campo Texto del mensaje: “¡Se acerca el carnaval!”.
4. En el desplegable **Aplicación** selecciona `org.example.eventos`
5. Asegúrate de haber abierto la aplicación Android del ejercicio anterior, que no se encuentre en primer plano y que no esté cerrada.
6. Pulsa sobre el botón **Enviar mensaje** en la consola de Firebase. Te mostrará otra ventana para que confirmes el envío. Pulsa en **Enviar**.
7. Comprueba que recibes la notificación en el dispositivo.

Normalmente la recepción es inmediata, pero ocasionalmente puede tardar unos minutos. Es recomendable utilizar un dispositivo real o un emulador actualizado para probar las notificaciones. Hemos enviado una notificación desde la consola de Firebase a la aplicación Android. Simplemente conectando la aplicación a Firebase, de igual forma que lo hacemos para otro servicio, y añadiendo las dependencias de FCM (`com.google.firebase:firebase-messaging`), hemos conseguido que la aplicación pueda recibir notificaciones.

8. Cuando recibas la notificación, pulsa sobre ella. Se abrirá la aplicación.
9. Duplica el mensaje enviado. Selecciona **Duplicar** en el menú del mensaje "¡Se acerca carnaval!" que has enviado anteriormente.



10. Con la aplicación abierta, pulsa sobre el botón **Enviar mensaje** en la consola de Firebase y confirma su envío.
11. Comprueba lo que ocurre en la aplicación. Aunque la aplicación ha recibido la notificación, no ha habido ninguna acción que indique que ha sido así. Para ello, tendremos que administrar los mensajes, como veremos posteriormente.

Esta es la forma más rápida de usar Firebase Cloud Messaging en una aplicación Android. ¿Cómo funciona? Hemos creado un proyecto en Firebase, donde hemos activado el servicio de notificaciones asociando una aplicación Android. Firebase Notifications conforma la aplicación servidor. La aplicación Android es la aplicación cliente. Es el cliente el que recibirá las notificaciones. Una vez realizados los preparativos, cuando el usuario abra por primera vez la aplicación, quedará registrada en Firebase y será capaz de recibir notificaciones. Enviamos las notificaciones desde una aplicación de servidor, en este caso la consola de Firebase. Posteriormente, el servicio Firebase Cloud Messaging es el que gestionará la cola de mensajes y su entrega a la aplicación cliente.

Seguidamente crearemos la aplicación básica "Eventos":

Ejercicio: Aplicación base "Eventos"

1. En el proyecto "Eventos" añade las siguientes dependencias:

```
implementation 'com.google.firebase:firebase-firestore:11.8.0'
implementation 'com.firebaseui:firebase-ui-firestore:3.1.3'
implementation 'com.android.support:cardview-v7:27.0.2'
implementation 'com.android.support:recyclerview-v7:27.0.2'
```

2. Si tienes problemas con las librerías, cambia los valores subrayados en el fichero gr addl e.

```
android {
    compileSdkVersion 27
    defaultConfig {
        targetSdkVersion 27
        …
dependencies {
```

```
    implementation 'com.android.support:appcompat-v7:27.0.2'
    implementation 'com.android.support:design:27.0.2'
    …
```

3. Sustituye el contenido del layout `content_actividad_principal` por:

```
<android.support.v7.widget.RecyclerView
              xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:id="@+id/reciclerViewEventos"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   app:layout_behavior="@string/appbar_scrolling_view_behavior" />
```

4. Crea un layout llamado `evento.xml` e introduce el código siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   xmlns:tools="http://schemas.android.com/tools"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_margin="8dp"
   android:layout_height="wrap_content">
   <android.support.constraint.ConstraintLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content">
      <ImageView android:layout_height="100dp"
         android:layout_alignParentLeft="true"
         android:layout_centerVertical="true"
         android:id="@+id/imgImagen"
         android:layout_width="150dp" />
      <TextView
         android:id="@+id/txtEvento"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_alignParentEnd="true"
         android:layout_alignParentRight="true"
         android:layout_toRightOf="@+id/imgImagen"
         android:layout_weight="0.70"
         android:textAppearance="@style/TextAppearance.AppCompat.Subhead"
         app:layout_constraintEnd_toEndOf="parent"
         app:layout_constraintStart_toEndOf="@+id/imgImagen"
         app:layout_constraintTop_toTopOf="parent" />
      <TextView
         android:id="@+id/txtCiudad"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
```

```
         android:layout_below="@+id/txtEvento"
         android:layout_toRightOf="@+id/imgImagen"
         app:layout_constraintEnd_toEndOf="parent"
         app:layout_constraintStart_toEndOf="@+id/imgImagen"
         app:layout_constraintTop_toBottomOf="@+id/txtEvento" />
      <TextView
         android:id="@+id/txtFecha"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_below="@+id/txtCiudad"
         android:layout_toRightOf="@+id/imgImagen"
         android:textAppearance="@style/TextAppearance.AppCompat.Caption"
         android:textStyle="italic"
         app:layout_constraintEnd_toEndOf="parent"
         app:layout_constraintStart_toEndOf="@+id/imgImagen"
         app:layout_constraintTop_toBottomOf="@+id/txtCiudad" />
   </android.support.constraint.ConstraintLayout>
</android.support.v7.widget.CardView>
```

5. Crea una clase `Evento` con el siguiente contenido para definir el modelo:

```
public class Evento {
   private String evento;
   private String ciudad;
   private String fecha;
   private String imagen;

   public Evento() {}
}
```

6. Pulsa con el botón derecho al final del código e indica Generate… / Constructor. Selecciona todos los campos y pulsa **OK**.

7. Pulsa con el botón derecho al final del código e indica Generate… / Getter and Setter. Selecciona todos los campos y pulsa **OK**.

8. La siguiente clase nos va a permitir crear la colección `eventos` en una base de datos Firestore. A diferencia de Realtime Database, no disponemos de una opción directa para importar datos desde JSON.

```
public class EventosFirestore {
   public static String EVENTOS = "eventos";
   static String SERVIDOR = "https://pixabay.com/static/uploads/photo/";

   public static void crearEventos() {
      Evento evento;
      FirebaseFirestore db = FirebaseFirestore.getInstance();
      evento = new Evento("Carnaval", "Rio de Janeiro", "21/02/2017",
              SERVIDOR+"2014/08/06/11/51/carnival-411494_960_720.jpg");
      db.collection(EVENTOS).document("carnaval").set(evento);
      evento = new Evento("Fallas", "Valencia", "19/03/2017",
              SERVIDOR+"2015/03/22/18/12/failures-685055_960_720.jpg");
```

```
        db.collection(EVENTOS).document("fallas").set(evento);
        evento = new Evento("Nochevieja", "Nueva York", "31/12/2016",
                SERVIDOR+"2013/02/10/15/35/fireworks-80187_960_720.jpg");
        db.collection(EVENTOS).document("nochevieja").set(evento);
        evento = new Evento("Noche de San Juan", "Alicante", "23/06/2017",
                SERVIDOR+"2013/06/30/18/29/midsummer-142484_960_720.jpg" );
        db.collection(EVENTOS).document("sanjuan").set(evento);
        evento = new Evento("Semana Santa", "Sevilla", "14/04/2017",
                SERVIDOR+"2016/03/17/10/08/easter-1262664_960_720.jpg");
        db.collection(EVENTOS).document("semanasanta").set(evento);
    }
}
```

9. En la consola de Firebase abre el proyecto “Eventos”. Selecciona el apartado **Database**. Pulsa en el botón **Prueba la versión beta de Firestore**.

10. Te preguntará que tipo de reglas de seguridad quieres utilizar. Selecciona:

NOTA: Recuerda cambiar estas reglas una vez tu aplicación contenga datos reales.

11. Añade una clase nueva llamada `AdaptadorEventos`:

```
public class AdaptadorEventos extends FirestoreRecyclerAdapter
                                    <Evento, AdaptadorEventos.ViewHolder> {
    protected View.OnClickListener onClickListener;

    public AdaptadorEventos(@NonNull FirestoreRecyclerOptions<Evento>
                                                                options) {
        super(options);
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.evento, parent, false);
        view.setOnClickListener(onClickListener);
        return new ViewHolder(view);
    }
```

```
   class ViewHolder extends RecyclerView.ViewHolder {
      public TextView txtEvento, txtCiudad, txtFecha;
      public ImageView imagen;
      public ViewHolder(View itemView) {
         super(itemView);
         txtEvento = (TextView) itemView.findViewById(R.id.txtEvento);
         txtCiudad = (TextView) itemView.findViewById(R.id.txtCiudad);
         txtFecha = (TextView) itemView.findViewById(R.id.txtFecha);
         imagen = (ImageView) itemView.findViewById(R.id.imgImagen);
      }
   }

   @Override protected void onBindViewHolder(@NonNull ViewHolder holder,
                                     int position, @NonNull Evento item) {
      holder.txtEvento.setText(item.getEvento());
      holder.txtCiudad.setText(item.getCiudad());
      holder.txtFecha.setText(item.getFecha());
      new DownloadImageTask((ImageView) holder.imagen)
              .execute(item.getImagen());
      holder.itemView.setOnClickListener(onClickListener);
   }

   public void setOnItemClickListener(View.OnClickListener onClick) {
      onClickListener = onClick;
   }

   private class DownloadImageTask extends AsyncTask<String,Void,Bitmap> {
      ImageView imageView;

      public DownloadImageTask(ImageView bmImage) {
         this.imageView = bmImage;
      }

      protected Bitmap doInBackground(String... urls) {
         String urldisplay = urls[0];
         Bitmap mImagen = null;
         try {
            InputStream in = new java.net.URL(urldisplay).openStream();
            mImagen = BitmapFactory.decodeStream(in);
         } catch (Exception e) {
            e.printStackTrace();
         }
         return mImagen;
      }

      protected void onPostExecute(Bitmap result) {
         imageView.setImageBitmap(result);
      }
   }
}
```

12. Declara el adaptador en la clase `ActividadPrincipal`:

```
private AdaptadorEventos adaptador;
```

13. En la clase `ActividadPrincipal` añade al final del método `onCreate()`:

```
crearEventos();
Query query = FirebaseFirestore.getInstance()
        .collection(EVENTOS)
        .limit(50);
FirestoreRecyclerOptions<Evento> opciones = new FirestoreRecyclerOptions
        .Builder<Evento>().setQuery(query, Evento.class).build();
adaptador = new AdaptadorEventos(opciones);
final RecyclerView recyclerView = findViewById(R.id.reciclerViewEventos);
recyclerView.setLayoutManager(new LinearLayoutManager(this));
recyclerView.setAdapter(adaptador);
```

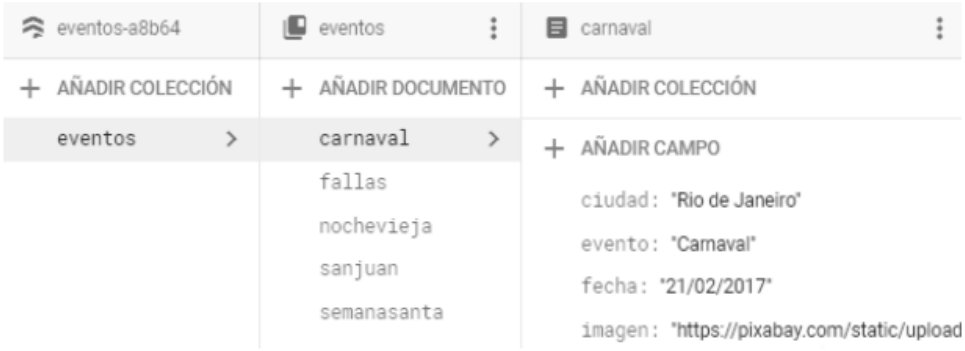
14. En esta clase añade los siguientes métodos:

```
@Override public void onStart() {
   super.onStart();
   adaptador.startListening();
}
@Override public void onStop() {
   super.onStop();
   adaptador.stopListening();
}
```

Gracias a los escuchadores que activamos cuando la actividad está visible, cualquier cambio que se produzca en tiempo real en la base de datos será reflejada directamente en el `RecyclerView`.

15. Ejecuta la aplicación y comprueba que funciona correctamente.

16. Vuelve a la consola de Firebase y comprueba que se ha creado la base de datos:



17. Comenta la línea `crearEventos();` para evitar que se creen más eventos.

Hemos creado una aplicación que muestra la información de una serie de eventos. Para cada evento se muestra una imagen, el nombre del evento, la ciudad y la fecha donde se celebra. La información se encuentra almacenada en Firestore.

Podemos consultar los eventos cuando abrimos la aplicación. Si el usuario no la abre, puede que se pierda un evento de su interés. Con el fin de evitar esto, hacemos uso de mensajes de notificación para que los usuarios estén informados de eventos cercanos a realizarse sin entrar en la aplicación.

3.1.3. Aplicación cliente Firebase Cloud Messaging

La aplicación que hemos realizado es capaz de recibir notificaciones. La notificación se muestra en el área de notificaciones del dispositivo. Si pulsamos sobre ella, se abre la aplicación. Esto ocurre si la aplicación está en segundo plano, pero en el momento de escribir este manual, si se encuentra cerrada, la notificación no se recibe al instante e incluso puede llegar a perderse. En la anterior versión de notificaciones push, Google Cloud Messaging, se recibían notificaciones incluso si la aplicación estaba cerrada. Podremos solucionarlo añadiendo un `BroadcastReceiver` al iniciar el dispositivo, lo veremos más adelante. Otro problema que tiene nuestra aplicación es que, con la aplicación abierta y visible, aunque se recibe la notificación, no se muestra nada al usuario. Vamos a ver en detalle cómo crear una aplicación Android capaz de gestionar todo tipo de mensajes de notificación.

Para crear una aplicación cliente en Android que utilice el servicio FCM, necesitamos hacer uso de Google Play Services. En versiones anteriores, se usaban unas librerías propias para funcionar.

Distinguimos dos tipos de mensaje: **mensaje de notificación** (sin carga de datos) y **mensaje de datos** [notification payload] (con carga de datos). Firebase Cloud Messaging distingue entre estos dos tipos de mensajes. La notificación sin carga de datos es la opción más liviana, con un límite de 2 Kb y un conjunto predefinido de claves visibles para el usuario. Las notificaciones pueden contener una carga de datos opcional. La carga de datos permite enviar hasta 4 Kb de pares de clave/valor personalizados.

Si queremos que la aplicación, siempre informe al usuario cuando reciba notificaciones, incluso cuando se encuentre en primer plano, debemos programar la administración de mensajes.

3.1.3.1. Administración de mensajes FCM en Android

Administrar mensajes significa que podemos procesar la información que se nos envía de la forma que deseemos. Por ejemplo, si nos envían una notificación con el nombre de un color, podemos mostrar al usuario el nombre del color o bien cambiar el color de fondo de la aplicación a ese color sin advertir al usuario. Para administrar las notificaciones FCM en Android, debemos usar un servicio que extienda `FirebaseMessagingService`. Tenemos que sobrescribir el método `onMessageReceived` y realizar el tratamiento de las notificaciones como deseemos. Podemos tratar la mayoría de las notificaciones mediante el método `onMessageReceived`, salvo para las siguientes excepciones:

- **Mensajes de notificación sin carga de datos recibidas con la aplicación en segundo plano:** La notificación se muestra en el área de notificaciones. Cuando un usuario hace clic sobre ella, se abre la aplicación de forma predeterminada.
- **Mensajes de notificación con carga de datos con la aplicación en segundo plano:** La notificación se muestra en el área de notificaciones y la carga de datos se entrega en los extras de la intención de la actividad inicial.

Si al abrir una notificación de la bandeja, deseamos que la aplicación realice una acción específica tenemos que implementarlo. El siguiente cuadro resume como se comporta la aplicación dependiendo del estado en el que se encuentre y el tipo de notificación que se reciba.

Estado	Notificación	Datos	Ambos
Primer plano	`onMessageReceived`	`onMessageReceived`	`onMessageReceived`
Segundo plano	Área de notificaciones	`onMessageReceived`	Notificación: Área de notificaciones Datos: extras de la intención

Ejercicio: Administrar notificaciones push en una aplicación Android

1. Para usar el servicio `FirebaseMessagingService`, necesitarás agregarlo en el manifiesto:

```
<service
   android:name=".EventosFCMService">
   <intent-filter>
```

```
      <action         android:name="com.google.firebase.MESSAGING_EVENT"/>
   </intent-filter>
</service>
```

Con el `intent-filter com.google.firebase.MESSAGING_EVENT`, indicamos que el servicio va a invocarse cuando se reciban mensajes desde Firebase.

2. Crea una nueva clase llamada `EventosFCMService` con el siguiente contenido:

```
public class EventosFCMService extends FirebaseMessagingService {
   @Override
   public void onMessageReceived(RemoteMessage remoteMessage) {
      if (remoteMessage.getNotification() != null) {
         mostrarDialogo(getApplicationContext(),
               remoteMessage.getNotification().getBody());
      }
   }
}
```

Al sobrescribir el método `onMessageReceived` en la clase `FirebaseMessagingService`, puedes obtener los datos de la notificación y realizar las acciones que estimes oportunas. Obtendremos un objeto `RemoteMessage` que contendrá la información de la notificación. La notificación que hemos recibido no contiene sobrecarga de datos. Por ello, capturamos la información que se nos pasa en el cuerpo de la notificación, es decir, el mensaje que se muestra en el área de notificaciones si la aplicación no está en primer plano, con `getNotification().getBody()`. También podríamos capturar el título de la notificación con `getNotification().getTitle()`.

3. Inserta al final del método onStart en la clase `ActividadPrincipal`:

```
current=this;
```

4. Añade al final de la clase `ActividadPrincipal`:

```
private static ActividadPrincipal current;
public static ActividadPrincipal getCurrentContext() {
   return current;
}
```

Cuando recibimos la notificación, mostramos un diálogo. Para mostrar un diálogo desde un servicio que no tiene un contexto, hemos creado estos eventos a fin de recuperar el contexto de `ActividadPrincipal`.

5. Crea una clase llamada `Comun` con el siguiente contenido:

```
public class Comun {
   static void mostrarDialogo (final Context context
                                                , final String mensaje){
      Intent intent = new Intent(context, Dialogo.class);
      intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
      intent.putExtra("mensaje", mensaje);
```

```
      context.startActivity(intent);
   }
}
```

6. Crea una nueva clase llamada `Dialogo` con el siguiente contenido:

```
public class Dialogo extends AppCompatActivity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      Bundle extras = getIntent().getExtras();
      if (getIntent().hasExtra("mensaje")) {
         AlertDialog alertDialog = new AlertDialog.Builder(this).create();
         alertDialog.setTitle("Mensaje:");
         alertDialog.setMessage(extras.getString("mensaje"));
         alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL, "CERRAR",
                               new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
               dialog.dismiss();
               finish();
            }
         });
         alertDialog.show();
         extras.remove("mensaje");
      }
   }
}
```

7. Declara la actividad `Dialogo` en el manifiesto:

```
<activity
    android:name=".Dialogo"
    android:label="@string/app_name">
</activity>
```

8. Ejecuta la aplicación.

9. Accede a la consola de Firebase con la cuenta de Google que utilizaste para dar de alta el proyecto “Eventos” y selecciónala.

10. Selecciona el menú **Notifications** en la sección GROW, en el menú lateral izquierdo.

11. Duplica un mensaje.

12. Con la aplicación Android abierta y visible, pulsa sobre el botón **Enviar mensaje** en la consola de Firebase.

    Comprueba que hemos sido capaces de procesar la notificación y mostrarla al usuario. En el método `onMessageReceived`, consultamos si `remoteMessage.getNotification()` es distinto de nulo, lo cual indica que hemos recibido un texto en la notificación. Recuperamos el texto de la notificación mediante `remoteMessage.getNotification().getBody()`. Lo que

hacemos es mostrarlo mediante un diálogo. Podríamos haber mostrado como un `Toast`, pero tendríamos que estar atentos para ver lo que nos han enviado.

13. Vuelve a la consola de Firebase y duplica el último mensaje.

14. Con la aplicación Android en segundo plano pulsa sobre el botón **Enviar mensaje** en la consola de Firebase.

    Si la aplicación está en segundo plano, Android dirige los mensajes de notificación al área de notificaciones. Cuando el usuario pulsa sobre la notificación, se abre la aplicación de forma predeterminada. La carga de datos de una notificación se entrega en los extras de la intención de la actividad inicial. Vas a capturar los extras, para mostrar un mensaje cuando se abra la aplicación.

15. Añade al final de la clase `ActividadPrincipal` el siguiente código:

```
@Override
protected void onResume() {
   super.onResume();
   Bundle extras = getIntent().getExtras();
   if (getIntent().hasExtra("body")) {
      mostrarDialogo(this, extras.getString("body"));
      extras.remove("body");
   }
}
```

16. Ejecuta la aplicación y ponla en segundo plano.

17. Envía un mensaje desde la consola de Firebase.

Comprueba que recibes la notificación en la bandeja del sistema. Pulsa sobre ella y comprueba que se abre la aplicación, pero que no ocurre nada. Esto pasa porque la notificación no tiene carga de datos. Cuando una notificación se envía con carga de datos es cuando los datos se reciben como extras en la aplicación.

Seguidamente vamos a ver las notificaciones con carga de datos:

Ejercicio: Administrar notificaciones push con carga de datos

1. Sustituye el método `onResume` de la clase `ActividadPrincipal` por el siguiente:

```
@Override
protected void onResume() {
   super.onResume();
   Bundle extras = getIntent().getExtras();
   if (extras!=null && extras.keySet().size()>4) {
      String evento="";
      evento ="Evento: "+extras.getString("evento")+ "\n";
      evento = evento + "Día: "+ extras.getString("dia")+ "\n";
      evento = evento +"Ciudad: "+extras.getString("ciudad")+ "\n";
      evento = evento +"Comentario: "+extras.getString("comentario");
```

```
      mostrarDialogo(getApplicationContext(), evento);
      for (String key : extras.keySet()) {
         getIntent().removeExtra(key);
      }
      extras = null;
   }
}
```

Utilizamos la instrucción `extras!=null` para comprobar si recibimos algún extra y `extras.keySet().size()>4`. `keySet` nos permite preguntar por el array que contiene los valores de los extras y que sea mayor a 4. Siempre se reciben cuatro 4 extras por defecto con información básica de la notificación. Si solo recibimos 4 extras es que no estamos recibiendo carga de datos.

2. Envía un mensaje desde la consola de Firebase: duplica el último y añade los siguientes pares clave/valor en la sección OPCIONES AVANZADAS y la subsección DATOS PERSONALIZADOS:

CLAVE	VALOR
evento	Carnaval 2018
dia	21/02/2018
ciudad	Rio de Janeiro
comentario	Ven disfrazado a la fiesta del año

3. Ejecuta la aplicación y, con la aplicación en segundo plano, envía la notificación.

   Comprueba que la recibes y que, al pulsar sobre la notificación en la bandeja del sistema, se abre la aplicación y se muestra un diálogo con la información.

4. Duplica la última notificación que has enviado y vuélvela a enviar, pero esta vez, asegúrate de que la aplicación está visible.

   Comprueba que no aparece el mismo mensaje que cuando pulsas sobre la notificación de la bandeja del sistema y se abre la aplicación. En el método `onMessageReceived` no tenemos en cuenta la carga de datos de una notificación.

5. Sustituye el método `onMessageReceived` de la clase `EventosFCMService` por el siguiente:

```
@Override
public void onMessageReceived(RemoteMessage remoteMessage) {
   if (remoteMessage.getData().size() > 0) {
      String evento="";
      evento ="Evento: "+remoteMessage.getData().get("evento")+ "\n";
      evento = evento + "Día: "+ remoteMessage.getData().get("dia")+ "\n";
      evento = evento +"Ciudad: "+ remoteMessage.getData().get("ciudad")+"\n";
      evento = evento +"Comentario: "
                              +remoteMessage.getData().get("comentario");
      mostrarDialogo(getApplicationContext(), evento);
   } else {
      if (remoteMessage.getNotification() != null) {
```

```
            mostrarDialogo(getApplicationContext(),
                                remoteMessage.getNotification().getBody());
        }
    }
}
```

Mediante `remoteMessage.getData().size()` consultamos si el mensaje contiene carga de datos. Si el tamaño es mayor que 0, entonces podremos recuperar la carga de datos. Tenemos que saber exactamente cómo se llama cada una de las cargas. Por ejemplo, si en el servidor hemos definido una carga llamada "ciudad", la recuperaremos en la aplicación con el mismo nombre, llamando a `remoteMessage.getData().get("ciudad")`.

6. Ejecuta la aplicación.
7. Duplica y envía el último mensaje asegurándote que la aplicación se encuentra abierta en primer plano.

Comprueba que ahora sí que se muestran los datos del mensaje cuando la aplicación se encuentra en primer plano. Tanto los datos propios de la notificación como la sobrecarga de datos se manejan en `onMessageReceived`. Al recibir el mensaje, hubiéramos podido programar lo que quisiéramos. Pero hemos de tener en cuenta que, por defecto, el servicio `FirebaseMessagingService` no se dispara cuando la aplicación se encuentra en segundo plano.

3.1.3.2. Administración de identificadores FCM en Android

Cada dispositivo en el que se instale una aplicación que utilice FCM obtendrá un identificador único formado por números, letras y caracteres especiales. El identificador es único para cada instancia de la aplicación. Es un mecanismo para autentificar y autorizar acciones. FCM utilizará dicho identificador para enviarle mensajes. Se almacenará en la consola Firebase y también lo podemos almacenar en un servidor propio desde el cual podremos enviar mensajes. No podemos consultar el identificador en la consola de Firebase de forma predeterminada. Podríamos utilizar Firebase Realtime Database o Firestore para almacenarlos, si es necesario. Si utilizamos un servidor propio, será nuestra responsabilidad almacenar y mantener los identificadores. Tener un almacenamiento no actualizado puede provocar que no se envíe a algunos dispositivos o que se envíe a dispositivos que ya no usan ese identificador.

Hasta el momento no hemos realizado ninguna acción con los identificadores y hemos sido capaces de enviar y recibir mensajes. Pero, si utilizamos un servidor propio para enviar mensajes, sí que vamos a necesitar tener almacenados los identificadores.

Para obtener el identificador de los dispositivos, utilizaremos un servicio que extienda `FirebaseInstanceIdService`. Lo usaremos para capturar y almacenar los identificadores en su creación y actualización. Es un requisito para enviar mensajes a dispositivos específicos o para crear grupos de dispositivos.

Ejercicio: Administrar los identificadores FCM en Android

1. Añade el siguiente servicio en el archivo de manifiesto:

```
<service
    android:name=".EventosFCMInstanceIDService">
    <intent-filter>
        <action     android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
    </intent-filter>
</service>
```

2. Crea una clase `EventosFCMInstanceIDService` con el siguiente contenido:

```
public class EventosFCMInstanceIDService
                                        extends FirebaseInstanceIdService {
   @Override
   public void onTokenRefresh() {
      String idPush;
      idPush = FirebaseInstanceId.getInstance().getToken();
      guardarIdRegistro(getApplicationContext(), idPush);
   }
}
```

En el arranque inicial de la aplicación, FCM genera un identificador (token) de registro para la instancia de la aplicación. Cuando se genera el identificador, se llama al método `onTokeRefresh` de la clase `FirebaseInstanceIdService`. También es llamado cuando el identificador del dispositivo caduca. Podemos obtener el identificador mediante una llamada a `FirebaseInstanceId.getInstance().getToken()`.

El identificador es estable, pero se puede convertir en inválido si:

- La aplicación elimina el número de instancia.
- El dispositivo realiza un borrado de fábrica.
- El usuario desinstala la aplicación.
- El usuario limpia los datos de la aplicación.

Si el identificador se convierte en inválido, el servicio `FirebaseInstanceIdService` realiza una llamada a `onTokenRefresh()`, en la que podremos volver a preguntar por el nuevo identificador y guardarlo en nuestro servidor, si es que lo utilizamos.

Para acceder al identificador extiende `FirebaseInstanceIdService`. Tienes que haber agregado el servicio al manifiesto, luego llama a `getToken` dentro del método `onTokenRefresh`. Guardamos el identificador devuelto por FCM mediante el método `guardarIdRegistro`. Vamos a guardar y mantener los identificadores en un servidor propio. Nuestra aplicación va a poder utilizar tanto Firebase Notifications como un servidor propio.

3. Declara las siguientes variables en la clase `Comun`:

```
static final String URL_SERVIDOR =
                        "http://cursoandroid.hol.es/notificaciones/";
static String ID_PROYECTO="eventos-xxxxx";
```

Guardamos el URL del servidor propio en la variable `URL_SERVIDOR`. Para este ejercicio encontrarás un servidor activo con el que podrás hacer las pruebas que necesites. En la variable `ID_PROYECTO` vamos a introducir el identificador del proyecto en la consola Firebase. Tendrás que sustituir `"eventos-xxxxx"` por tu identificador del proyecto. Lo encontrarás en la consola de Firebase, seleccionando el proyecto y accediendo a la opción **Configuración de proyecto**.

4. Declara la siguiente variable en la clase `Comun`:

```
String idRegistro ="";
```

5. Inserta al final de la clase `Comun` los siguientes métodos:

```
public static class registrarDispositivoEnServidorWebTask
                              extends AsyncTask<Void, Void, String> {
   String response="error";
   Context contexto;
   String idRegistroTarea ="";

   public void onPreExecute() {
      super.onPreExecute();
   }

   @Override
   protected String doInBackground(Void... arg0) {
      try{
         Uri.Builder constructorParametros = new Uri.Builder()
                  .appendQueryParameter("iddevice", idRegistroTarea)
                  .appendQueryParameter("idapp", ID_PROYECTO);
         String parametros =
                  constructorParametros.build().getEncodedQuery();
         String url = URL_SERVIDOR + "registrar.php";
         URL direccion = new URL(url);
         HttpURLConnection conexion = (HttpURLConnection)
                  direccion.openConnection();
         conexion.setRequestMethod("POST");
         conexion.setRequestProperty("Accept-Language", "UTF-8");
         conexion.setDoOutput(true);
         OutputStreamWriter outputStreamWriter = new
                  OutputStreamWriter(conexion.getOutputStream());
         outputStreamWriter.write(parametros.toString());
         outputStreamWriter.flush();
         int respuesta = conexion.getResponseCode();
         if (respuesta==200){
            response="ok";
         } else {
            response="error";
         }
```

```
      } catch (IOException e) {
         response= "error";
      }
      return response;
   }

   public void onPostExecute(String res) {
   }
}
public static void guardarIdRegistro(Context context, String idRegistro){
   registrarDispositivoEnServidorWebTask tarea =
                              new registrarDispositivoEnServidorWebTask();
   tarea.contexto=context;
   tarea.idRegistroTarea=idRegistro;
   tarea.execute();
}
```

Almacenaremos el identificador de registro obtenido, en el servidor web propio, mediante la tarea asíncrona `registrarDispositivoEnServidorWebTask`. A través de una llamada HTTP POST insertaremos el identificador del dispositivo y el identificador del proyecto en nuestro servidor web. El objetivo es almacenar los identificadores de los dispositivos a los cuales les podemos enviar mensajes. En este ejercicio también almacenamos el identificador de proyecto, esto solo es necesario en esta aplicación de ejemplo. Esto completará el registro tanto con el servidor de FCM como con nuestro servidor web. Recuerda que si no realizas el registro en el servidor web propio del identificador FCM, el dispositivo no podrá recibir ningún mensaje desde este servidor.

El identificador de registro es válido para un dispositivo y una instalación. Si se instala de nuevo la aplicación, el dispositivo se debe volver a registrar. Si tenemos una copia de seguridad de los datos de la aplicación, el identificador de registro FCM no se debe restaurar.

6. Ejecuta la aplicación.

   Consulta el servidor propio que vamos a utilizar para enviar notificaciones y almacenar identificadores, comprueba que no se ha almacenado el dispositivo. Accede a la siguiente dirección con el navegador de tu ordenador: http://cursoandroid.hol.es/notificaciones/listar.html e introduce el identificador del proyecto que obtuviste en la consola de Firebase (recuerda que tiene la siguiente forma: `"eventos-xxxxx")`. Pulsa el botón listar y verifica que no aparece ningún identificador.

   No hemos conseguido incluir el identificador FCM en el servidor porque ya hemos ejecutado la aplicación anteriormente y se ha registrado en el servicio FCM. Por lo tanto, no se vuelve a llamar a `onTokenRefresh()` y no se va a registrar en nuestro servidor.

7. Desinstala la aplicación.
8. Vuelve a instalarla.
9. Ejecuta la aplicación.

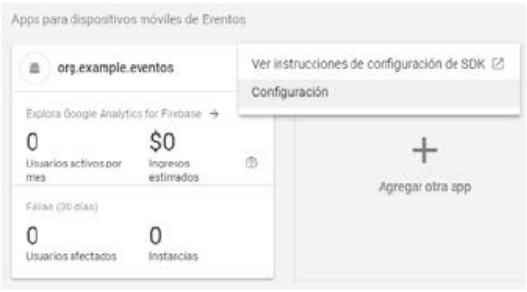
Vuelve a consultar el servidor propio y revisa que se lista un identificador. Dicho identificador es el obtenido por el dispositivo al registrarse en FCM y que hemos almacenado en nuestro servidor web.

Una vez que hemos conseguido almacenar el identificador en el servidor propio, vamos a ser capaces de enviar mensajes tanto desde la consola de Firebase, como ya hemos hecho, como desde el servidor propio. Vamos a comprobar que también puedes enviar mensajes desde el servidor propio.

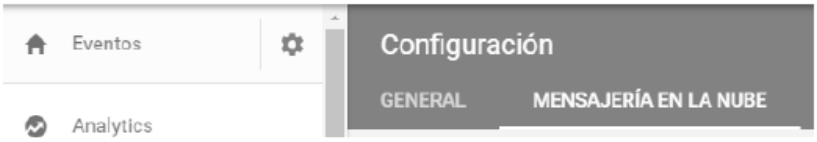
Puedes comprobar la aplicación en http://cursoandroid.hol.es. Elige **Notificaciones PUSH**:

1. Listar dispositivos registrados: Introduce el identificador del proyecto; te presentará los identificadores de los dispositivos registrados y que, por tanto, pueden recibir notificaciones.
2. Enviar una notificación a los dispositivos: Introduce:
   - Identificador de proyecto: necesario para filtrar los dispositivos registrados en tu aplicación y no enviar a los dispositivos de todos los alumnos. No es necesario en una aplicación real.
   - Clave de servidor: necesario para validar el servidor propio en FCM y poder enviar notificaciones.
   - Mensaje: Introduce en el campo Mensaje lo que desees enviar.

Para enviar un mensaje desde el servidor propio del curso, vamos a necesitar una clave llamada “Clave de servidor". Esta clave la puedes encontrar en la consola de Firebase. Elige el proyecto y **Configuración**, en las opciones del proyecto.



Selecciona la pestaña **Mensajería en la nube**.



Encontraremos, además de la “Clave de servidor”, encontramos la “Clave de servidor heredada”, que es su equivalente en las versiones anteriores de notificaciones push, a la que Google está dejando de prestar asistencia. La “Clave de servidor” la tendremos que introducir en los mensajes que enviemos para que FCM los valide y se puedan enviar a los dispositivos.

Comprueba que el dispositivo puede recibir mensajes. Accede al servidor del curso http://cursoandroid.hol.es, elige la opción **Notificaciones PUSH**, luego **Enviar una notificación a los dispositivos**", introduce el identificador del proyecto, la clave de servidor y un texto en el campo Mensaje. Pulsa el botón **Enviar notificación** y revisa que se recibe en el dispositivo.

3.1.3.3. Iniciar aplicaciones FCM

Para que una aplicación pueda recibir un mensaje debe encontrarse en ejecución en primer o segundo plano. Cuando un usuario inicia el dispositivo y no ha iniciado la aplicación todavía o cierra la aplicación completamente, no recibirá ningún mensaje. Para solucionarlo utilizaremos un `BroadcastReceiver` que iniciará el servicio `FirebaseMessagingService`. Vamos a ver cómo hacerlo.

Ejercicio: BroadcastReceiver en aplicaciones FCM en Android

1. Añade los siguientes permisos en el manifiesto:

```
<uses-permission android:name=
                            "android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

2. Declara el receptor dentro de la etiqueta `application` en el manifiesto insertando el código:

```
<receiver
   android:name=".ReceptorInicio"
   android:exported="true"
   android:enabled="true">
      <intent-filter>
         <action android:name="android.intent.action.BOOT_COMPLETED" />
         <action android:name="android.intent.action.QUICKBOOT_POWERON" />
      </intent-filter>
</receiver>
```

3. Crea una nueva clase llamada `ReceptorInicio` con el siguiente código:

```
public class ReceptorInicio extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      context.startService(new Intent(context,
                                          EventosFCMService.class));
   }
}
```

4. Ejecuta la aplicación.

Comprueba que, si reinicias el dispositivo o cierras la aplicación desde AJUSTES del dispositivo, la aplicación es capaz de recibir un mensaje enviado desde la consola de Firebase o desde el servidor propio.

3.1.3.4. Suscripción a temas

Hemos visto que a un dispositivo que utilice nuestra aplicación se le asigna un identificador único. Dicho identificador es el que nos permite enviarle mensajes. Los podemos enviar desde un servidor web propio o desde la consola de Firebase. Debemos almacenar los identificadores en el servidor propio para poder notificar, en la consola de Firebase se hace automáticamente. Hasta el momento, hemos enviado mensajes a todos los dispositivos que utilizan la aplicación sin hacer distinciones.

Nuestra aplicación de eventos puede enviar multitud de mensajes, no todos interesantes para el usuario. ¿Cómo podemos evitarlo y hacer que reciba los que realmente sean interesantes? Esto se soluciona con la mensajería de temas de FCM.

La mensajería de temas de Firebase Cloud Messaging nos permite enviar un mensaje a múltiples dispositivos que se suscribieron a un tema. Según el modelo de publicación/suscripción, la mensajería de temas admite un número ilimitado de suscripciones a temas. Firebase se encarga de enviar los mensajes dirigidos a los suscriptores de un tema.

Por ejemplo, los usuarios que deseen recibir eventos religiosos, quizás no deseen recibir eventos ateos. Los desarrolladores pueden elegir cualquier nombre de tema.

Para suscribirse a un tema, desde la aplicación Android se debe realizar una llamada a `MessagingsubscribeToTopic()` con el nombre del tema. Por ejemplo, `FirebaseMessaging.getInstance().subscribeToTopic("deportes")` suscribirá al usuario al tema “deportes”. Desde que el primer usuario se suscribe a un nuevo tema hasta que se da de alta el tema en la consola de Firebase, puede pasar hasta 1 día.

Para anular la suscripción, debemos realizar una llamada a `MessagingunsubscribeFromTopic()` con el nombre del tema. `FirebaseMessaging.getInstance().unsubscribeFromTopic("deportes")` dará de baja la suscripción al tema “deportes”.

Vamos a modificar nuestra aplicación para que sea capaz de suscribirse y desuscribirse a temas.

Ejercicio: Suscribir y dar de baja a usuarios en temas en FCM

1. Sustituye el contenido del archivo `menu_actividad_principal` de la carpeta `menu`:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".ActividadPrincipal">
    <item
        android:id="@+id/action_temas"
        android:orderInCategory="100"
        android:title="Suscripciones"
        app:showAsAction="never" />
</menu>
```

2. Añade un nuevo `layout` llamado `temas` con el siguiente código, que será la pantalla que utilizará el usuario para suscribirse:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:text="Selecciona tus suscripciones"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/textView2"
   />
   <CheckBox
      android:text="Deportes"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/checkBoxDeportes" />
   <CheckBox
      android:text="Teatro"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/checkBoxTeatro" />
   <CheckBox
      android:text="Cine"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/checkBoxCine" />
   <CheckBox
      android:text="Fiestas"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/checkBoxFiestas" />
</LinearLayout>
```

3. Crea una clase llamada `Temas` con el siguiente contenido:

```
public class Temas extends AppCompatActivity {
   CheckBox checkBoxDeportes;
   CheckBox checkBoxTeatro;
   CheckBox checkBoxCine;
   CheckBox checkBoxFiestas;
   @Override
```

```
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.temas);
      checkBoxDeportes = (CheckBox) findViewById(R.id.checkBoxDeportes);
      checkBoxTeatro = (CheckBox) findViewById(R.id.checkBoxTeatro);
      checkBoxCine = (CheckBox) findViewById(R.id.checkBoxCine);
      checkBoxFiestas = (CheckBox) findViewById(R.id.checkBoxFiestas);
      checkBoxDeportes.setChecked(
                              consultarSuscripcionATemaEnPreferencias(
                              getApplicationContext(), "Deportes"));
      checkBoxTeatro.setChecked(
                              consultarSuscripcionATemaEnPreferencias(
                              getApplicationContext(), "Teatro"));
      checkBoxCine.setChecked(
                              consultarSuscripcionATemaEnPreferencias(
                              getApplicationContext(), "Cine"));
      checkBoxFiestas.setChecked(
                              consultarSuscripcionATemaEnPreferencias(
                              getApplicationContext(), "Fiestas"));
      checkBoxDeportes.setOnCheckedChangeListener(
                             new CompoundButton.OnCheckedChangeListener(){
         public void onCheckedChanged(CompoundButton buttonView,
                              boolean isChecked)
         {
            mantenimientoSuscripcionesATemas("Deportes", isChecked);
         }
      });
      checkBoxTeatro.setOnCheckedChangeListener(
                             new CompoundButton.OnCheckedChangeListener(){
         public void onCheckedChanged(CompoundButton buttonView,
                              boolean isChecked)
         {
            mantenimientoSuscripcionesATemas("Teatro", isChecked);
         }
      });
      checkBoxCine.setOnCheckedChangeListener(
                             new CompoundButton.OnCheckedChangeListener(){
         public void onCheckedChanged(CompoundButton buttonView,
                             boolean isChecked)
         {
            mantenimientoSuscripcionesATemas("Cine", isChecked);
         }
      });
      checkBoxFiestas.setOnCheckedChangeListener(
                            new CompoundButton.OnCheckedChangeListener(){
         public void onCheckedChanged(CompoundButton buttonView,
                            boolean isChecked)
         {
            mantenimientoSuscripcionesATemas("Fiestas", isChecked);
         }
      });
   }
```

```
   private void mantenimientoSuscripcionesATemas(String tema,
                                                 Boolean suscribir){
      if ( suscribir )
      {
         mostrarDialogo(getApplicationContext(),
                                        "Te has suscrito a: "+tema);
         FirebaseMessaging.getInstance().subscribeToTopic(tema);
         guardarSuscripcionATemaEnPreferencias(getApplicationContext(),
                                                        tema, true);
      } else {
         mostrarDialogo(getApplicationContext(),
                                      "Te has dado de baja de: "+tema);
         FirebaseMessaging.getInstance().unsubscribeFromTopic(tema);
         guardarSuscripcionATemaEnPreferencias(getApplicationContext(),
                                                      tema, false);
      }
   }
   public static void guardarSuscripcionATemaEnPreferencias(
                  Context context, String tema, Boolean suscrito) {
      final SharedPreferences prefs = context.getSharedPreferences(
                                       "Temas", Context.MODE_PRIVATE);
      SharedPreferences.Editor editor = prefs.edit();
      editor.putBoolean(tema, suscrito);
      editor.commit();
   }
   public static Boolean consultarSuscripcionATemaEnPreferencias(
                                     Context context, String tema) {
      final SharedPreferences preferencias =
                               context.getSharedPreferences("Temas",
                               Context.MODE_PRIVATE);
      return preferencias.getBoolean(tema, false);
   }
}
```

En la clase `Temas` es donde mantenemos las suscripciones a los temas. Para ello nos valemos de los métodos: `FirebaseMessaging.getInstance().subscribeToTopic` y `FirebaseMessaging.getInstance().unsubscribeFromTopic`, a los que pasaremos como parámetro el nombre del tema. Nos apoyamos en las preferencias para guardar y consultar aquellas a las que está suscrito el usuario.

4. Recuerda dar de alta la actividad `Temas` en el manifiesto:

```
<activity
   android:name=".Temas"
   android:label="@string/app_name"
   android:theme="@style/AppTheme.NoActionBar">
</activity>
```

5. Sustituye los métodos `onCreateOptionsMenu` y `onOptionsItemSelected` en la clase `ActividadPrincipal`:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
   getMenuInflater().inflate(R.menu.menu_actividad_principal, menu);
```

```
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.action_temas) {
        Intent intent = new Intent(getBaseContext(), Temas.class);
        startActivity(intent);
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

6. Ejecuta la aplicación.

Elige el menú **SUSCRIPCIONES** y selecciona todas las suscripciones. En la consola de Firebase, duplica el último mensaje enviado. En el apartado OBJETIVO, elige **Tema**. Revisa si en el desplegable aparecen todos los temas que has seleccionado. Recuerda que puede pasar hasta 1 día para que aparezcan. Cuando aparezcan, envía un mensaje a los usuarios suscritos a un tema y comprueba que se recibe en el dispositivo.

Si quisiéramos hacer lo mismo en un servidor propio, tendríamos que mantener nosotros la relación de qué usuarios se han suscrito y dado de baja en un tema. Normalmente, esto se realiza mediante una base de datos. Realizaríamos una consulta para que nos devolviera los usuarios que están suscritos a uno o varios temas. El servidor de pruebas no está preparado para esto.

Puede que el usuario de nuestra aplicación no quiera recibir mensajes de nuestra aplicación. Podría ir a los ajustes de dispositivo y ocultar las notificaciones. Pero seguiríamos enviándole mensajes, bien desde la consola de Firebase, bien desde el servidor propio. Una buena práctica es incluir una opción en la configuración de nuestra aplicación para que no reciba mensajes y hacerlo fácil al usuario.

Si enviamos mensajes desde la consola de Firebase, podríamos incluir un tema llamado “Todos”, en el que estarían suscritos en principio todos los usuarios de nuestra aplicación. Siempre enviaríamos los mensajes por temas. Si queremos enviar a todos los usuarios de nuestra aplicación, usaríamos el tema “Todos”. Por ejemplo, un mensaje que indicara cambios de funcionalidad de la aplicación. Si quisiéramos enviar a un grupo de usuarios concreto, usaríamos un tema en concreto. Si el usuario se diera de baja del tema “Todos”, daríamos de baja al usuario de todos los temas.

Ejercicio: Restringir el envío de mensajes mediante temas

1. Inserta el siguiente código entre la declaración del `LinearLayout` y el primer `TextView` en el layout `Temas`:

```
<TextView
   android:text="No recibir notificaciones:"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:id="@+id/textView" />
<CheckBox
   android:text="No deseo recibir notificaciones"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:id="@+id/checkBoxNoRecibirNotificaciones" />
```

2. Añade el siguiente código al final del método `onCreate` de la clase `ActividadPrincipal`:

```
final SharedPreferences preferencias =
                getApplicationContext().getSharedPreferences("Temas",
                                                    Context.MODE_PRIVATE);
if (preferencias.getBoolean("Inicializado", false)==false){
   final SharedPreferences prefs =
                        getApplicationContext().getSharedPreferences(
                        "Temas", Context.MODE_PRIVATE);
   SharedPreferences.Editor editor = prefs.edit();
   editor.putBoolean("Inicializado", true);
   editor.commit();
   FirebaseMessaging.getInstance().subscribeToTopic("Todos");
}
```

Inicialmente, si queremos hacer lo descrito anteriormente. Por defecto cuando el usuario abra la aplicación por primera vez, lo suscribiremos al tema "Todos". Para ello, primero vemos si en preferencias hemos inicializado la aplicación mediante la preferencia "Inicializado". Si no se ha inicializado es porque es la primera vez que accedemos. Suscribimos al usuario al tema "Todos".

3. Declara la siguiente variable en la clase `Temas`:

```
CheckBox checkBoxNoRecibirNotificaciones;
```

4. Inserta el código siguiente debajo de la instrucción `checkBoxFiestas = (CheckBox) findViewById(R.id.checkBoxFiestas);`, que encontrarás en el método `onCreate` de la clase `Temas`:

```
checkBoxNoRecibirNotificaciones =
            (CheckBox) findViewById(R.id.checkBoxNoRecibirNotificaciones);
Boolean noRecibirNotificaciones =
consultarSuscripcionATemaEnPreferencias(getApplicationContext(), "Todos");
checkBoxNoRecibirNotificaciones.setChecked(noRecibirNotificaciones);
```

```
checkBoxDeportes.setEnabled(!noRecibirNotificaciones);
checkBoxTeatro.setEnabled(!noRecibirNotificaciones);
checkBoxCine.setEnabled(!noRecibirNotificaciones);
checkBoxFiestas.setEnabled(!noRecibirNotificaciones);
```

5. Copia el siguiente código al final del método `onCreate` de la clase `Temas`:

```
checkBoxNoRecibirNotificaciones.setOnCheckedChangeListener(
                            new CompoundButton.OnCheckedChangeListener(){
   public void onCheckedChanged(CompoundButton buttonView,
                                                        boolean isChecked)
   {
      mantenimientoSuscripcionesATemas("Todos", isChecked);
   }
});
```

6. Sustituye el método `mantenimientoSuscripcionesATemas` de la clase `Temas` por el siguiente:

```
private void mantenimientoSuscripcionesATemas(String tema, Boolean
suscribir){
   if (tema.equals("Todos")) {
      if (suscribir) {
         FirebaseMessaging.getInstance().unsubscribeFromTopic(tema);
         guardarSuscripcionATemaEnPreferencias(getApplicationContext(),
                                                              tema, true);
         checkBoxDeportes.setChecked(false);
         checkBoxTeatro.setChecked(false);
         checkBoxCine.setChecked(false);
         checkBoxFiestas.setChecked(false);
      }else {
         FirebaseMessaging.getInstance().subscribeToTopic(tema);
         guardarIdRegistro(getApplicationContext(),

FirebaseInstanceId.getInstance().getToken());
         guardarSuscripcionATemaEnPreferencias(getApplicationContext(),
tema, false);
      }
      checkBoxDeportes.setEnabled(!suscribir);
      checkBoxTeatro.setEnabled(!suscribir);
      checkBoxCine.setEnabled(!suscribir);
      checkBoxFiestas.setEnabled(!suscribir);
   } else {
      if (suscribir) {
         FirebaseMessaging.getInstance().subscribeToTopic(tema);
         guardarSuscripcionATemaEnPreferencias(getApplicationContext(),
                tema, true);
      } else {
         FirebaseMessaging.getInstance().unsubscribeFromTopic(tema);
         guardarSuscripcionATemaEnPreferencias(getApplicationContext(),
                tema, false);
      }
   }
}
```

Este método se ejecuta cuando se activa o desactiva un `CheckBox`. Si el usuario activa el `CheckBox` **Todos**, da de baja al usuario en todos los temas, incluido el tema "Todos". Se deshabilitan todos los `CheckBox` de los demás temas. Si se activa, vuelve a habilitar los `CheckBox` de los temas y se suscribe al tema "Todos". No recuerda los temas a los que estaba suscrito el usuario anteriormente.

7. Ejecuta la aplicación.
8. Cuando aparezca el tema "Todos" en la consola de Firebase, envía un mensaje a todos los usuarios del tema "Todos".
9. Envía un segundo mensaje a un tema que esté suscrito el usuario.
10. En la aplicación deshabilita que se puedan recibir mensajes y vuelve a enviar un mensaje para los usuarios del tema "Todos".
11. Envía otro mensaje para los usuarios de otro tema que no sea "Todos".

Comprueba que cuando se está suscrito al tema "Todos" se reciben mensajes y que cuando no se está suscrito, no se reciben desde ninguno de los temas.

Si se utiliza un servidor propio, sería suficiente con marcar en el almacenamiento de identificadores que no se le puede enviar mensajes o simplemente eliminar el identificador.

Vamos a ver cómo desregistrar el dispositivo en nuestro servidor de pruebas.

Ejercicio: Desregistrar un dispositivo en un servidor propio FCM

1. Añade el siguiente código al final de la clase `Comun` para dar de baja el dispositivo en el servidor propio:

```
public static void eliminarIdRegistro(Context context){
    desregistrarDispositivoEnServidorWebTask tarea =
            new desregistrarDispositivoEnServidorWebTask();
    tarea.contexto=context;
    tarea.idRegistroTarea= FirebaseInstanceId.getInstance().getToken();
    tarea.execute();
}

public static class desregistrarDispositivoEnServidorWebTask
                                extends AsyncTask<Void, Void, String> {
   String response="error";
   Context contexto;
   String idRegistroTarea;
   public void onPreExecute() {
      super.onPreExecute();
   }
```

```
   @Override
   protected String doInBackground(Void... arg0) {
      try {
         Uri.Builder constructorParametros = new Uri.Builder()
                     .appendQueryParameter("iddevice", idRegistroTarea)
                     .appendQueryParameter("idapp", ID_PROYECTO);
         String parametros =
                       constructorParametros.build().getEncodedQuery();
         String url = URL_SERVIDOR + "desregistrar.php";
         URL direccion = new URL(url);
         HttpURLConnection conexion = (HttpURLConnection)
                                          direccion.openConnection();
         conexion.setRequestMethod("POST");
         conexion.setRequestProperty("Accept-Language", "UTF-8");
         conexion.setDoOutput(true);
         OutputStreamWriter outputStreamWriter = new
                     OutputStreamWriter(conexion.getOutputStream());
         outputStreamWriter.write(parametros.toString());
         outputStreamWriter.flush();
         int respuesta = conexion.getResponseCode();
         if (respuesta == 200) {
            response = "ok";
         } else {
            response = "error";
         }
      } catch (IOException e) {
         response = "error";
      }
      return response;
   }
   public void onPostExecute(String res) {
   }
}
```

2. Inserta el siguiente código, en el método `mantenimientoSuscripcionesATemas` de la clase `Temas`, antes de la instrucción `FirebaseMessaging.getInstance().unsubscribeFromTopic(tema);`:

```
eliminarIdRegistro(getApplicationContext());
```

Cuando se marque el `CheckBox` para no recibir mensajes, nos daremos de baja de todos los temas en el servicio FCM y eliminaremos el identificador de nuestro servidor web.

Si el usuario desmarca el `CheckBox` **Todos**, quiere recibir mensajes. Lo volveremos a suscribir al tema "Todos" en el servidor FCM y daremos de alta de nuevo el identificador FCM en nuestro servidor.

3. Ejecuta la aplicación y marca la opción del menú **SUSCRIPCIONES** que indica que no se quieren recibir más mensajes.

Comprueba que funciona el desregistro en el dispositivo. Accede a la siguiente dirección con el navegador de tu ordenador: http://cursoandroid.hol.es/notificaciones/listar.html e introduce el identificador de proyecto que obtuviste en la consola de Firebase. Pulsa el botón **Listar** y verifica que desaparece un identificador cuando se ha pulsado el botón de desregistro. Cuando termines, vuelve a registrar el dispositivo desmarcando **No recibir notificaciones** accediendo desde el menú **SUSCRIPCIONES**.

3.1.3.5. Personalización

Observa que cuando se recibe un mensaje y la aplicación está en segundo plano, solo se muestra el texto del mensaje en el área de notificaciones. Los datos adicionales no se muestran. Este es el comportamiento por defecto. No podemos actuar sobre la notificación para cambiar, por ejemplo, el texto a mostrar. En cuanto a los datos adicionales, se pasan en los extras de la actividad que se abre al pulsar sobre la notificación.

Si quisiéramos crear una notificación personalizada en el área de notificaciones, con los datos del mensaje recibido, lo tendríamos que hacer en `onMessageReceived`, pero o bien el método no se dispara si la aplicación está en segundo plano, o bien no tiene sentido hacerlo si la aplicación se encuentra abierta y ya mostramos un cuadro de diálogo con los datos.

Sí que podemos actuar sobre su aspecto. Cambiar el icono y el color de la notificación recibida.

Por defecto, el icono de la aplicación va a acompañar al texto de la notificación. Se puede indicar el uso de uno específico en la carga de datos del mensaje.

Podemos definir qué color será usado en la notificación. Para diferentes versiones de Android funciona de forma diferente: si se usa una versión menor a Android 6.0, el color será el color de fondo del icono. Para versiones superiores a Android 6.0, será el color de fondo del icono y del nombre de la aplicación.

Modifiquemos nuestra aplicación.

Ejercicio: Personalizar una notificación FCM recibida en la bandeja del sistema

1. Busca en Internet una imagen png 72×72 que te guste como icono de las notificaciones y descárgala. Renómbrala a `notificacion.png`
2. Copia la imagen en la carpeta drawables en los recursos de la aplicación.
3. Añade dentro de la etiqueta `application`, en el manifiesto, el siguiente `meta`:

```
<meta-data
   android:name="com.google.firebase.messaging.default_notification_icon"
   android:resource="@drawable/notificacion" />
```

Con esta pequeña modificación de la aplicación, hemos conseguido cambiar el icono de las notificaciones.

4. Para cambiar el color de la notificación en área de notificaciones, añade el siguiente `meta` dentro de la etiqueta `application` en el manifiesto:

```
<meta-data
  android:name="com.google.firebase.messaging.default_notification_color"
  android:resource="@color/colorAccent" />
```

5. Ejecuta la aplicación y ponla en segundo plano.
6. Duplica y envía el último mensaje que has enviado desde la consola de Firebase.

   Comprueba que la notificación recibida ha cambiado con respecto a las anteriores que has recibido.

Hemos visto que, cuando recibimos un mensaje podemos mostrar la información de diferentes formas, según cómo la recibimos. Si la recibimos con la aplicación abierta, será el evento `onMessageReceived` el que procese la información. Si la aplicación está en segundo plano, recibiremos la información como extras de la actividad con la que se iniciará la aplicación y mostramos un cuadro de diálogo. Es evidente que esto es bastante mejorable. Por ejemplo, si tuviéramos una actividad concreta en nuestra aplicación que mostrara los detalles de un evento, sería mejor que el cuadro de diálogo que se muestra en la aplicación.

Podemos indicar en el mensaje qué actividad concreta de nuestra aplicación abrir. Para ello utilizaremos la sobrecarga de `notification`, `click_action` (las diferentes sobrecargas las veremos en la parte del servidor) y definiremos en el manifiesto el `intent-filter` de la actividad que queremos abrir.

Ejercicio: Abrir una actividad determinada desde un mensaje

Primero vamos a crear una actividad para mostrar los detalles de un evento al pulsar sobre uno en la lista de eventos.

1. Crea un `layout` llamado `evento_detalles.xml` con el siguiente contenido:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <android.support.design.widget.AppBarLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      <android.support.v7.widget.Toolbar
         android:id="@+id/toolbar"
```

```
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary" />
   </android.support.design.widget.AppBarLayout>
   <RelativeLayout
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="0.5">
      <ImageView
         android:id="@+id/imgImagen"
         android:layout_width="match_parent"
         android:layout_height="match_parent"
         android:adjustViewBounds="true" />
   </RelativeLayout>
   <RelativeLayout
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="0.5">
      <TextView
         android:id="@+id/txtEvento"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:textAppearance="@style/TextAppearance.AppCompat.Subhead"
      />
      <TextView
         android:id="@+id/txtCiudad"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_below="@+id/txtEvento"
         android:gravity="center" />
      <TextView
         android:id="@+id/txtFecha"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_below="@+id/txtCiudad"
         android:gravity="center"
         android:textAppearance="@style/TextAppearance.AppCompat.Caption"
         android:textStyle="italic" />
   </RelativeLayout>
</LinearLayout>
```

2. Crea una nueva clase llamada `EventoDetalles` y sustituye su contenido por:

```
public class EventoDetalles extends AppCompatActivity {
   TextView txtEvento, txtFecha, txtCiudad;
   ImageView imgImagen;
   String evento;
   CollectionReference registros;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.evento_detalles);
```

```
      txtEvento = (TextView) findViewById(R.id.txtEvento);
      txtFecha = (TextView) findViewById(R.id.txtFecha);
      txtCiudad = (TextView) findViewById(R.id.txtCiudad);
      imgImagen = (ImageView) findViewById(R.id.imgImagen);
      Bundle extras = getIntent().getExtras();
      evento = extras.getString("evento");
      if (evento==null) evento="";
        registros = FirebaseFirestore.getInstance().collection("eventos");
        registros.document(evento).get().addOnCompleteListener(
                              new OnCompleteListener<DocumentSnapshot>() {
           @Override
           public void onComplete(@NonNull Task<DocumentSnapshot> task) {
              if (task.isSuccessful()) {
                 txtEvento.setText(task.getResult()
                                              .get("evento").toString());
                 txtCiudad.setText(task.getResult()
                                              .get("ciudad").toString());
                 txtFecha.setText(task.getResult()
                                               .get("fecha").toString());
                 new DownloadImageTask(
                        (ImageView)   imgImagen).execute(task.getResult()
                                              .get("imagen").toString());
              }
           }
        });
   }
   private class DownloadImageTask
                                 extends AsyncTask<String, Void, Bitmap> {
      ImageView bmImage;
      public DownloadImageTask(ImageView bmImage) {
         this.bmImage = bmImage;
      }
      protected Bitmap doInBackground(String... urls) {
         String urldisplay = urls[0];
         Bitmap mImagen = null;
         try {
            InputStream in = new java.net.URL(urldisplay).openStream();
            mImagen = BitmapFactory.decodeStream(in);
         } catch (Exception e) {
            e.printStackTrace();
         }
         return mImagen;
      }
      protected void onPostExecute(Bitmap result) {
         bmImage.setImageBitmap(result);
      }
```

```
    }
}
```

3. Añade al final de la clase `ActividadPrincipal`:

```
public static Context getAppContext() {
    return ActividadPrincipal.getCurrentContext();
}
```

4. Añade el siguiente código al final del método `onCreate` de la clase `ActividadPrincipal`:

```
adaptador.setOnItemClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        int position = recyclerView.getChildAdapterPosition(view);
        Evento currentItem = (Evento) adaptador.getItem(position);
        String idEvento =
                    adaptador.getSnapshots().getSnapshot(position).getId();
        Context context = getAppContext();
        Intent intent = new Intent(context, EventoDetalles.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        intent.putExtra("evento", idEvento);
        context.startActivity(intent);
    }
});
```

5. Declara la clase `EventoDetalles` en el manifiesto:

```
<activity
    android:name=".EventoDetalles"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.NoActionBar"
    android:exported="true">
</activity>
```

6. Ejecuta la aplicación.

   Comprueba que, al pulsar sobre un evento en la lista inicial de eventos, se abre una nueva pantalla con los detalles del evento.

   Al pulsar sobre un mensaje en el área de notificaciones, por defecto, se abre la actividad inicial de la aplicación. Vamos a indicar a la aplicación que se abra la actividad que acabamos de crear con los detalles de un evento cuando se reciba un cierto tipo de mensajes.

7. Añade el siguiente `intent-filter` en la declaración de la actividad `EventoDetalles` en el manifiesto.

```
<intent-filter>
    <action android:name="OPEN_ACTIVITY_1" />
```

```
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

8. Ejecuta la aplicación y ocúltala en segundo plano.
9. Accede al servidor propio, http://cursoandroid.hol.es/notificaciones/notificar.html , y envía el siguiente mensaje:

Notificación

```
Mensaje: Prepárate para las Fallas
click_action: OPEN_ACTIVITY_1
Carga de datos
evento: fallas
```

Comprueba que has recibido el mensaje y al pulsar sobre él, se abre la actividad que muestra los detalles del evento “fallas”. Al utilizar el campo de `click_action` se va a abrir la actividad que coincida con su valor con el del `intent-filter` declarado en el manifiesto. La actividad obtiene mediante extras de los datos del mensaje. En nuestro caso el identificador del evento en el valor del campo “evento”.

Práctica: Mensajes ampliados

Cuando la aplicación está abierta y recibimos un mensaje, se muestra un cuadro de diálogo con la información. Modifica la aplicación para que la aplicación se comporte de la misma forma que si recibimos un mensaje con `click_action` y se encuentra en segundo plano. Debes mostrar un diálogo y al cerrarlo, se debe abrir la actividad `EventoDetalles` con los datos del evento.

3.1.4. Aplicación servidor Firebase Cloud Messaging

Resulta imprescindible disponer de un servidor vinculado a una aplicación FCM. Necesitaremos almacenar los identificadores FCM y un lugar desde donde enviar las notificaciones.

Para una aplicación servidor de FCM, podemos utilizar la consola de Firebase o diferentes lenguajes de programación y bases de datos para implementar el servidor web de FCM.

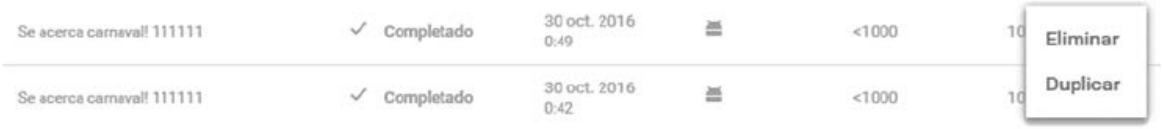
3.1.4.1. Servidor: Firebase Notifications

Notifications es una funcionalidad que podemos encontrar en la consola de Firebase y que nos va a permitir enviar mensajes a los dispositivos que usan nuestra aplicación sin desarrollar un servidor propio.

Notifications se integra con Firebase Analytics, lo que nos permite orientar los mensajes a usuarios que cumplen ciertas características. Podemos enviar mensajes a segmentos de usuarios predefinidos, que utilizan cierta versión de la aplicación o idioma. Al orientar los mensajes a segmentos de usuarios, podemos contactar exactamente con aquellos usuarios que deseemos que lo reciban.

Para enviar un mensaje con Notifications tenemos varias opciones:

- **Nuevo mensaje:** Rellenaremos un mensaje vacio.
- **Duplicar:** Partiendo de un mensaje ya creado, lo podemos duplicar seleccionando **Duplicar** en las opciones del mensaje.

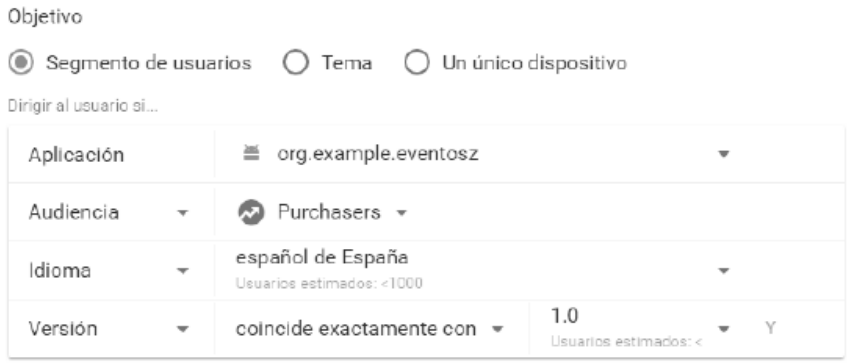


Vamos a ver todos los campos de un mensaje en Notifications:

NOTIFICACIÓN

- **Texto del mensaje**: Mensaje que se va a mostrar al usuario en área de notificaciones.
- **Etiqueta del mensaje**: Nombre empleado para identificar el mensaje en Firebase. El nombre no se muestra a los usuarios.
- **Fecha de entrega**: Momento en que debe enviarse el mensaje. Podemos enviarlo ahora o programarlo con 1 mes de antelación como máximo.

DESTINO



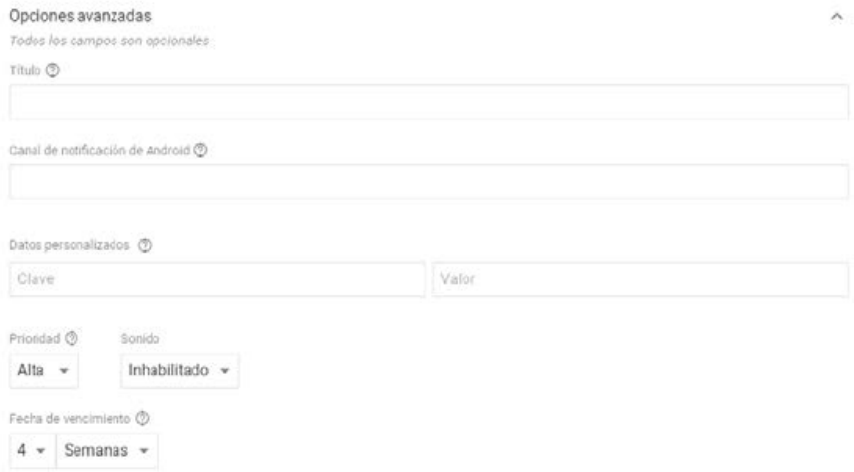
- **Segmento de usuarios:** Podemos elegir una o varias aplicaciones, dadas de alta en el proyecto. Además, gracias a la integración con Analytics, podemos determinar la audiencia, el idioma y la versión de la aplicación para enviar el mensaje a los usuarios que cumplan las condiciones.
- **Tema:** Mecanismo para enviar mensajes a los suscriptores de un tema. Puede haber hasta 1 día de retraso entre la creación de un nuevo tema y el momento en el que está disponible como destino.
- **Un único dispositivo:** Envía un mensaje a un único dispositivo. Introduce el identificador de registro FCM del dispositivo.

EVENTOS DE CONVERSIÓN

Al marcar un evento como una conversión, podemos obtener informes y retroalimentación del evento. Son los eventos clave para evaluar la eficacia de un mensaje. Aparecerán en las estadísticas. Los eventos de conversión son actualizados inmediatamente para evaluar la eficacia rápidamente. Los estados enviado y abierto aparecen de forma predeterminada.

Para marcar un evento como una conversión, tenemos que acceder a Firebase Analytics del proyecto. En la pestaña **Eventos**, marcar o desmarcar el evento deseado como conversión.

OPCIONES AVANZADAS

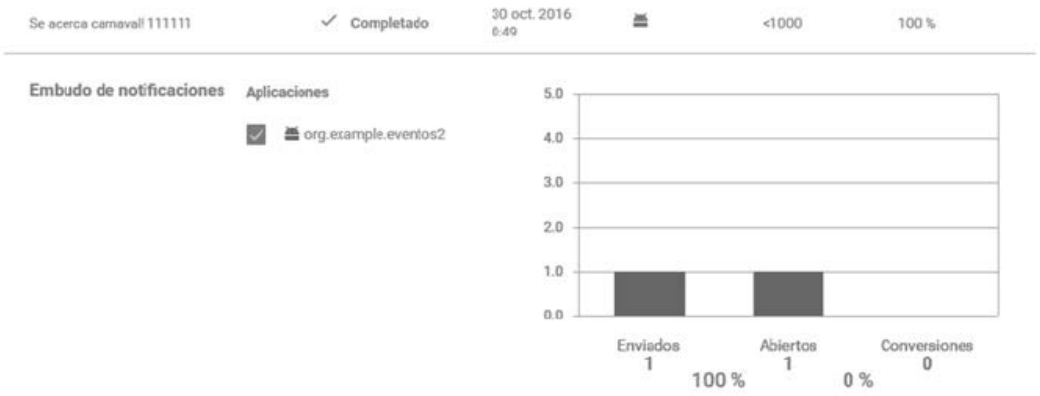


- **Título:** Se muestra a los usuarios finales como título del mensaje.
- **Canal de notificación de Android:** El nombre del canal de notificación de Android. Android O incluye los canales de notificaciones, una nueva forma de gestionar mensajes por grupos y prioridad.

- **Datos personalizados:** Pares clave-valor incluidos en el mensaje que se enviará a la aplicación.
- **Prioridad:** Nivel de prioridad de este mensaje. Si un mensaje se envía a varias plataformas, se aplica el mismo nivel de prioridad a todas ellas.
- **Sonido:** Habilitado/Deshabilitado.
- **Fecha de caducidad:** Tiempo durante el que debe guardarse el mensaje para poder reenviarlo. El periodo máximo es de 4 semanas a partir del primer intento de envío.

Una vez que hayamos rellenado todos los campos necesarios para el envío del mensaje, pulsamos en el botón **Enviar mensaje**. Podremos revisar el mensaje en una pantalla de confirmación de envío. Pulsamos el botón **Enviar** si todo está correcto. El mensaje pasará a estar en estado En curso. Cuando se envíe, pasará a estado Completado.

Una vez en estado completado, podremos ver las estadísticas pulsando sobre el mensaje.



La tabla de mensajes de Notifications contiene la siguiente información en sus columnas:

- **Mensaje:** Muestra el título (si se ha rellenado) y el mensaje.
- **Estado:** Los posibles estados son: Borrador, Programado, En curso, Completado y Error. Podemos guardar un mensaje como borrador, si al editarlo, pulsamos el botón **Guardar como borrador** en vez de enviar el mensaje.
- **Plataforma:** Plataformas a las que se ha enviado el mensaje.
- **Público estimado:** Número estimado de dispositivos a los que llegará este mensaje.
- **Porcentaje de notificaciones abiertas:** Porcentaje de mensajes abiertos respecto a los enviados.

3.1.4.2. Servidor: Propio

Si no deseamos utilizar Notifications o, además, queremos utilizar un servidor propio, lo podemos desarrollar en multitud de lenguajes de programación y opciones de almacenamiento. Es básico tener un sistema de almacenamiento para los identificadores FCM de los usuarios. Lo más habitual es que almacenemos la información en una base de datos. Realizaremos el mantenimiento del almacenamiento de identificadores y el envío de los mensajes mediante un lenguaje de programación.

Por ejemplo, podemos utilizar los siguientes lenguajes de programación con sus sistemas de almacenamiento de base de datos más afines:

- Tomcat + Java: http://developer.android.com/google/gcm/demo.html
- ASP.Net + SQL Server: http://www.sgoliver.net/blog/?p=2861
- PHP + mySQL: Este manual

Hemos elegido PHP y mySQL, por ser la combinación de lenguaje de programación y base de datos muy utilizada en los sitios web actualmente. Para realizar esta parte, necesitarás ciertos conocimientos de HTML, PHP y mySQL, además de un servidor web que las soporte y esté disponible en Internet. Te recomiendo que si no dispones de uno propio busques uno gratuito. Por ejemplo: http://www.hostinger.es.

Seguidamente vamos a crear una aplicación web para enviar mensajes a la aplicación de los dispositivos.

Ejercicio: Crear la aplicación del servidor con PHP y mySQL [OPCIONAL]

1. Crea una base de datos mySQL en el servidor.

   Al crear la base de datos tendremos que tener en cuenta los siguientes datos, que serán utilizados en los ficheros PHP posteriores:

   - nombre_servidor: Nombre del servidor donde se encuentra alojada la base de datos. Normalmente: “localhost”
   - base_datos: Nombre de la base de datos mySQL donde se alojarán los datos.
   - nombre_usuario: Nombre del usuario con acceso a la base de datos.
   - password: Contraseña que utilizará el usuario para acceder a la base de datos.

2. Crea la tabla dispositivos en la base de datos con el siguiente campo:

Campo	Tipo	Tamaño	Índice
Iddevice	Varchar	255	PRIMARY

Hemos creado una base de datos mySQL con una tabla llamada dispositivos, en la que se almacenarán los identificadores de registro en FCM de los dispositivos en el campo `iddevice`. Un registro provocará que añadamos un registro a la tabla y un desregistro hará que eliminemos un registro de la tabla.

3. Crea el archivo PHP para registrar los dispositivos (registrar.php):

```
<?php
   //Conexión a la base de datos
   $conexion    =    mysqli_connect("nombre_servidor",    "nombre_usuario",
"pasword") or die
         (mysqli_error());
   mysqli_select_db($conexion, "base_datos");
   //Insertamos id de registro devuelto por el FCM.
   mysql_query($conexion, "INSERT INTO dispositivos (iddevice) VALUES
         ('".$_POST["iddevice"]."')") or die(mysqli_error());
   mysqli_close();
?>
```

El registro del dispositivo provocará que añadamos un registro en la tabla dispositivos. Haremos el registro cuando obtengamos el identificador del FCM. Añadiremos el identificador de registro en FCM desde la aplicación Android mediante una llamada HTTP POST. Recogeremos su valor en registrar.php mediante el valor recogido en la variable `$_POST['iddevice']`.

4. Crea el archivo PHP para desregistrar dispositivos (desregistrar.php):

```
<?php
   //Conexión a la base de datos
   $conexion    =    mysqli_connect("nombre_servidor",    "nombre_usuario",
"password") or die
         (mysqli_error());
   mysqli_select_db($conexion, "base_datos");
   //Eliminamos el dispositivo basándonos en el id de registro del FCM.
   $sql = "DELETE FROM dispositivos WHERE
         iddevice='".$_POST["iddevice"]."'";
   mysqli_query($conexion, $sql) or die(mysqli_error());
   mysqli_close();
?>
```

El desregistro del dispositivo provocará que eliminemos un registro en la tabla dispositivos. Haremos el desregistro cuando el usuario desregistre su dispositivo o cuando FCM varíe el identificador. Eliminaremos el registro de la tabla que tenga en el campo `iddevice` el mismo valor que le enviaremos desde la aplicación Android mediante una llamada HTTP POST. Recogeremos su valor en desregistrar.php mediante el valor recogido en la variable `$_POST['iddevice']`.

5. Crea el archivo HTML notificar.html con un formulario para introducir el mensaje a enviar. Crear el archivo PHP notificar.php que conformará el mensaje y lo enviará al FCM para que este lo envíe a los dispositivos.

Formulario HTML para introducir el mensaje (notificar.html):

```
<html>
   <head>
   </head>
   <body>
      <form action="notificar.php" name="formulario" method="post">
         Mensaje:<br>
         <input type="text" name="mensaje"><br>
         <input type="submit" name="btnEnviar" id="btnEnviar"
               value="Enviar notificación" />
       </form>
   </body>
</html>
```

El formulario tendrá el siguiente aspecto:

Introduciremos el mensaje que queremos enviar en la caja de texto. Cuando pulsemos el botón **Enviar notificación**, llamaremos al fichero notificar.php que será el encargado de realizar la acción de enviar el mensaje.

6. Crea un archivo PHP para procesar el formulario y enviar los mensajes (notificar.php):

```
<?php
   $host = "nombre_servidor";
   $user = "nombre_usuario";
   $pass = "password";
   $database = "base_datos";

   //Conexión a la base de datos
   $conexion = mysqli_connect ($host, $user, $pass) or die ('Error al
         conectar con el servidor'.mysqli_error());
    mysqli_select_db($conexion, $database) or die ('->>Error seleccionando
la base de datos'.mysqli_error());

   if ( $_POST['mensaje'] != "") {
      $message =$_POST['mensaje'];
      //Cambiar Token de Firebase Cloud Messaging
      $apiKey = "Token de Firebase Cloud Messaging";
      $result=mysqli_query($conexion, "SELECT * FROM dispositivos");
      while($row = mysqli_fetch_assoc ( $result )) {
          //Recuperamos el id de registro del dispositivo en FCM
          $deviceToken = $row['iddevice'];
          //IMPORTANTE: Array con la información que enviará la
          //notificación.
          $data = array(
             'to' => $deviceToken,
             'collapse_key' => 'col_key',
             'notification' => array(
                'body' => $message)
          );
```

```
          //Código para conectar con FCM y enviar notificación.
          // No modificar.
          $ch = curl_init();
          curl_setopt($ch, CURLOPT_URL,
                "https://fcm.googleapis.com/fcm/send");
          $headers = array('Authorization:key=' . $apiKey,
                                'Content-Type: application/json');
          $data=json_encode($data);
          curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
          curl_setopt($ch, CURLOPT_POST, true);
          curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
          curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
          $resultado = curl_exec($ch);
          curl_close($ch);
       };
   }
?>
```

El fichero notificar.php lo que hace es obtener el mensaje introducido en notificar.html mediante `$_POST['mensaje']`. Para cada registro de la tabla dispositivos, obtendremos el identificador de registro de cada dispositivo en el FCM mediante el campo `iddevice`. Para cada uno de los registros conformaremos el mensaje y se lo enviaremos.

Para conformar el mensaje rellenaremos un array con los siguientes datos:

- `to`: Dispositivo destino mediante el identificador de registro en el FCM del dispositivo, que lo obtendremos de la tabla dispositivos.
- `collapse_key`: Es la clave de colapso, es importante, porque indica cómo se va a comportar la cola de mensajes. Imaginemos que el dispositivo está apagado, FCM guarda la notificación hasta que esté activo. Si se envía un segundo mensaje y `collapse_key` es la misma, el primer mensaje es desechado. Si es diferente, el dispositivo recibirá 2 mensajes cuando se active.
- `notification`: Información propia de la notificación, no de la carga de datos. La información se envía en un `array`. Hemos utilizado el campo `body`, que es el equivalente al campo Mensaje en Notifications.

Estos son los mínimos datos que necesitas para enviar un mensaje. Puedes enviar tanta información o más que la que se envía desde Notifications en la consola de Firebase.

En el `array` que enviamos con el mensaje, hemos enviado otro `array notification` con información específica de la notificación, pero sin carga de datos. Podemos añadir un segundo array con la carga de datos, mediante un `array` llamado `data`.

Vamos a ver unas cuantas características de ambos arrays:

Campos de `notification`:

- `body`: Texto del mensaje. Se mostrará en la bandeja de entrada del dispositivo del usuario.

- `click_action`: Indica qué acción, qué actividad se va a abrir, cuando se pulse sobre la notificación mostrada en la bandeja de entrada.
- `title`: Título del mensaje. Invisible al usuario.

Campos de `data`: Campos personalizados, formados por pares nombre y valor que contendrán la carga de datos del mensaje. Podremos crear tantos como necesitemos. Por ejemplo, podremos crear `'evento' => $evento`. Mandaremos un campo llamado `'evento'` con un valor determinado por el valor de la variable `$evento`.

Preguntas de repaso: Mensajes PUSH

Práctica: Enviar mensajes desde la aplicación

Para enviar mensajes tenemos que acceder al servidor web y rellenar un formulario. Podemos sustituir el formulario web por una llamada desde la aplicación. Fíjate en las funciones registrarDispositivoEnServidorWeb y desregistrarDispositivoEnServidorWeb de la clase EventosAplicacion. Son muy parecidas, prácticamente lo que cambia es la dirección donde van a realizar la llamada HTTP POST y los parámetros que se envían:

```
Uri.Builder constructorParametros = new Uri.Builder()
        .appendQueryParameter("iddevice", regId)
        .appendQueryParameter("idapp", SENDER_ID);
String parametros = constructorParametros.build().getEncodedQuery();

String url = SERVER_URL + "registrar.php";
URL direccion = new URL(url);
HttpURLConnection conexion = (HttpURLConnection)
        direccion.openConnection();
```

El resto de código de ambos métodos lo que hace es realizar la llamada y comprobar el resultado.

Al ser una llamada a un recurso fuera del dispositivo, la necesitas realizar en una actividad, debes utilizar obligatoriamente un hilo.

Para enviar un mensaje desde la aplicación, debes utilizar el servidor del curso.

Crea una nueva opción de menú en la clase ActividadPrincipal llamada “Enviar evento”. Al pulsar sobre ella accederás a una nueva actividad compuesta por un texto Mensaje, un EditText y un botón **Enviar**. El usuario introducirá en el EditText un mensaje y cuando pulse un botón **Enviar**, todos los usuarios recibirán el mensaje. Crea una llamada HTTP POST a notificar.php con los siguientes parámetros:

- apiKey: API de acceso

- idapp: número de proyecto
- mensaje: mensaje a enviar

Los nombres de los parámetros tienen que ser exactamente los indicados arriba.

3.2. Almacenamiento en la nube

Cuando se crea una aplicación Android, pueden surgir necesidades de almacenamiento que no pueden ser cubiertas solamente con el disponible en el dispositivo. A veces es conveniente que ciertos ficheros se almacenen fuera del mismo, bien sea por seguridad, para compartirlo con otros usuarios, etc. Una buena forma de realizarlo es aprovechando el almacenamiento en la nube, un espacio de almacenamiento que dependerá de lo que tengamos contratado y accesible desde casi cualquier dispositivo que disponga de conexión a Internet. Nosotros vamos a centrarnos en dos servicios de almacenamiento: Firebase Storage y Google Drive.

3.2.1. Firebase Storage

Firebase Storage se creó para almacenar y proporcionar contenido generado por el usuario, como fotos o vídeos. Ofrece la posibilidad de subir y descargar archivos de forma segura desde aplicaciones de Firebase, independientemente de la calidad de la red. Podemos almacenar imágenes, audio, vídeo y otro contenido generado por el usuario. Está respaldado por Google Cloud Storage, un servicio de almacenamiento de objetos potente, simple y rentable.

Las funciones clave de Firebase Storage son:

- **Robusto:** Firebase Storage realiza subidas y descargas independientemente de la calidad de la red. Se reinician donde se detuvieron y permiten ahorrar tiempo y ancho de banda.
- **Seguro:** Se integra con Firebase Authentication para proporcionar autentificación simple e intuitiva. Podemos usar el modelo de seguridad declarativa para permitir el acceso en función del nombre del archivo, el tamaño, el tipo de contenido y otros metadatos.
- **Escalabe**: Está respaldado por Google Cloud Storage, que permite el almacenamiento de petabytes. Pasa sin esfuerzos de un prototipo a producción usando la misma infraestructura que respalda a Snapchat.

Firebase Storage se usa para subir y bajar archivos directamente de los clientes. Los archivos se almacenan en Google Cloud Storage, lo que significa que podemos acceder a ellos a través de Firebase y de las API de Google Cloud. Esto permite subir y descargar archivos a través de Firebase y realizar procesamientos en el servidor, como filtrado de imágenes o transcodificación de vídeo utilizando Google Cloud Platform. Esta integración nos permite acceder directamente a los archivos desde bibliotecas cliente de Google Cloud Storage, de modo que podemos usar Firebase Storage con nuestros lenguajes preferidos en el servidor.

Firebase Storage se integra perfectamente con Firebase Authentication para identificar usuarios y proporciona seguridad que permite controlar archivos individuales o grupos. Podemos hacer que los archivos sean públicos o privados.

Los archivos se almacenan en una cubeta de Firebase Storage. Se presentan en una estructura jerárquica, como el sistema de archivos en nuestro disco duro local o los datos en la Firebase Realtime Database.

3.2.1.1. Referencias

Creamos una referencia para cargar, descargar o eliminar un archivo, o para obtener o actualizar sus metadatos. Una referencia es un puntero a un archivo en la nube. Las referencias son ligeras, por lo que podemos crear todas las que necesitemos. También son reutilizables para múltiples operaciones.

Las referencias se crean en Android desde el servicio Firebase Storage llamando al método `getReferenceFromUrl` y con el URL de la cubeta donde se almacenan los archivos con la forma `gs://<url de almacenamiento>`. Podemos encontrar el URL en la sección STORAGE de la consola de Firebase.

```
StorageReference storageRef = storage.getReferenceFromUrl(
                                        "gs://eventos-99dd8.appspot.com");
```

Se puede crear una referencia a una ubicación inferior en el árbol; por ejemplo, `"imagenes/flor.jpg"`, usando el método `child()`.

```
StorageReference imagenesRef = storageRef.child("imagenes");
StorageReference florRef = storageRef.child("imagenes/flor.jpg");
```

Podemos usar los métodos `getParent()` y `getRoot()` para navegar hacia arriba en la jerarquía de archivos. `getParent()` navega un nivel hacia arriba, mientras `getRoot()` navega a la raíz de la cubeta.

```
imagenesRef = florRef.getParent();
StorageReference raizRef = florRef.getRoot();
```

`child()`, `getParent()` y `getRoot()` se pueden encadenar múltiples veces, ya que cada uno devuelve una referencia. Pero al llamar a `getRoot().getParent()`, devuelve `null`.

```
StorageReference tierraRef = florRef.getParent().child("tierra.jpg");
StorageReference nullRef = florRef.getRoot().getParent();
```

Los métodos `getPath()`, `getName()` y `getBucket()` permiten obtener la ruta completa, el nombre y la cubeta del archivo.

```
spaceRef.getPath();
spaceRef.getName();
spaceRef.getBucket();
```

Las rutas y los nombres de las referencias pueden contener cualquier secuencia de caracteres Unicode válidos, pero se aplican ciertas restricciones. La longitud total de `reference.fullPath` debe ser de 1 a 1024 bytes cuando posea codificación UTF-8. No debe contener saltos de línea ni saltos de sección. Se debe

evitar usar #, [,], * o ?, ya que no funcionan bien con otras herramientas como Firebase Realtime Database o gsutil.

Vamos a añadir Firebase Storage a nuestro proyecto “Eventos”. Como la aplicación ya usa Firebase, obviaremos los pasos que nos permiten conectar una aplicación Android con un proyecto de Firebase. Pero recuerda que si la aplicación todavía no está conectada con Firebase, tienes que realizarlos.

Ejercicio: Usar Firebase Storage en una aplicación Android

1. Añade la siguiente dependencia al fichero `build.gradle (Module: app)`:

```
implementation 'com.google.firebase:firebase-storage:11.8.0'
```

   Por defecto, Firebase Storage tiene activada la autentificación para acceder a los archivos que almacena. Aunque es una buena práctica y no es recomendable publicar ninguna aplicación de almacenamiento sin la autentificación, vamos a modificar las reglas para permitir el acceso total sin identificación.

2. Accede a la consola de Firebase y selecciona el proyecto “Eventos”. Pulsa sobre **Storage** en el apartado DEVELOP del menú de la izquierda.

3. Selecciona la pestaña **Reglas** y cambia las reglas que permiten el acceso con autentificación por la siguiente, que permite el acceso público, tanto a lectura como a escritura:

```
service firebase.storage {
   match /b/{bucket}/o {
      match /{allPaths=**} {
      allow read, write;
      }
   }
}
```

4. Declara las siguientes variables en la clase `Comun`:

```
static FirebaseStorage storage;
static StorageReference storageRef;
```

5. Inserta al final del método `onCreate` de la clase `ActividadPrincipal`:

```
storage = FirebaseStorage.getInstance();
storageRef = storage.getReferenceFromUrl(
    "gs://eventos-xxxx.appspot.com");
```

   Sustituye el URL que utilizamos como parámetro en `storage.getReferenceFromUrl` por el tuyo propio. Lo podrás encontrar en la pestaña **Archivos** de Firebase Storage en la consola de Firebase.

6. Añade al final de la clase `Comun`:

```
public static StorageReference getStorageReference() {return storageRef;}
```

Hemos implementado el código necesario para inicializar Firebase Storage en nuestra aplicación Android. Para ello hemos añadido las dependencias necesarias en el archivo `gradle`. Al iniciar la aplicación creamos una instancia con `FirebaseStorage.getInstance()`. Por último, inicializamos una referencia al contenedor de nuestro proyecto con `getReferenceFromUrl("Url_contenedor")`. A partir de la referencia, será donde crearemos otras referencias hijas para subir, consultar, modificar o borrar archivos.

3.2.1.2. Subir archivos

Firebase Storage permite subir archivos de forma rápida y fácil a Google Cloud Storage, administrado por Firebase. De forma predeterminada, se requiere autentificación para acceder a Firebase Storage, podemos permitir el acceso sin autentificación si cambiamos las reglas en la consola de Firebase.

Para subir un archivo, primero debemos crear una referencia a la ruta completa del archivo, en la que se incluya su nombre. Una vez creada la referencia, podemos utilizar los siguientes métodos para subir el archivo a Firebase: `putBytes()`, `putFile()` o `putStream()`.

No se pueden subir datos a una referencia a la raíz del almacenamiento, debe apuntar a un URL secundario.

- Subir datos desde la memoria: `putBytes()`

  Usar el método `putBytes()` es la forma más sencilla de subir datos a Firebase Storage. Requiere como parámetros un `byte[]` y devuelve una `UploadTask`, con la que podemos administrar y controlar el estado de la subida. `putBytes()` es el método más ineficiente en cuanto a uso de memoria. Esto es debido a que requiere como parámetro un `byte[]`, para ello debemos cargar en memoria el archivo entero que queremos subir. Si consideras que esto es un problema, utiliza `putStream()` o `putFile()` para usar menos memoria.

- Subir datos en streaming (retransmisión): `putStream()`

  El método `putStream()` es la forma más versátil de subir un archivo. Requiere como parámetro un `InputStream` y devuelve una `UploadTask`.

- Subir desde un archivo local: `putFile()`

  `putFile()` permite subir archivos del dispositivo. Requiere como parámetro un `File` y retorna una `UploadTask`.

Podemos agregar receptores para controlar el éxito, fallo, progreso o pausa de la tarea que se encarga de subir o descargar un archivo:

Tipo de receptor	Uso típico
`OnProgressListener`	Llamado con frecuencia mientras se transfieren datos Se puede usar para mostrar un indicador de subida/descarga
`OnPausedListener`	Llamado cada vez que se pausa la tarea
`OnSuccessListener`	Llamado cuando la tarea finaliza correctamente
`OnFailureListener`	Llamado cuando falla la subida/descarga Esto puede deberse a tiempos de espera de la red, errores de autorización o si se cancela la tarea

Cuando un receptor es llamado, se obtiene un objeto `UploadTask.TaskSnapshot`. Este objeto es una vista inmutable de la tarea en el momento que ocurre el evento.

Vamos a ver un ejemplo de cómo utilizar `putBytes()` , `putStream()`y `putFile()`.Si te has dado cuenta, las imágenes de los eventos las obtenemos de un servidor externo a Firebase. Vamos a almacenarlas en Firebase Storage. Para ello utilizaremos `putBytes()` a partir de la imagen del `ImageView` de cada evento que se muestra al acceder a los detalles del mismo. Utilizaremos `putStream()` y `putFile()` para subir un archivo que el usuario seleccionará en el dispositivo.

Ejercicio: Subir datos a Firebase Storage

1. Crea un nuevo menú llamado `menu_detalles` con el siguiente contenido:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   tools:context="org.example.eventos.ActividadPrincipal">
   <item
      android:id="@+id/action_putData"
      android:orderInCategory="100"
      android:title="Subir Imagen"
      app:showAsAction="never" />
   <item
      android:id="@+id/action_streamData"
      android:orderInCategory="101"
      android:title="Subir Stream"
      app:showAsAction="never" />
   <item
      android:id="@+id/action_putFile"
      android:orderInCategory="102"
      android:title="Subir Fichero"
      app:showAsAction="never" />
</menu>
```

2. Declara las siguientes constantes y variables en la clase `EventoDetalles`:

```
final int SOLICITUD_SUBIR_PUTDATA = 0;
final int SOLICITUD_SUBIR_PUTSTREAM = 1;
final int SOLICITUD_SUBIR_PUTFILE = 2;
final int SOLICITUD_SELECCION_STREAM = 100;
final int SOLICITUD_SELECCION_PUTFILE = 101;
private ProgressDialog progresoSubida;
Boolean subiendoDatos =false;
```

3. Copia el siguiente código al final del método `onCreate` de la clase `EventoDetalles`:

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```

4. En la clase `EventoDetalles`, activa el menú que acabas de crear añadiendo el siguiente código al final de la clase:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
   getMenuInflater().inflate(R.menu.menu_detalles, menu);
   return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   View vista = (View) findViewById(android.R.id.content);
   int id = item.getItemId();
   switch (id) {
      case R.id.action_putData:
         subirAFirebaseStorage(SOLICITUD_SUBIR_PUTDATA,null);
         break;
      case R.id.action_streamData:
         seleccionarFotografiaDispositivo(vista,
                                          SOLICITUD_SELECCION_STREAM);
         break;
      case R.id.action_putFile:
         seleccionarFotografiaDispositivo(vista,
                                         SOLICITUD_SELECCION_PUTFILE);
         break;
   }
   return super.onOptionsItemSelected(item);
}
```

El usuario puede seleccionar tres opciones:

- `action_putData`: subir el contenido del `ImageView` a un archivo en Firebase Storage mediante `putBytes`.
- `action_streamData`: seleccionar un archivo del dispositivo y subirlo a Firebase Storage mediante `putStream`.
- `action_putFile`: seleccionar un archivo del dispositivo y subirlo a Firebase directamente mediante `putFile`.

5. Copia al final de la clase `EventoDetalles` el método encargado de seleccionar una fotografía del dispositivo:

```
public void seleccionarFotografiaDispositivo(View v, Integer solicitud) {
   Intent seleccionFotografiaIntent = new Intent(Intent.ACTION_PICK);
   seleccionFotografiaIntent.setType("image/*");
   startActivityForResult(seleccionFotografiaIntent, solicitud);
}
```

El método `seleccionarFotografiaDispositivo` lanzará una actividad de una aplicación (Galeria, Fotos… cualquier aplicación que tenga permiso para abrir imágenes) que permitirá al usuario seleccionar una imagen. La ruta del archivo elegido será capturada en `onActivityResult`.

6. Añade al final de la clase `EventoDetalles` el método `onActivityResult`, que obtendrá la ruta del archivo elegido por el usuario:

```
@Override
protected void onActivityResult(final int requestCode,
                                final int resultCode, final Intent data) {
   Uri ficheroSeleccionado;
   Cursor cursor;
   String rutaImagen;
   if (resultCode == Activity.RESULT_OK) {
      switch (requestCode) {
         case SOLICITUD_SELECCION_STREAM:
            ficheroSeleccionado = data.getData();
            String[] proyeccionStream = {MediaStore.Images.Media.DATA};
            cursor = getContentResolver().query(ficheroSeleccionado,
                                    proyeccionStream, null, null, null);
            cursor.moveToFirst();
            rutaImagen = cursor.getString(
                           cursor.getColumnIndex(proyeccionStream[0]));
            cursor.close();
            subirAFirebaseStorage(SOLICITUD_SUBIR_PUTSTREAM, rutaImagen);
            break;
         case SOLICITUD_SELECCION_PUTFILE:
            ficheroSeleccionado = data.getData();
            String[] proyeccionFile = {MediaStore.Images.Media.DATA};
            cursor = getContentResolver().query(ficheroSeleccionado,
                                    proyeccionFile, null, null, null);
            cursor.moveToFirst();
            rutaImagen = cursor.getString(
                                cursor.getColumnIndex(proyeccionFile[0]));
            cursor.close();
            subirAFirebaseStorage(SOLICITUD_SUBIR_PUTFILE, rutaImagen);
            break;
      }
   }
}
```

Guardamos en la variable `rutaImagen` la ruta del archivo. En el método `subirAFirebaseStorage` será donde subiremos la imagen a Firebase

Storage. Desde `onActivityResult` llamaremos a `subirAFirebaseStorage` si hemos elegido `putStream` o `putFile`, y desde `onOptionsItemSelected` si elegimos `putBytes` (no elegimos un archivo, sino que subimos una imagen en memoria).

7. Declara la tarea de que subirá los datos y la referencia `StorageReference` al archivo en Firebase Storage en la clase `EventoDetalles`:

```
static UploadTask uploadTask=null;
StorageReference imagenRef;
```

8. Crea al final de la clase `EventoDetalles` el método `subirAFirebaseStorage` insertando el siguiente código:

```
public void subirAFirebaseStorage(Integer opcion,
                                          String ficheroDispositivo) {
   String fichero = evento;
   imagenRef = getStorageReference().child(fichero);
   try {
      switch (opcion) {
         case SOLICITUD_SUBIR_PUTDATA:
            imgImagen.setDrawingCacheEnabled(true);
            imgImagen.buildDrawingCache();
            Bitmap bitmap = imgImagen.getDrawingCache();
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos);
            byte[] data = baos.toByteArray();
            uploadTask = imagenRef.putBytes(data);
            break;
         case SOLICITUD_SUBIR_PUTSTREAM:
            InputStream stream = new FileInputStream(
                                          new File(ficheroDispositivo));
            uploadTask = imagenRef.putStream(stream);
            break;
         case SOLICITUD_SUBIR_PUTFILE:
            Uri file = Uri.fromFile(new File(ficheroDispositivo));
            uploadTask = imagenRef.putFile(file);
            break;
      }
   } catch (IOException e) {
      mostrarDialogo(getApplicationContext(), e.toString());
   }
}
```

En el método `subirAFirebaseStorage` definimos una referencia al fichero que vamos a subir. Para ello creamos una nueva referencia Firebase Storage con `FirebaseStorage.getInstance().child(fichero)`. En función de la opción elegida, usaremos para subir el fichero uno de los siguientes métodos: `putBytes()`, `putStream()` o `putFile()`,. La invocación de uno de estos métodos retornará un `UploadTask` que te permitirá monitorizar y actuar sobre la subida de los datos.

9. Añade el permiso para acceder al almacenamiento en el manifiesto:

```
<uses-permission
            android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

10. Inserta al final del método `onCreate` de la clase `ActividadPrincipal`:

```
String[] PERMISOS = {android.Manifest.permission.WRITE_EXTERNAL_STORAGE};
ActivityCompat.requestPermissions(this, PERMISOS, 1);
```

11. Añade al final de la clase `ActividadPrincipal`:

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                String permissions[], int[] grantResults) {
   switch (requestCode) {
      case 1: {
         if (!(grantResults.length > 0 &&
                  grantResults[0] == PackageManager.PERMISSION_GRANTED)) {
          Toast.makeText(ActividadPrincipal.this,
          "Has denegado algún permiso de la aplicación.",
          Toast.LENGTH_SHORT).show();
         }
         return;
      }
   }
}
```

12. Ejecuta la aplicación.

Entra en los detalles del evento “Carnaval”. En el menú elige “Subir Imagen”, “Subir Stream” o “Subir Fichero”. Elige un fichero del dispositivo, si es necesario. Los datos de la imagen empezarán a subirse a Firebase Storage, pero no hay nada que lo indique.

Accede a la consola de Firebase al apartado STORAGE. Comprueba que hay un nuevo fichero llamado “carnaval”. Si pinchas sobre él, en la vista previa se verá la imagen del `ImageView` o el fichero elegido.

En resumen, el funcionamiento de la aplicación hasta el momento es el siguiente:

- `putBytes`: Seleccionamos uno de los eventos y presionamos el menú **SUBIR IMAGEN** cuando nos encontremos en sus detalles. Comprobamos que se ha subido la imagen al almacenamiento de Firebase. Revisamos que en la base de datos el campo `Imagen` ha cambiado el url apuntando a Firebase Storage. Almacenamos el contenido de un objeto `ImageView` en memoria en un `array` de `bytes`. Declaramos una referencia al archivo en Firebase con `child`, posteriormente subimos el contenido con `putBytes`.
- `putStream`: Seleccionamos un evento y cuando nos encontremos en la pantalla de los detalles del mismo, elegimos la opción **Subir stream**. Se abrirá una nueva pantalla en la que podremos seleccionar un archivo de nuestro dispositivo. Cuando lo hayamos seleccionado y la aplicación muestre el mensaje que se ha subido correctamente, comprobamos que se encuentra en la consola de Firebase.

- `putFile`: Comprueba lo mismo que en el ejercicio anterior. Es decir, que puedes subir un fichero del dispositivo a Firebase Storage, pero esta vez mediante el menú **SUBIR FICHERO**.

La ejecución de los métodos anteriores nos proporcionará una tarea `UploadTask`. En `UploadTask` capturaremos los eventos `onFailureListener`, `onSuccesslistener`, `onProgressListener` y `onPauseListener`, donde sobrescribiremos los métodos `onFailure`, `onSuccess`, `onProgress` y `onPaused`. Podremos mostrar los mensajes o ejecutaremos acciones en función del evento capturado. Los eventos de la tarea `UploadTask` proporcionan un forma simple y potente de controlar eventos de subida. Puedes administrar las subidas: iniciarlas, pausarlas, reanudarlas y cancelarlas. Para ello usa los métodos: `pause()`, `resume()` y `cancel()`. El evento de pausa genera un cambio en el estado a pausa, y el método de reanudación cambia el estado en progreso. La cancelación de una subida provoca un fallo y genera un error e indica que se ha cancelado.

Una tarea `UploadTask` proporciona un objeto `UploadTask.TaskSnapshot`. Es una instantánea del objeto subido en el instante que se invoca. Tiene las siguientes propiedades:

Propiedad	Descripción
`getDownloadUrl`	Obtiene el URL para descargar un archivo Es un URL público indescifrable que se puede compartir. Su valor se obtiene cuando finaliza una subida Tipo: `String`
`getError`	Indica la causa del fallo Tipo: `Exception`
`getBytesTransferred`	Cantidad de bytes transferidos Tipo: `Long`
`getTotalByteCount`	Cantidad de bytes que se prevé subir Tipo: `Long`
`getUploadSessionUri`	URI para continuar una subida con `putFile` Tipo: `String`
`getMetadata`	Consulta los metadatos de un archivo Tipo: `StorageMetadata`
`getTask`	Tarea que creó esta instantánea Usa esta tarea para cancelar, pausar o reanudar la subida Tipo: `FirebaseStorage.UploadTask`
`getStorage`	`StorageReference` usada para crear `UploadTask` Tipo: `StorageReference`

Vamos a implementar avisos al usuario, refrescar los datos en la pantalla … con los métodos de una tarea `UploadTask` proporcionada al subir datos a Firebase Storage.

Ejercicio: Eventos UploadTask en Firebase Storage

1. Declara las siguientes variables en la clase `EventoDetalles`:

```
private ProgressDialog progresoSubida;
Boolean subiendoDatos =false;
```

Con `ProgressDialog` mostraremos al usuario un cuadro de diálogo con la información de lo que está ocurriendo en la aplicación. `subiendoDatos` es una variable auxiliar que utilizaremos para saber si estamos subiendo datos a Firebase Storage.

2. Copia al principio del método `subirAFirebaseStorage`, de la clase `EventoDetalles`, la implementación de un `ProgressDialog` que informará al usuario sobre lo que está ocurriendo con la subida de datos a Firebase Storage:

```
final ProgressDialog progresoSubida = new
ProgressDialog(EventoDetalles.this);
progresoSubida.setTitle("Subiendo...");
progresoSubida.setMessage("Espere...");
progresoSubida.setCancelable(true);
progresoSubida.setCanceledOnTouchOutside(false);
```

3. Inserta el siguiente código delante de la línea `} catch (IOException e) {` que encontrarás al final del método `subirAFirebaseStorage`, en la clase `EventoDetalles`:

```
uploadTask
   .addOnFailureListener(new OnFailureListener() {
      @Override
      public void onFailure(@NonNull Exception exception) {
         //UploadTask error
      }
   })
   .addOnSuccessListener(
                       new OnSuccessListener<UploadTask.TaskSnapshot>() {
      @Override
      public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
         //UploadTask exito
      }
   })
   .addOnProgressListener(
                      new OnProgressListener<UploadTask.TaskSnapshot>() {
      @Override
```

```
      public void onProgress(UploadTask.TaskSnapshot taskSnapshot) {
         //UploadTask progreso
      }
   })
   .addOnPausedListener(new OnPausedListener<UploadTask.TaskSnapshot>() {
      @Override
      public void onPaused(UploadTask.TaskSnapshot taskSnapshot) {
         //UploadTask pausa
      }
   });
```

Acabamos de implementar los eventos de la tarea `UploadTask` y los métodos de cada uno de los eventos. Vamos a ir otorgando funcionalidad. Primero vamos a mostrar un diálogo con el porcentaje de datos subidos.

4. Sustituye la línea `//UploadTask progreso`, en el método `subirAFirebaseStorage`, por el siguiente código:

```
if (!subiendoDatos) {
   progresoSubida.show();
   subiendoDatos=true;
} else {
   if (taskSnapshot.getTotalByteCount()>0)
   progresoSubida.setMessage("Espere... "
                  +String.valueOf(100*taskSnapshot.getBytesTransferred()
                  /taskSnapshot.getTotalByteCount())+"%");
}
```

La primera vez que se llama al evento `OnProgressListener` vamos a mostrar un cuadro de diálogo para informar al usuario. Si ya se está mostrando el cuadro de diálogo, mostramos el porcentaje de datos subidos. Para ello nos valemos de `getTotalByteCount()` (previsión de bytes totales que se van a subir) y de `getBytesTransferred()` (cantidad de bytes transferidos).

5. Sustituye la línea `//UploadTask exito`, en el método `subirAFirebaseStorage`, por el siguiente código:

```
   Map<String, Object> datos = new HashMap<>();
datos.put("imagen",taskSnapshot.getDownloadUrl().toString());
FirebaseFirestore.getInstance().collection("eventos")
                        .document(evento).set(datos, SetOptions.merge());
new DownloadImageTask((ImageView) imgImagen)
                     .execute(taskSnapshot.getDownloadUrl().toString());
progresoSubida.dismiss();
subiendoDatos=false;
mostrarDialogo(getApplicationContext(),
                                        "Imagen subida correctamente.");
```

Cuando la subida de datos a Firebase Storage ha tenido éxito, informamos al usuario mediante un cuadro de diálogo. Guardamos, en la base de datos Firestore de Firebase, el URL al fichero que hemos subido. Conseguimos el URL y otros datos al preguntar las propiedades a `TaskSnapshot`.

`TaskSnapshot` es una instantánea de la tarea `UploadTask` en el momento en que se le pregunta. Por ejemplo, una vez subido el fichero, podremos obtener el URL del fichero subido a Firebase Storage con `taskSnapshot.getDownloadUrl()`.

6. Sustituye la línea `//UploadTask error`, en el método `subirAFirebaseStorage`, por el siguiente código:

```
subiendoDatos=false;
mostrarDialogo(getApplicationContext(), "Ha ocurrido un error al subir la
                            imagen o el usuario ha cancelado la subida.");
```

Cuando se produce un error se muestra un mensaje al usuario.

7. Sustituye la línea `//UploadTask pausa`, en el método `subirAFirebaseStorage`, por el siguiente código:

```
subiendoDatos=false;
mostrarDialogo(getApplicationContext(), "La subida ha sido pausada.");
```

Cuando se pausa la subida, lo indicas al usuario.

8. Ejecuta la aplicación.

Sube la imagen de un evento a Firebase Storage. Observa que se muestra un cuadro de diálodo con el progreso y. al finalizar la subida, aparece un diálogo indicándolo.

9. Inserta el siguiente código debajo de `progresoSubida.setCanceledOnTouchOutside(false);`, en el método `subirAFirebaseStorage`:

```
progresoSubida.setButton(DialogInterface.BUTTON_NEGATIVE, "Cancelar", new
DialogInterface.OnClickListener() {
   @Override
   public void onClick(DialogInterface dialog, int which) {
      uploadTask.cancel();
   }
});
```

Hemos añadido un botón al diálogo de progreso de la subida para permitir al usuario cancelar la subida. Cancelamos la subida llamando al método `cancel()` de la tarea `UploadTask`. Cuando un usuario cancela una subida, se activa el evento `OnFailureListener()`, la cancelación se considera un error.

10. Ejecuta la aplicación.

Comprueba que puedes cancelar una subida de información a Firebase Storage pulsando el botón **Cancelar** del cuadro de diálogo.

Vuelve a subir una nueva imagen, cuando empiece a subir pulsa el botón **Volver** hasta ir a la pantalla inicial y espera. Comprueba que muestra el diálogo informando que se ha subido la imagen correctamente.

En Android, a veces, el ciclo de vida de una actividad puede ocasionar problemas inesperados. Los escuchadores vinculados a una tarea de almacenamiento de Firebase continúan ejecutándose, en segundo plano, incluso después de un cambio en el ciclo de vida de una actividad. Si una actividad es destruida y recreada (por ejemplo, al rotar la pantalla), mientras una tarea se está ejecutando, podríamos obtener un `NullPointerException` cuando la tarea se complete. Para evitarlo, debemos guardar el `StorageReference` como un `String` en el `Bundle` referenciado en el método `onSaveInstanceState(Bundle)` y luego recuperarlo y agregar escuchadores a cada `FileDownloadTask` o `FileUploadTask` asociada a esa `StorageReference`. Cuando la actividad se reinicie, usamos el método `getActiveUploadTask` para obtener las tareas que se están ejecutando o se han completado recientemente.

Ejercicio: Reanudar tareas de Firebase Storage

1. Inserta el siguiente código al final de la clase `EventoDetalles`:

```
@Override
protected void onSaveInstanceState(Bundle outState) {
   super.onSaveInstanceState(outState);
   if (imagenRef != null) {
      outState.putString("EXTRA_STORAGE_REFERENCE_KEY",
                                                  imagenRef.toString());
   }
}

@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
   super.onRestoreInstanceState(savedInstanceState);
   final String stringRef = savedInstanceState
                              .getString("EXTRA_STORAGE_REFERENCE_KEY");
   if (stringRef == null) {
      return;
   }
   imagenRef = FirebaseStorage.getInstance()
                                        .getReferenceFromUrl(stringRef);
   List<UploadTask> tasks = imagenRef.getActiveUploadTasks();
   for( UploadTask task : tasks ) {
      task
               .addOnFailureListener(new OnFailureListener() {
                   @Override
                   public void onFailure(@NonNull Exception exception) {
                       upload_error(exception);
                   }
                })
```

```
                .addOnSuccessListener(new
                          OnSuccessListener<UploadTask.TaskSnapshot>() {
                    @Override
                    public void onSuccess(UploadTask.TaskSnapshot
                                                          taskSnapshot) {
                        upload_exito(taskSnapshot);
                    }
                })
                .addOnProgressListener(new
                          OnProgressListener<UploadTask.TaskSnapshot>() {
                    @Override
                    public void onProgress(UploadTask.TaskSnapshot
                                                          taskSnapshot) {
                        upload_progreso(taskSnapshot);
                    }
                })
                .addOnPausedListener(new
                             OnPausedListener<UploadTask.TaskSnapshot>() {
                    @Override
                    public void onPaused(UploadTask.TaskSnapshot
                                                           taskSnapshot) {
                        upload_pausa(taskSnapshot);
                    }
                });
    }
}
```

2. Añade los siguientes métodos al final de la clase EventoDetalles:

```
    private void upload_error(Exception exception){
   subiendoDatos=false;
   mostrarDialogo(getApplicationContext(),
      "Ha ocurrido un error al subir la imagen o el usuario ha
      cancelado la subida.");
}

private void upload_exito(UploadTask.TaskSnapshot taskSnapshot){
   Map<String, Object> datos = new HashMap<>();
   datos.put("imagen",taskSnapshot.getDownloadUrl().toString());
   FirebaseFirestore.getInstance().collection("eventos")
                                          .document(evento).set(datos);
   new DownloadImageTask((ImageView) imgImagen)
                    .execute(taskSnapshot.getDownloadUrl().toString());
   progresoSubida.dismiss();
   subiendoDatos=false;
   mostrarDialogo(getApplicationContext(),
                                          "Imagen subida correctamente.");
   }

   private void upload_progreso(UploadTask.TaskSnapshot taskSnapshot){
```

```
      if (!subiendoDatos) {
         progresoSubida = new ProgressDialog(EventoDetalles.this);
         progresoSubida.setTitle("Subiendo...");
         progresoSubida.setMessage("Espere...");
         progresoSubida.setCancelable(true);
         progresoSubida.setCanceledOnTouchOutside(false);
         progresoSubida.setButton(DialogInterface.BUTTON_NEGATIVE,
                     "Cancelar", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
               uploadTask.cancel();
            }
         });
         progresoSubida.show();
         subiendoDatos=true;
      } else {
         if (taskSnapshot.getTotalByteCount()>0)
            progresoSubida.setMessage("Espere... "
                  +String.valueOf(100*taskSnapshot.getBytesTransferred()
                  /taskSnapshot.getTotalByteCount())+"%");
      }
   }
private void upload_pausa(UploadTask.TaskSnapshot taskSnapshot){
   subiendoDatos=false;
   mostrarDialogo(getApplicationContext(), "La subida ha sido pausada.");
}
```

3. Ejecuta la aplicación y gira la pantalla cuando estés subiendo un fichero.

Si por cualquier motivo las subidas en curso se suspenden, posteriormente, podremos reanudar la subida desde el punto donde se interrumpieron. Esto puede ahorrar tiempo y ancho de banda.

Para realizarlo, comenzamos a subir el archivo con el método `putFile`. En la `StorageTask` resultante, llamamos al método `getUploadSessionUri` y guardamos el valor resultante en un almacenamiento persistente (como `SharedPreferences`).

```
uploadTask = mStorageRef.putFile(localFile);
sessionUri = uploadTask.getUploadSessionUri();
//Guarda sessionUri en un almacenamiento persistente.
```

Para reiniciar el proceso de una subida interrumpida, volvemos a llamar `putFile`. Pero le pasamos el URI que guardamos al pausar la tarea como el último argumento en la llamada.

```
uploadTask = mStorageRef.putFile(localFile,
                     new StorageMetadata.Builder().build(), sessionUri);
```

Las sesiones duran 1 semana. Si intentamos reanudar una sesión que ha caducado o tiene un error, recibiremos un `callback` de fallo. Es la responsabilidad del programador garantizar que el archivo no ha cambiado entre subidas.

Existen múltiples motivos por los que se pueden producir errores durante la subida. Por ejemplo: que el archivo local no exista, que el usuario no tenga permisos…

3.2.1.3. Descargar archivos

Firebase Storage permite descargar archivos de forma rápida y fácil. Para descargar un archivo, necesitamos la referencia Firebase Storage al archivo.

Podemos descargar archivos de Firebase Storage llamando a `getBytes()` o `getFile()`. También podemos obtener un URL de descarga con `getDownloadUrl()` para realizar la descarga desde otra plataforma.

Descargar el archivo en un `byte[]` con el método `getBytes()` es la forma más fácil de descargar un archivo, pero el contenido descargado se carga en memoria. Así, cuanto más grande sea el archivo, menor será la memoria disponible en la aplicación, lo que puede provocar que falle. Para proteger la aplicación contra los problemas de memoria, configuramos el tamaño máximo de memoria para la aplicación o usamos otro método de descarga.

El método `getFile()` descarga un archivo directamente en el dispositivo local. Usamos este método para tener acceso al archivo sin conexión o para compartirlo con otra aplicación. `getFile()` devuelve una `DownloadTask` que podemos usar para administrar la descarga.

Si la aplicación obtiene sus recursos mediante URL o necesitamos un URL para compartir, podemos obtenerlo llamando al método `getDownloadUrl()`.

Al igual que en la subida de archivos, en la descarga también obtenemos una tarea para conseguir información acerca del estado de la descarga. Al invocar uno de los métodos de descarga, obtenemos una tarea `DownloadTask`. Podemos actuar ante los eventos `OnSuccessListener`, `OnFailureListener`, `OnProgressListener` y `OnPauseListener`, en los que sobrescribiremos los métodos `onFailure`, `onSuccess`, `onProgress` y `onPaused`. También podremos administrar las descargas: iniciarlas, pausarlas, reanudarlas y cancelarlas. Para ello, usamos los métodos: `pause()`, `resume()` y `cancel()`.

Vamos a ver cómo descargar un archivo desde Firebase Storage. Utilizaremos el método `getFile()`, ya que lo vamos a guardar en el almacenamiento del dispositivo. Al entrar en los detalles de un evento, tendremos una nueva opción de menú para descargar la imagen del evento a un fichero en el dispositivo.

Ejercicio: Descargar archivos de Firebase Storage

1. Añade un nuevo elemento en el menú `menu_detalles`:

```
<item
   android:id="@+id/action_getFile"
   android:orderInCategory="102"
```

```
   android:title="Descargar                                    Fichero"
   app:showAsAction="never" />
```

2. Inicializa la nueva opción de menú que has creado añadiendo el siguiente código, en el método `onOptionsItemSelected` de la clase `EventoDetalles`:

```
case R.id.action_getFile:
    descargarDeFirebaseStorage(evento);
    break;
```

3. Inserta al final de la clase `EventoDetalles`:

```
public void descargarDeFirebaseStorage(String fichero) {
   StorageReference referenciaFichero =
                                    getStorageReference().child(fichero);
   File rootPath = new File(Environment.getExternalStorageDirectory(),
                                                              "Eventos");
   if(!rootPath.exists()) {
      rootPath.mkdirs();
   }
   final File localFile = new File(rootPath,evento+".jpg");
   referenciaFichero.getFile(localFile).addOnSuccessListener(new
                    OnSuccessListener<FileDownloadTask.TaskSnapshot>() {
      @Override
      public void onSuccess(FileDownloadTask.TaskSnapshot taskSnapshot) {
         mostrarDialogo(getApplicationContext(),
                  "Fichero descargado con éxito: "+localFile.toString());
      }
   }).addOnFailureListener(new OnFailureListener() {
      @Override
      public void onFailure(@NonNull Exception exception) {
         mostrarDialogo(getApplicationContext(),
                                      "Error al descargar el fichero.");
      }
   });
}
```

4. Ejecuta la aplicación.

Comprueba que cuando ejecutas el menú **DESCARGAR FICHERO**, en los detalles de un evento, se descarga el fichero en el almacenamiento del dispositivo.

Al igual que las subidas de archivos, las descargas continúan ejecutándose en segundo plano incluso después de que cambie el ciclo de vida de una actividad; esto, al igual que en las descargas, puede presentar problemas. Debes controlarlo de la misma forma que hemos hecho con las subidas a Firebase Storage. Lo único que cambia en las subidas respecto de las descargas es que tenemos que controlar `DownloadTask` y no `UploadTask`.

Existen diferentes motivos por los que pueden producirse errores en la descarga. Los más habituales son porque el archivo no exista o el usuario no tenga permisos.

3.2.1.4. Metadatos de archivos

Después de cargar un archivo en Firebase Storage, también es posible obtener y actualizar los metadatos del archivo. Podemos almacenar pares clave-valor personalizados como metadatos adicionales.

Los metadatos contienen propiedades comunes como `name`, `size` y `contentType`, además de otras menos comunes como `contentDisposition` y `timeCreated`. Los metadatos se recuperan de una referencia de Firebase Storage usando el método `getMetadata()`.

```
referenciaImagen.getMetadata()
   .addOnSuccessListener( new OnSuccessListener<StorageMetadata>() {
      @Override
      public void onSuccess(StorageMetadata storageMetadata) {
         // Metadatos obtenidos con éxito.
         String nombre = storageMetadata.getName();
         String evento = storageMetadata.getCustomMetadata("evento");
      }
   })
   .addOnFailureListener(new OnFailureListener() {
      @Override
      public void onFailure(@NonNull Exception exception) {
          // Ha ocurrido un error al descargar los metadatos
      }
   });
```

En el ejemplo anterior utilizamos el método `getMetadata()` para obtener metadatos. Podemos asociar escuchadores a la consulta. Con `storageMetadata.getName()` obtenemos el nombre del fichero. Con `storageMetadata.getCustomMetadata("evento")` preguntamos por el valor de un metadato personalizado.

Después de que finalice la carga de un archivo, podemos actualizar sus metadatos usando el método `updateMetadata()`. Solo se actualizan las propiedades especificadas; todas las demás quedan sin modificar.

Se pueden especificar metadatos personalizados con un `Map<String, String>`. También podemos utilizar `setCustomMetadata` con pares clave-valor para establecer los metadatos personalizados.

```
StorageMetadata metadata = new StorageMetadata.Builder()
   .setContentType("image/jpg")
   .setCustomMetadata("autor", "Pere Perez")
   .build();
forestRef.updateMetadata(metadata)
   .addOnSuccessListener(new OnSuccessListener<StorageMetadata>() {…}
   .addOnFailureListener(new OnFailureListener() {...}
```


En el ejemplo anterior, hemos utilizado `setCustomMetadata` para guardar metadatos personalizados y `setContentType` para actualizar la propiedad `contentType` del archivo.

Aunque podemos almacenar datos específicos en cada archivo en forma de metadatos personalizados, lo recomendable es usar una base de datos para almacenar y sincronizar este tipo de datos.

La lista completa de las propiedades de los metadatos la puedes consultar en:

https://firebase.google.com/docs/storage/android/file-metadata?hl=es#propiedades_de_los_metadatos_de_archivos

3.2.1.5. Eliminar archivos

Para borrar un archivo, primero creamos una referencia al archivo. Luego llamamos al método `delete()` de esa referencia. Por ejemplo:

```
StorageReference storageRef = referenciaAlmacenamiento
                                        .getReferenceFromUrl("gs://...");
StorageReference referenciaImagen = referenciaAlmacenamiento
                                                .child("images/flor.jpg");
referenciaImagen.delete()
   .addOnSuccessListener(new OnSuccessListener() {
      @Override
      public void onSuccess(Void aVoid) {
         // Fichero borrado correctmente
      }})
   .addOnFailureListener(new OnFailureListener() {
      @Override
      public void onFailure(@NonNull Exception exception) {
         //Error al subir el fichero
      }});
```

En el ejemplo anterior vemos cómo eliminar un archivo de Firebase Storage. Aplicamos el método `delete` a la referencia del archivo que queremos borrar (`referenciaImagen.delete()`). Al igual que en las subidas, descargas y consulta y actualización de metadatos, podemos establecer una serie de escuchadores. En el ejemplo, hemos implementado los escuchadores `OnSuccessListener` y `OnFailureListener`. Eliminar un archivo es una acción permanente. Creamos una copia de seguridad o habilitamos versiones en Google Cloud Storage si queremos restaurar archivos borrados.

Práctica: Eliminar archivos en Firebase Storage

Añade un nuevo elemento en el menú de los detalles de un evento llamado “Eliminar”. Al seleccionarlo, se le indicará al usuario mediante un cuadro de diálogo si desea eliminar la imagen del evento. Si la respuesta es positiva, eliminarás la imagen de Firebase Storage y eliminarás también en Firebase

Database la referencia a dicha imagen dejando el campo Imagen en blanco. No elimines el registro del evento en la base de datos.

Preguntas de repaso: Firebase Storage

3.2.2. Almacenamiento en Google Drive

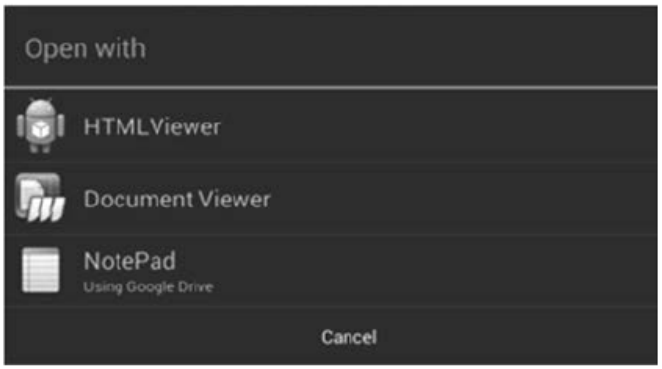
Hemos visto los servicios ofrecidos por Firebase, pero un desarrollador puede interacturar con más servicios ofrecidos por Google Play Services. Como, por ejemplo: Drive, Youtube, Calendar, Gmail… Para utilizarlos, usamos Google Cloud Platform. Para ello, accedemos a la consola de Google API (https://**console**.developers.**google**.com/?hl=ES) con una cuenta de Google. Los proyectos creados en Firebase son visibles en Google Cloud Platform. Así que podemos añadir nuevos servicios a nuestro proyecto. Seguidamente veremos cómo manejar la plataforma y cómo utilizar un servicio en una aplicación Android. Para ello, ampliaremos el proyecto "Eventos" con el servicio Google Drive.

Google Drive es un contenedor de archivos y carpetas ubicado en la nube. En él podemos guardar y compartir todo lo que queramos. Con Google Drive, podemos acceder a archivos y carpetas desde un navegador web o desde cualquier dispositivo en el que hayamos instalado una aplicación que utilice la API de Google Drive. Pase lo que pase con nuestros dispositivos, siempre tendremos nuestros archivos guardados de forma segura en Google Drive.

Google Drive ofrece muchas maneras de ver, buscar y ordenar los archivos. Incluye opciones de búsqueda potentes (incluso la capacidad de buscar texto en imágenes) para que podamos encontrar rápidamente lo que buscamos.

Si usamos los servicios de Google Drive en Google Play Services, podemos integrar aplicaciones móviles con Google Drive en Android. Vamos a distinguir dos tipos de aplicaciones:

1. Aplicaciones que abren archivos de Google Drive. Cuando los usuarios de Android eligen abrir archivos alojados en Google Drive pueden seleccionar nuestra aplicación desde una lista y abrir el fichero con ella, siempre que el tipo MIME del fichero sea soportado por nuestra aplicación.



2. Aplicaciones que leen y/o modifican el contenido de una cuenta en Google Drive. Por ejemplo, hacer una fotografía con la cámara o seleccionarla de la galería del dispositivo y subirla a una carpeta de Google Drive. Otra aplicación podría listar las fotografías subidas.

3.2.3. Google Drive API

La API de Drive para Android es una API nativa que conforma las mejores prácticas y convenciones de codificación para Android de forma estándar. Esto significa que se simplifican algunas tareas asociadas al uso del servicio Drive en dispositivos móviles. La API maneja automáticamente tareas complejas, tales como el acceso fuera de línea y la sincronización de archivos. Te permite leer y escribir archivos como si Google Drive fuera un sistema de archivos local.

La API de Google Drive reduce significativamente el tamaño de la aplicación. Esto es debido a que forma parte de la librería cliente de Google Play Services. No necesitaremos librerías Java para acceder al servicio.

Para utilizar Google Drive en Android, disponemos de dos opciones: utilizar llamadas `REST` o bien conectar con los servicios de Google Play Services mediante un cliente `GoogleApiClient`. Vamos a ver la primera opción: utilizar llamadas `REST`.

Para realizar una aplicación que utilice Google Drive vamos a tener que utilizar su API, que no son más que llamadas HTTP por el método `get` o `post`. Puedes consultar todas las llamadas de Google Drive API REST v3 en https://developers.google.com/drive/v3/reference/ .

3.2.4. Crear una aplicación Android para Google Drive

Vamos a ampliar la aplicación “Eventos” para que permita utilizar Google Drive. Hemos comentado que podemos crear dos tipos de aplicaciones. Al primer tipo pertenecen aquellas aplicaciones que cuando se abre un archivo en Google Drive aparecen en la lista de aplicaciones que pueden abrir ese tipo de archivo. También has de saber que Google Drive API incluye una carpeta especial oculta que puede usar una aplicación para almacenar datos específicos y que no es accesible al usuario. Puedes consultarlo en https://developers.google.com/drive/android/appfolder.

Vamos a centrarnos en el tipo de aplicaciones que leen y/o modifican ficheros y/o carpetas alojadas en Google Drive. La aplicación subirá fotografías hechas con la cámara o seleccionadas de la galería del dispositivo a una carpeta de Google Drive. Además, listará los nombres de las fotografías subidas.

Para crear una aplicación que utilice los servicios de Google Drive, necesitarás: una cuenta en Google con Drive habilitado, la librería Google Play Services y Google Plugin para Android Studio.

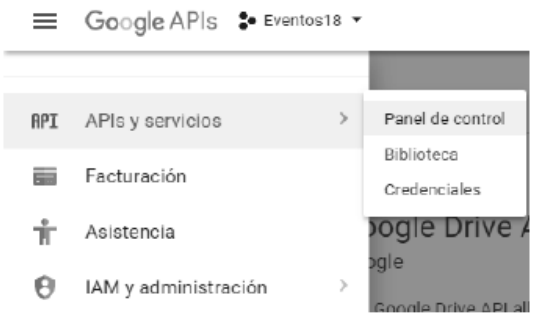
3.2.4.1. Habilitar el servicio Google Drive API

Es necesario crear un proyecto en la consola de Google API que tenga habilitado el servicio Drive API. Además, tenemos que configurar el acceso mediante OAuth 2.0.

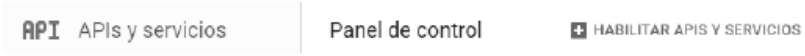
Ejercicio: Habilitar el servicio Drive en la consola de Google API

1. Accede a consola de Google API (http://code.google.com/apis/console) con tu cuenta de Google.
2. Selecciona el proyecto ”Eventos”.

   Los proyectos de la consola de Firebase se pueden ver en la consola de Google API. Se pueden añadir funcionalidades de Google API a un proyecto Firebase y al revés. Por esto es por lo que no ha sido necesario dar de alta el proyecto “Eventos”, ya lo tenemos dado de alta.
3. Selecciona el menú **APIS Y SERVICIOS** y luego **Panel de control**.

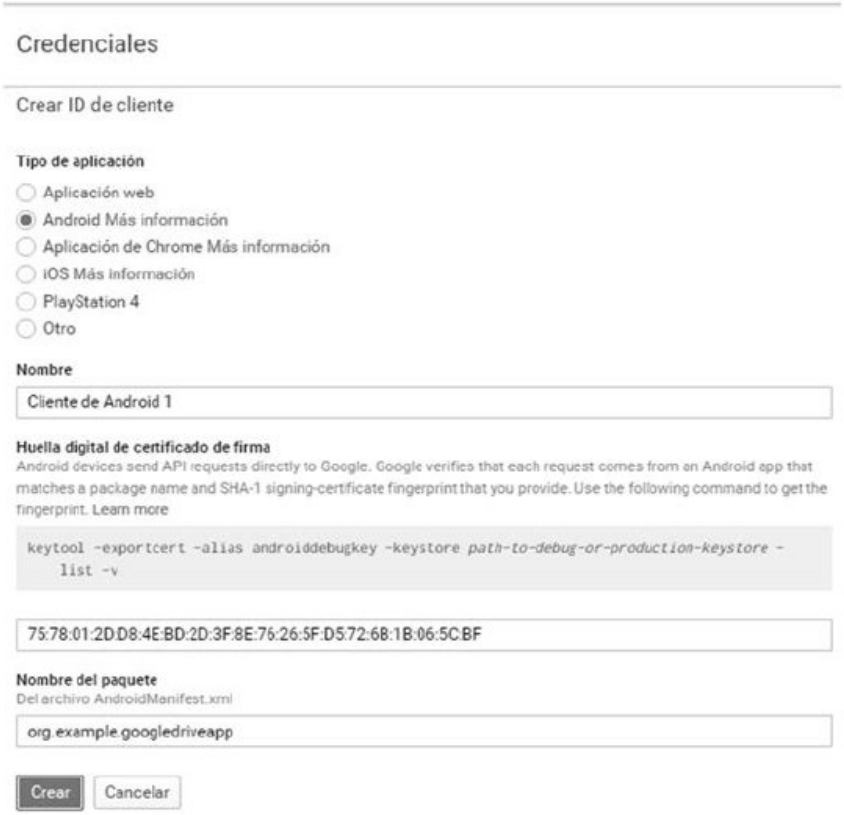


4. Pulsa en el botón **Habilitar APIS y servicios**.



5. En la nueva pantalla, busca “Google Drive API”, selecciona el ítem Google Drive API que obtendrás con la búsqueda.
6. Pulsa el botón **Habilitar**.
7. Selecciona el menú **CREDENCIALES**.
8. Haz clic en el botón **Crear credenciales** y elige en el desplegable ”**ID de cliente OAuth**”.
9. Configura la pantalla de consentimiento. Es la que verá el usuario cuando se le pida autorización para que tu aplicación utilice su Google Drive. Solo es obligatorio que introduzcas tu e-mail y el nombre de la aplicación. Pulsa en **Guardar** cuando termines de introducir los datos.
10. En la nueva ventana, selecciona Android en **Tipo de aplicación**.
11. Introduce “Eventos” en **Nombre**.

12. Escribe "org.example.eventos" en **Nombre del paquete**". Es el nombre del paquete en Android Studio.
13. Introduce la huella digital SHA-1.
14. Haz clic en **Crear**.



3.2.4.2. Autorizar el acceso a Google Drive

Para poder leer y/o modificar los archivos y/o carpetas de una cuenta en Google Drive, tenemos que pedir al usuario con Google Drive habilitado qué cuenta desea utilizar. Los archivos de dicha cuenta serán a los que podamos acceder en Google Drive desde la aplicación.

Vamos a añadir los servicios de Drive API a la aplicación “Eventos” que estamos desarrollando.

Ejercicio: Preparar la aplicación para usar la API de Google Drive

1. Abre el fichero build.gradle [Module:app] y añade las siguientes dependencias:

```
implementation 'com.google.http-client:google-http-client-gson:1.17.0-rc'
implementation 'com.google.api-client:google-api-client-xml:1.17.0-rc'
implementation 'com.google.api-client:google-api-client-android:1.17.0-rc'
implementation 'com.google.apis:google-api-services-drive:v3-rev52-1.22.0'
implementation 'com.google.android.gms:play-services-auth:11.8.0'
```

2. Inserta al final del fichero build.graddle [Module:app] las siguientes excepciones:

```
configurations {
   all*.exclude group: 'xpp3', module: 'xpp3'
   compile.exclude group: "org.apache.httpcomponents",
                                                  module: "httpclient"
}
```

Si realizas una aplicación que no utilice Firebase, las excepciones no son necesarias.

3. Añade el siguiente permiso en Android manifest:

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

Nuestro proyecto ya tiene el permiso `INTERNET`, necesario para usar Drive API. Recuerda incluirlo en una aplicación nueva. También necesitamos el permiso `GET_ACCOUNTS` para poder acceder a las cuentas de usuario y que se nos indique qué cuenta se va a utilizar para acceder a Google Drive.

4. En el método `onCreate` de la clase `ActividadPrincipal` sustituye la declaración de la variable `PERMISOS` por:

```
String[]                              PERMISOS                              =
{android.Manifest.permission.WRITE_EXTERNAL_STORAGE,
 android.Manifest.permission.GET_ACCOUNTS};
```

5. Crea un layout llamado `fotografias_drive.xml` con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <android.support.design.widget.AppBarLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content">
      <android.support.v7.widget.Toolbar
         android:id="@+id/toolbar"
         android:layout_width="match_parent"
         android:layout_height="?attr/actionBarSize"
         android:background="?attr/colorPrimary" />
   </android.support.design.widget.AppBarLayout>
   <TextView
      android:id="@+id/display"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
</LinearLayout>
```

6. Crea un nuevo menú en recursos llamado `menu_drive` con el siguiente código:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
```

```
   tools:context="org.example.eventosx.FotografiasDrive">
   <item
      android:id="@+id/action_camara"
      android:orderInCategory="100"
      android:title="Hacer foto"
      app:showAsAction="never" />
   <item
      android:id="@+id/action_galeria"
      android:orderInCategory="101"
      android:title="Elegir imagen"
      app:showAsAction="never" />
</menu>
```

7. Crea una nueva clase llamada `FotografiasDrive` con el siguiente código:

```
public class FotografiasDrive extends AppCompatActivity {
   public TextView mDisplay;
   String evento;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.fotografias_drive);
      Bundle extras = getIntent().getExtras();
      evento =extras.getString("evento");
      Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
      setSupportActionBar(toolbar);
   }
   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.menu_drive, menu);
      return true;
   }
   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      View vista = (View) findViewById(android.R.id.content);
      int id = item.getItemId();
      switch (id) {
         case R.id.action_camara:
            break;
         case R.id.action_galeria:
            break;
      }
      return super.onOptionsItemSelected(item);
   }
}
```

8. Declara la actividad en el manifiesto con:

```
<activity
    android:name=".FotografiasDrive"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.NoActionBar">
</activity>
```

9. Añade un nuevo elemento en el menú `menu_detalles`:

```
<item
   android:id="@+id/action_fotografiasDrive"
   android:orderInCategory="103"
   android:title="Fotografias Drive"
   app:showAsAction="never" />
```

10. Declara la siguiente constante en la clase `EventoDetalles`:

```
final int SOLICITUD_FOTOGRAFIAS_DRIVE = 102;
```

11. Añade la siguiente opción en el switch del método `onOptionsItemSelected` en la clase `EventoDetalles`:

```
case R.id.action_fotografiasDrive:
   Intent intent = new Intent(getBaseContext(), FotografiasDrive.class);
   intent.putExtra("evento", evento);
   startActivity(intent);
   break;
```

12. Ejecuta la aplicación.

Comprueba que si seleccionas un evento de la lista y accedes a sus detalles, aparece un nuevo menú llamado **FOTOGRAFÍAS DRIVE**. Esta es la nueva actividad que acabamos de crear y es donde el usuario podrá subir las fotografías que desee sobre el evento a su unidad de Google Drive. Para ello utilizará el menú de la actividad, donde podrá elegir entre hacer una fotografía o seleccionar una imagen desde la galería del dispositivo, y subirla a Google Drive. La actividad recibe mediante un extra el identificador del evento seleccionado.

Hemos añadido la dependencia a los servicios de Google Drive incluidos en Google Play Services, añadiendo la dependencia en el fichero build.gradle. Posteriormente, hemos agregado los permisos `android.permission.GET_ACCOUNTS` para poder interactuar con las cuentas Google del dispositivo, y `android.permission.INTERNET`, necesario para las solicitudes a la API de Google Drive. También hemos añadido un layout donde manejaremos los ficheros que se suban a Google Drive. Seguidamente vamos a conectar la aplicación con la cuenta del usuario en Google Drive. Cuando se muestre la actividad, pedirá una cuenta de usuario si no se ha autorizado el acceso anteriormente.

Vamos a conectar nuestra aplicación a Google Drive:

Ejercicio: Obtener acceso a Google Drive

1. Declara las siguientes variables en la clase `FotografiasDrive`:

```
static Drive servicio = null;
static GoogleAccountCredential credencial = null;
static String nombreCuenta = null;
```

```
static final int PLAY_SERVICES_RESOLUTION_REQUEST = 9000;
static final String DISPLAY_MESSAGE_ACTION =
                                    "org.example.eventos.DISPLAY_MESSAGE";
private static Handler manejador = new Handler();
private static Handler carga = new Handler();
private             static             ProgressDialog             dialogo;
private Boolean noAutoriza=false;
```

2. Inserta al final de la clase `FotografiasDrive` los siguientes métodos:

```
static void mostrarMensaje(final Context context, final String mensaje) {
   manejador.post(new Runnable() {
      public void run() {
         Toast.makeText(context, mensaje, Toast.LENGTH_SHORT).show();
      }
   });
}
static void mostrarCarga(final Context context, final String mensaje) {
   carga.post(new Runnable() {
      public void run() {
         dialogo = new ProgressDialog(context);
         dialogo.setMessage(mensaje);
         dialogo.show();
      }
   });
}
static void ocultarCarga(final Context context) {
   carga.post(new Runnable() {
      public void run() {
         dialogo.dismiss();
      }
   });
}
```

Hemos declarado las variables que nos permitirán acceder a los servicios de Google Drive con un usuario validado (`servicio` y `credencial`) y otras auxiliares. Además, hemos incluido los siguientes métodos:

- `mostrarMensaje`: nos permitirá mostrar mensajes al usuario mediante un `Toast`.
- `mostrarCarga`: indicará, mediante una ventana de diálogo, que la aplicación está realizando una operación.
- `ocultarCarga`: ocultará la ventana de diálogo, indicando al usuario que ha finalizado la operación y ya se puede utilizar la aplicación.

3. Declara las siguientes variables en la clase `FotografiasDrive`:

```
static final int SOLICITUD_SELECCION_CUENTA = 1;
static final int SOLICITUD_AUTORIZACION = 2;
static final int SOLICITUD_SELECCIONAR_FOTOGRAFIA = 3;
static final int SOLICITUD_HACER_FOTOGRAFIA = 4;
private static Uri uriFichero;
```

4. Inserta al final del método `onCreate` de la clase `FotografiasDrive`:

```
credencial = GoogleAccountCredential.usingOAuth2(this,
                                    Arrays.asList(DriveScopes.DRIVE));
SharedPreferences prefs = getSharedPreferences("Preferencias",
                                                Context.MODE_PRIVATE);
nombreCuenta = prefs.getString("nombreCuenta", null);
noAutoriza = prefs.getBoolean("noAutoriza",false);
if (!noAutoriza){
    if (nombreCuenta == null) {
        PedirCredenciales();
   } else {
      credencial.setSelectedAccountName(nombreCuenta);
      servicio = obtenerServicioDrive(credencial);
   }
}
```

5. Añade al final de la clase `FotografiasDrive`:

```
private void PedirCredenciales() {
   if (nombreCuenta == null) {
      startActivityForResult(credencial.newChooseAccountIntent(),
                                            SOLICITUD_SELECCION_CUENTA);
   }
}
@Override
protected void onActivityResult(final int requestCode,
                                final int resultCode, final Intent data) {
   switch (requestCode) {
      case SOLICITUD_SELECCION_CUENTA:
         if (resultCode == RESULT_OK && data != null
                                        && data.getExtras() != null) {
            nombreCuenta = data
                      .getStringExtra(AccountManager.KEY_ACCOUNT_NAME);
            if (nombreCuenta != null) {
               credencial.setSelectedAccountName(nombreCuenta);
               servicio = obtenerServicioDrive(credencial);
               SharedPreferences prefs = getSharedPreferences(
                                  "Preferencias", Context.MODE_PRIVATE);
               SharedPreferences.Editor editor = prefs.edit();
               editor.putString("nombreCuenta", nombreCuenta);
               editor.commit();
            }
         }
         break;
      case SOLICITUD_HACER_FOTOGRAFIA:
         break;
      case SOLICITUD_SELECCIONAR_FOTOGRAFIA:
         break;
      case SOLICITUD_AUTORIZACION:
         break;
   }
}
```

```
private Drive obtenerServicioDrive(GoogleAccountCredential credencial) {
   return new Drive.Builder(AndroidHttp.newCompatibleTransport(),
                                 new GsonFactory(), credencial).build();
}
```

6. Ejecuta la aplicación.

Accede a las fotografías de un evento. Comprueba que cuando se accede a la actividad `FotografiasDrive`, se lanza una actividad para que el usuario elija una cuenta con la que conectarse a Google Drive. Seleciona otro evento y comprueba que cuando vuelva a acceder a la actividad, no se nos pide la cuenta.

Para tener acceso a Google Drive es necesario hacerlo mediante OAuth 2.0. Es un protocolo abierto que permite la autorización segura, desde una API de modo estándar y simple, para aplicaciones de escritorio, móviles y web. Permite a un usuario del sitio A compartir su información en el sitio A (proveedor) con el sitio B (consumidor) sin compartir toda su identidad.

Lo primero que tendremos que comprobar es que Google Play Services está instalado y actualizado en el dispositivo o emulador donde vamos a ejecutar la aplicación. Esto ya lo realizamos en la clase `ActividadPrincipal`.

Desde la actividad `FotografiasDrive` lanzaremos otras actividades con el método `startActivityForResult()`. Esto implica que podemos procesar los datos devueltos por las actividades lanzadas en el método `onActivityResult()`. Llamaremos a 4 actividades a las que tendremos que asignar un código de respuesta:

- *SOLICITUD_SELECCION_CUENTA*: Se asocia a la actividad lanzada con el `intent newChooseAccountIntent` para que el usuario seleccione una cuenta de Google.
- *SOLICITUD_AUTORIZACION:* Para usar el servicio Google Drive desde una aplicación, es imprescindible que el usuario haya dado el visto bueno. Cuando nuestra aplicación acceda por primera vez a Google Drive, el usuario no habrá dado todavía esta autorización, por lo que se generará un error. Este error ha de ser capturado por la excepción `UserRecoverableAuthIOException`, donde lanzaremos una actividad para pedirle al usuario la autorización. Esta actividad tendrá asociado el código `SOLICITUD_AUTORIZACION` para poder procesar la respuesta.
- *SOLICITUD_HACER_FOTOGRAFIA*: Nuestra aplicación toma una fotografía con la cámara y la sube a Google Drive. Para realizar la fotografía lanzaremos una actividad por medio de una intención explicita. Usaremos este código para asociarlo a la actividad.
- *SOLICITUD_SELECCIONAR_FOTOGRAFIA*: Seleccionamos una fotografía de la galería y la sube a Google Drive. Al igual que para hacer una fotografía con la cámara, para seleccionar la fotografía lanzaremos una actividad por medio de una intención explicita.

En el método `onCreate` inicializamos la variable `credencial` de la clase `GoogleAccountCredential`. Indicamos que se va a usar OAuth2.0 con `usingOAuth2` para el acceso a Google Drive con

`Arrays.asList(DriveScopes.DRIVE)`. Consultamos si ya hemos pedido la cuenta; si es así, la habremos guardado en `SharedPreferences`. La leemos y activamos el acceso al servicio en Drive llamando a `obtenerServicioDrive` con el nombre de la cuenta. Si no, el usuario deberá elegir una cuenta o bien la creará. Esto lo hacemos en la función `pedirCredenciales`.

En la función `pedirCredenciales` lanzamos `newChooseAccountIntent` para solicitar una cuenta. Al seleccionar la cuenta procesamos la información en `onActivityResult`. En la opción del switch `SOLICITUD_SELECCION_CUENTA`, llamamos a `obtenerServicioDrive` mediante las credenciales de la cuenta seleccionada `credential.setSelectedAccountName(nombre_cuenta)`.

La función `obtenerServicioDrive` retorna el servicio Google Drive que utilizaremos para hacer las llamadas a la API. El servicio lo usaremos con una variable llamada `servicio` y estará configurado de la siguiente forma:

```
Drive.Builder(AndroidHttp.newCompatibleTransport(),new
GsonFactory(), credencial)
```

Con `AndroidHttp.newCompatibleTransport` indicamos que vamos a hacer llamadas HTTP. `GsonFactory` va a indicar que nos va a devolver resultados en JSON. Y con `credencial` indicamos que vamos a acceder con las credenciales de la cuenta del usuario, a la que podremos leer y/o modificar los ficheros.

Una vez generado el certificado de firma, configurado el proyecto en la consola de Google y creado el código fuente de la aplicación Android, la aplicación está lista para funcionar.

3.2.4.3. Subir ficheros a Google Drive

Uno de los objetivos principales de una aplicación que utilice Google Drive es la capacidad de subir ficheros. Vamos a añadir el código para que seamos capaces de hacer una foto y subirla a una carpeta de Google Drive. La carpeta se va a llamar EventosDrive, la cual también crearemos desde la aplicación.

Tanto para crear la carpeta como para subir el archivo, utilizaremos la llamada `Files().create`. Pero los parámetros serán diferentes en cada caso. Para crear la carpeta le pasaremos un parámetro con los atributos del fichero. Mientras que para la fotografía, le pasaremos dos parámetros: atributos del archivo y contenido. Para los atributos del archivo utilizaremos una variable de tipo `com.google.api.services.drive.model.File`. Estableceremos atributos asignando valores para `name` (nombre) y `mimeType` (tipo) del fichero. Para el contenido binario del archivo utilizaremos una variable de tipo `FileContent`:

```
Files().insert(File fichero, FileContent contenido )
```

Las llamadas a la API de Google Drive funcionan como las llamadas HTTP. Para que te hagas una idea, es como si rellenases un formulario HTML. Primero rellenas los campos, pero no se envía nada hasta que no pulsas el botón **Enviar** en el formulario. Por eso, acuérdate de poner `execute()` al final de la llamada `Files().create(body,contenido).execute()`.

Drive trabaja con los objetos de tipos MIME. Aunque son archivos y capetas, no entienden de extensiones. Así nos referiremos una imagen .jpg como image/jpeg, a una capeta, como application/vnd.google-apps.folder. Puedes consultar algunos tipos MIME en http://www.htmlquick.com/es/reference/mime-types.html.

Cuando vayamos a realizar una acción con Drive es obligatorio que se ejecute en un hilo. Si lo pensamos, esto tiene sentido: vamos a hacer una llamada a Internet que va a depender de la calidad de conexión en ese momento y de la cantidad de datos. Si la calidad de la conexión es mala o el archivo es muy grande, pueden provocar el típico mensaje de que la aplicación no responde con la lógica frustración del usuario. Así, en este ejemplo, si la cámara del dispositivo es de 5 mpx (y aunque la conexión sea 3G), si no la subiéramos mediante un hilo separado de la actividad principal, provocaría un mensaje de que la aplicación no responde. Si el archivo a subir o descargar es muy grande, podemos realizar la operación por partes.

Recuerda que todo lo relativo a Drive API lo puedes encontrar en https://developers.google.com/drive/v3/reference/ incluidos ejemplos en Java.

Ejercicio: Crear carpetas en Google Drive

1. Declara las variables `idCarpeta` y `idCarpetaEvento` en la clase `FotografiasDrive`:

```
private String idCarpeta="";
private String idCarpetaEvento="";
```

   Para acceder a un fichero en concreto lo hacemos mediante el identificador que tiene en Google Drive. Para subir las fotografías a la carpeta EventosDrive vamos a necesitar su identificador. `idCarpeta` contendrá el identificador de la carpeta base llamada EventosDrive e `idCarpetaEvento` contendrá el identificador de la carpeta del evento seleccionado, por ejemplo, “fallas”. Lo vamos a guardar en las preferencias y además en una variable local.

2. Añade debajo de la instrucción `noAutoriza = prefs.getBoolean("noAutoriza",false);`, en el método `onCreate` de la clase `FotografiasDrive`, el siguiente código:

```
idCarpeta = prefs.getString("idCarpeta", null);
idCarpetaEvento = prefs.getString("idCarpeta_"+evento, null);
```

3. Añade debajo de la instrucción `servicio = obtenerServicioDrive(credencial);`, en el método `onCreate` de la clase `FotografiasDrive`, el siguiente código:

```
if (idCarpetaEvento==null){
   crearCarpetaEnDrive(evento, idCarpeta);
}
```

4. Inserta al final de la clase `FotografiasDrive` el código siguiente:

```
private void crearCarpetaEnDrive(final String nombreCarpeta,
                                                 final String carpetaPadre) {
   Thread t = new Thread(new Runnable() {
      @Override
      public void run() {
         try {
            String idCarpetaPadre = carpetaPadre;
            mostrarCarga(FotografiasDrive.this, "Creando carpeta...");
            //Crear carpeta EventosDrive
            if (idCarpeta==null){
               File metadataFichero = new File();
               metadataFichero.setName("EventosDrive");
               metadataFichero.setMimeType("application/vnd.google-apps.folder");
               File fichero = servicio.files().create(metadataFichero)
                                                          .setFields("id")
                                                              .execute();
               if (fichero.getId() != null) {
                  SharedPreferences prefs = getSharedPreferences("Preferencias",
                                                       Context.MODE_PRIVATE);
                  SharedPreferences.Editor editor = prefs.edit();
                  editor.putString("idCarpeta", fichero.getId());
                  editor.commit();
                  idCarpetaPadre=fichero.getId();
               }
            }
            File metadataFichero = new File();
            metadataFichero.setName(nombreCarpeta);
            metadataFichero.setMimeType("application/vnd.google-apps.folder");
            if (!idCarpetaPadre.equals("")){
               metadataFichero.setParents(Collections.singletonList
                                                      (idCarpetaPadre));
            }
            File fichero = servicio.files().create(metadataFichero)
                                                      .setFields("id")
                                                           .execute();
            if (fichero.getId() != null) {
               SharedPreferences prefs = getSharedPreferences("Preferencias",
                                                    Context.MODE_PRIVATE);
               SharedPreferences.Editor editor = prefs.edit();
               editor.putString("idCarpeta_"+evento, fichero.getId());
               editor.commit();
               idCarpetaEvento=fichero.getId();
               mostrarMensaje(FotografiasDrive.this, "¡Carpeta creada!");
            }
            ocultarCarga(FotografiasDrive.this);
         } catch (UserRecoverableAuthIOException e) {
            ocultarCarga(FotografiasDrive.this);
            startActivityForResult(e.getIntent(), SOLICITUD_AUTORIZACION);
         } catch (IOException e) {
            mostrarMensaje(FotografiasDrive.this, "Error;" + e.getMessage());
            ocultarCarga(FotografiasDrive.this);
```

```
            e.printStackTrace();
         }
      }
   });
   t.start();
}
```

5. Sustituye el caso `SOLICITUD_AUTORIZACION`, en el método `onActivityResult()`, en la actividad `FotografiasDrive`, por el siguiente:

```
case SOLICITUD_AUTORIZACION:
   if (resultCode == Activity.RESULT_OK) {
      crearCarpetaEnDrive(evento, idCarpeta);
   } else {
      noAutoriza=true;
      SharedPreferences prefs = getSharedPreferences("Preferencias",
                                                     Context.MODE_PRIVATE);
      SharedPreferences.Editor editor = prefs.edit();
      editor.putBoolean("noAutoriza", true);
      editor.commit();
      mostrarMensaje(this,"El usuario no autoriza usar Google Drive");
   }
   break;
```

   Si el usuario da la autorización, intentamos crear la carpeta de nuevo. Si no, mostramos un mensaje y guardamos en las preferencias la elección del usuario de no permitir el acceso de la aplicación a su carpeta de Drive.

6. Añade dentro del método `onActivityResult()`, en la opción `SOLICITUD_SELECCION_CUENTA` debajo de `editor.commit();`, la llamada:

```
crearCarpetaEnDrive(evento, idCarpeta);
```

7. Ejecuta la aplicación. Elije la cuenta de Gmail que la usará.

Comprueba que después de seleccionar una cuenta el Google Drive pide la autorización para que nuestra aplicación lo use. Posiblemente necesites desinstalar la aplicación primero para que funcione. Verifica que al autorizar el acceso se ha creado una carpeta en tu Google Drive llamada EventosDrive y, dentro de ella, una carpeta con el identificador del evento que has seleccionado.

Creamos la carpeta en el método `crearCarpetaEnDrive`. Para ello utilizamos una llamada a Drive API `file.create()`. Le pasamos los atributos de la carpeta. Para ello usamos la variable `metadataFichero` del tipo `File`. Le indicamos el nombre con `setName` y el tipo MIME con `setMimeType`. Una carpeta es del tipo `application/vnd.google-apps.folder`. Observa que en este método creamos la carpeta del evento y la carpeta EventosDrive, si no está creada.

Realizamos la llamada a la API de Google Drive para que nos cree la carpeta `servicio.files().create(metadataFichero).execute()`. Nos devolverá un valor en el campo `id`, indicado en `setFields`. Si el identificador no es **`null`**, mostramos un `Toast` indicando que se ha creado la carpeta con éxito. Consultamos su identificador con `ficheroSubido.getId()`. Por último, nos

guardamos el identificador en las preferencias de la aplicación para poderlo usar cada vez que el usuario utilice la aplicación.

Cuando creamos o consultamos un fichero en Google Drive, se nos devuelve una serie de campos. Los campos a devolver los indicamos en `setFields` separados por comas. Por ejemplo, podemos pedir que se nos devuelva el identificador y el nombre con `setFields("id, name")`.

Cuando realizamos una llamada a Drive API, como ya hemos comentado, se debe de realizar en un nuevo hilo (`Thread t = new Thread(new Runnable(){…}`). Recuerda que el nuevo hilo se ejecuta con `start()`.

Mientras se esté creando la carpeta, mostraremos un diálogo de carga, para hacer ver al usuario que la aplicación está trabajando.

Como hemos comentado, la primera vez que el usuario realice esta acción, ocurrirá un error. Se debe a que todavía no ha autorizado el uso de Google Drive. No hay problema, capturamos la excepción `UserRecoverableAuthIOException` y lanzamos una actividad donde se pedirá autorización. Curiosamente, la misma excepción nos indica la intención que nos permite lanzar esta actividad (`e.getIntent()`). A esta actividad le asociamos el código adecuado para poder procesar la respuesta en `onActivityResult()`.

Ejercicio: Subir un fichero a Google Drive

1. Declara en el manifiesto el siguiente `FileProvider`:

```
<provider
   android:name="android.support.v4.content.FileProvider"
   android:authorities="${applicationId}.provider"
   android:exported="false"
   android:grantUriPermissions="true">
   <meta-data
      android:name="android.support.FILE_PROVIDER_PATHS"
      android:resource="@xml/provider_paths"/>
</provider>
```

2. Crea una carpeta llamada xml en los recursos y dentro un fichero llamado provider_paths.xml con el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
   <external-path name="external_files" path="."/>
</paths>
```

3. Añade los siguientes métodos en la clase `FotografiasDrive` para hacer una fotografía con la cámara y obtener el fichero que se genera con la fotografia:

```
public void hacerFoto(View v) {
   if (nombreCuenta == null) {
```

```
      mostrarMensaje(this,"Debes seleccionar una cuenta de Google Drive");
   } else {
      Intent takePictureIntent =
                              new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
     if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
         java.io.File ficheroFoto = null;
         try {
            ficheroFoto = crearFicheroImagen();
            if (ficheroFoto != null) {
               Uri fichero = FileProvider.getUriForFile(
                           FotografiasDrive.this,
                           BuildConfig.APPLICATION_ID + ".provider",
                           ficheroFoto);
               uriFichero =
                    Uri.parse("content://"+ficheroFoto.getAbsolutePath());
               takePictureIntent.putExtra(
                                       MediaStore.EXTRA_OUTPUT, fichero);
             startActivityForResult(takePictureIntent,
                                             SOLICITUD_HACER_FOTOGRAFIA);
            }
         } catch (IOException ex) {
            return;
         }
      }
   }
}

private java.io.File crearFicheroImagen() throws IOException {
   String tiempo = new SimpleDateFormat("yyyyMMdd_HHmmss")
                                                      .format(new Date());
   String nombreFichero = "JPEG_" + tiempo + "_";
   java.io.File dirAlmacenaje =
         new java.io.File(Environment.getExternalStoragePublicDirectory(
                                  Environment.DIRECTORY_DCIM), "Camera");
   java.io.File ficheroImagen = java.io.File.createTempFile(
                                                         nombreFichero,
                                                         ".jpg",
                                                         dirAlmacenaje);
   return ficheroImagen;
}
```

Para hacer una fotografía, primero nos aseguramos de que ya se ha introducido la cuenta de Google. Luego obtenemos la ruta de la carpeta pública de la SD donde se guardan las fotografías. El código puede presentar incompatibilidades con dispositivos con versiones 2.3 o inferiores. Si es tu caso, prueba la opción de seleccionar la fotografía desde la galería que desarrollaremos seguidamente. A continuación, obtenemos un `string` a partir

de la fecha actual. Esto nos permitirá crear nombres de ficheros únicos. Con la información anterior, obtenemos el nombre del fichero. Finalmente, creamos una intención para que se haga la fotografía.

4. Añade el siguiente método en la clase `FotografiasDrive` para seleccionar una fotografía de la galería del dispositivo:

```
public void seleccionarFoto(View v) {
   if (nombreCuenta == null) {
      mostrarMensaje(this,"Debes seleccionar una cuenta de Google Drive");
   } else {
      Intent seleccionFotografiaIntent = new Intent();
      seleccionFotografiaIntent.setType("image/*");
      seleccionFotografiaIntent.setAction(Intent.ACTION_PICK);

startActivityForResult(Intent.createChooser(seleccionFotografiaIntent,
      "Seleccionar fotografía"),SOLICITUD_SELECCIONAR_FOTOGRAFIA);
   }
}
```

Para seleccionar una fotografía lanzamos una intención con la cual recuperaremos la fotografía seleccionada de la galería del dispositivo.

5. Recuerda añadir los permisos para escribir en la SD y usar la cámara en el Android manifest:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.CAMERA" />
```

Necesitamos los permisos para acceder al almacenamiento del dispositivo (`WRITE_EXTERNAL_STORAGE`) y la cámara (`CAMERA`). La aplicación utiliza la cámara o ficheros del almacenamiento interno para subir imágenes a la cuenta de Google Drive del usuario.

6. En el método `onCreate` de la clase `ActividadPrincipal` sustituye la declaración de la variable `PERMISOS` por:

```
String[] PERMISOS =
{android.Manifest.permission.WRITE_EXTERNAL_STORAGE,
android.Manifest.permission.GET_ACCOUNTS,
android.Manifest.permission.CAMERA};
```

7. Trataremos la fotografía hecha con la cámara y guardada de forma local correctamente. Para ello sustituye el caso `SOLICITUD_HACER_FOTOGRAFIA` en el `switch` de `onActivityResult`, en la clase `FotografiasDrive`:

```
case SOLICITUD_HACER_FOTOGRAFIA:
   if (resultCode == Activity.RESULT_OK) {
      guardarFicheroEnDrive(this.findViewById(android.R.id.content));
   }
   break;
```

8. Capturamos la ruta de la fotografía seleccionada de la galería. Para ello sustituye el caso `SOLICITUD_SELECCIONAR_FOTOGRAFIA` en el `switch` de `onActivityResult`, en la clase `FotografiasDrive`:

```
case SOLICITUD_SELECCIONAR_FOTOGRAFIA:
   if (resultCode == Activity.RESULT_OK) {
      Uri ficheroSeleccionado = data.getData();
      String[] proyeccion = { MediaStore.Images.Media.DATA };
      Cursor cursor = managedQuery(ficheroSeleccionado, proyeccion,
                                                        null, null, null);
      int column_index =
              cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA);
      cursor.moveToFirst();
      uriFichero =
         Uri.fromFile( new java.io.File(cursor.getString(column_index)));
      guardarFicheroEnDrive(this.findViewById(android.R.id.content));
   }
   break;
```

9. Copia al final de la clase `FotografiasDrive` el método `guardarFicheroEnDrive`:

```
private void guardarFicheroEnDrive(final View view) {
   Thread t = new Thread(new Runnable() {
      @Override
      public void run() {
         try {
            mostrarCarga(FotografiasDrive.this,"Subiendo imagen...");
            java.io.File ficheroJava = new java.io.File(uriFichero.getPath());
            FileContent contenido = new FileContent("image/jpeg", ficheroJava);
            File ficheroDrive = new File();
            ficheroDrive.setName(ficheroJava.getName());
            ficheroDrive.setMimeType("image/jpeg");
            ficheroDrive.setParents(Collections.singletonList(idCarpetaEvento));
            File ficheroSubido = servicio.files().create(ficheroDrive,
                                       contenido).setFields("id").execute();
            if (ficheroSubido.getId() != null) {
               mostrarMensaje(FotografiasDrive.this, "¡Foto subida!");
            }
            ocultarCarga(FotografiasDrive.this);
         } catch (UserRecoverableAuthIOException e) {
            ocultarCarga(FotografiasDrive.this);
            startActivityForResult(e.getIntent(), SOLICITUD_AUTORIZACION);
         } catch (IOException e) {
            mostrarMensaje(FotografiasDrive.this, "Error;"+e.getMessage());
            ocultarCarga(FotografiasDrive.this);
            e.printStackTrace();
         }
      }
   });
   t.start();
}
```

Creamos un fichero Google Drive en `ficheroDrive`. Estableceremos su nombre con `setName()`, el tipo MIME con `setMimeType()` y en qué carpeta queremos guardarla e indicaremos quién es el padre `setParents()`, además

de su contenido en la variable `contenido`. Por último, subimos el fichero usando el método `create()`.

10. Para finalizar, sobrescribe el método `onOptionsItemSelected` en la clase `FotografiasDrive` para otorgar funcionalidad a los elementos del menú de la actividad:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   View vista = (View) findViewById(android.R.id.content);
   int id = item.getItemId();
   switch (id) {
      case R.id.action_camara:
         if (!noAutoriza) {
            hacerFoto(vista);
         }
         break;
      case R.id.action_galeria:
         if (!noAutoriza) {
            seleccionarFoto(vista);
         }
         break;
   }
   return super.onOptionsItemSelected(item);
}
```

11. Ejecuta la aplicación y comprueba que se pueden subir fotografías a Google Drive.

Ya puedes probar la aplicación. Luego accede a Google Drive mediante la cuenta que has seleccionado en la aplicación para ver que realmente se ha subido el fichero. Ejecuta la aplicación en un dispositivo con Google Play. Si vas a utilizar un dispositivo real, conéctate por wifi para agilizar la subida.

Si utilizas un emulador, activa el soporte para la cámara en sus propiedades. Te recomiendo que uses la opción Emulated.

Práctica: Subir fotografías a una carpeta compartida

Haz que los usuarios de la aplicación suban las fotografías a una misma carpeta en Google Drive. Para ello comparte una carpeta en Google Drive con las siguientes opciones:

- Opciones de visibilidad: cualquier usuario que reciba este enlace.
- Acceso: cualquier usuario puede editar.

El URL proporcionado en "Enlace para compartir" tiene el siguiente formato: https://drive.google.com/folderview?id=0B0BmNZ_qoOveR0Y4Mk5vZGdkX00&usp=sharing

Entre ?id= y &usp= tenemos el identificador de la carpeta. 0B0BmNZ_qoOveR0Y4Mk5vZGdkX00 en el ejemplo.

Crea un nuevo menú en la aplicación llamado “Compartir Foto Cámara” o “Compartir foto galería”, según prefieras. Al presionar el botón haremos una fotografía con la cámara o la elegiremos de la galería y la subirá a dicha carpeta.

Al fichero que hemos subido en el ejercicio paso a paso, le hemos indicado un nombre, de qué tipo de archivo se trata y la carpeta en la que ha de guardarse. Se lo hemos indicado con `setParents` y el identificador de la carpeta donde va a guardarse. Todos los archivos almacenados en Google Drive son objetos y todos contienen un identificador. Dicho identificador es básico en las funciones de consulta de información en Drive.

3.2.4.4. Listar ficheros de Google Drive

Podemos listar el contenido de una carpeta realizando una búsqueda de todos los archivos que contiene. Debemos especificar qué campos queremos que nos devuelva la consulta. Por ejemplo, el nombre.

Ejercicio: Listar las fotografías subidas en un TextView

Vamos a proceder a listar los nombres de las fotografías subidas en un `TextView`.

1. Inserta debajo de `setContentView`, en el método `onCreate` de la clase `FotografiasDrive`, el siguiente código:

```
registerReceiver(mHandleMessageReceiver, new
                                    IntentFilter(DISPLAY_MESSAGE_ACTION));
mDisplay = (TextView) findViewById(R.id.display);
```

2. Copia y pega el siguiente código al final de la clase `FotografiasDrive`, que nos permitirá añadir texto en `mDisplay`:

```
@Override
protected void onNewIntent(Intent intent) {
   super.onNewIntent(intent);
   setIntent(intent);
}
private    final    BroadcastReceiver    mHandleMessageReceiver    =    new
BroadcastReceiver() {
   @Override
   public void onReceive(Context context, Intent intent) {
      String nuevoMensaje = intent.getExtras().getString("mensaje");
      mDisplay.append(nuevoMensaje + "\n");
   }
};
static void mostrarTexto(Context contexto, String mensaje) {
   Intent intent = new Intent(DISPLAY_MESSAGE_ACTION);
```

```
   intent.putExtra("mensaje", mensaje);
   contexto.sendBroadcast(intent);
}
```

3. Añade el siguiente código al final la clase `FotografiasDrive` para listar los nombres de los ficheros:

```
public void listarFicheros(View v) {
   if (nombreCuenta == null) {
      mostrarMensaje(this,
                     "Debes seleccionar una cuenta de Google Drive");
  } else {
     Thread t = new Thread(new Runnable() {
        @Override
        public void run() {
           try {
              mostrarCarga(FotografiasDrive.this,"Listando archivos...");
              FileList ficheros = servicio.files().list()
                                  .setQ("'"+idCarpetaEvento+"' in parents")
                                  .setFields("*")
                                  .execute();
              for (File fichero : ficheros.getFiles()) {
                 mostrarTexto(getBaseContext(), fichero.getOriginalFilename());
              }
              mostrarMensaje(FotografiasDrive.this,
                                                 "¡Archivos listados!");
              ocultarCarga(FotografiasDrive.this);
           } catch (UserRecoverableAuthIOException e) {
              ocultarCarga(FotografiasDrive.this);
              startActivityForResult(e.getIntent(),
                                                 SOLICITUD_AUTORIZACION);
           } catch (IOException e) {
              mostrarMensaje(FotografiasDrive.this,
                                              "Error;" + e.getMessage());
              ocultarCarga(FotografiasDrive.this);
              e.printStackTrace();
           }
        }
     });
     t.start();
   }
}
```

4. Sustituye el siguiente código que encontrarás en el método `onCreate` de la clase `FotografiasDrive`:

```
if (idCarpetaEvento==null){
    crearCarpetaEnDrive(evento,idCarpeta);
}
```

por:

```
if (idCarpetaEvento==null){
    crearCarpetaEnDrive(evento,idCarpeta);
```

```
} else {
    listarFicheros(this.findViewById(android.R.id.content));
}
```

5. Cada vez que subas un fichero, volverás a listar los archivos. Inserta debajo de la instrucción `mostrarMensaje(FotografiasDrive.this, "¡Foto subida!")`, en el método `guardarFicheroEnDrive` de la clase `FotografiasDrive`:

```
listarFicheros(view);
```

6. Ejecuta la aplicación.

El modo de funcionamiento es igual que cuando creamos la carpeta o subimos la fotografía: realizamos una llamada HTTP a Google Drive utilizando Drive API. En esta ocasión utilizamos `files().list()` y como parámetros le pasaremos una consulta para realizar la búsqueda de los ficheros de una carpeta (`.setQ("q='"+idCarpeta+"' in parents")`) y que nos devuelva el nombre original del fichero (`.setFields("originalFilename")`). Recorriendo la lista devuelta, insertamos en el `TextView` una línea por cada nombre de archivo. Habrás pensado que listar los nombres de las fotografías en un `TextView` no es la mejor opción. En la siguiente unidad mejoraremos la aplicación.

Puedes realizar búsquedas con `setQ`. Pregunta por los campos de la tabla siguiente, utilizando operadores como: `=`, `!=`, `contains`, `<`, `>`, `<=`, `>=`,... para obtener un lista archivos de Google Drive que cumplan la condición que desees:

Campo	Descripción
`name`	Nombre del fichero
`fullText`	Texto completo del fichero, incluidos nombre, descripción., contenido y texto indexable
`mimeType`	Tipo MIME del fichero
`modifiedTime`	Fecha de la última modificación
`viewedByMeTime`	Fecha de la última visualización del fichero por el usuario
`trashed`	Indica si el fichero se encuentra o no en la papelera
`starred`	Indica si el fichero ha sido marcado con una estrella
`parents`	Retorna los ficheros que tienen como padre un identificador específico
`owners`	Usuarios que son propietarios del archivo
`writers`	Usuarios que tienen permiso para modificar el fichero
`readers`	Usuarios que tienen permiso para leer el fichero
`sharedWithMe`	Ficheros que han sido compartidos con el usuario
`properties`	Propiedades públicas del fichero
`appProperties`	Propiedades personalizadas el fichero

Puedes consultar cómo realizar búsquedas con `setQ` en https://developers.google.com/drive/v3/web/search-parameters.

Con respecto a la información devuelta en JSON con los campos de cada fichero, que puedes filtrar en `setFields`, los puedes consultar en https://developers.google.com/drive/v3/reference/files/list#try-it realizando consultas de prueba.

Preguntas de repaso: Google Drive

CAPÍTULO 4.

Aplicaciones web en Android

VICENTE CARBONELL

El éxito de desarrollo de aplicaciones móviles es indiscutible: cada día más personas cuentan con un dispositivo móvil con conexión a Internet y descubren las ventajas que su uso supone. Por otra parte, cada vez son más los desarrolladores que persiguen llegar a más consumidores, por lo que se plantean lanzar su aplicación en diferentes plataformas.

Una posibilidad para desarrollar aplicaciones independientes de la plataforma consiste en usar tecnología web; es decir: HTML5, CSS3 y JavaScript. Las aplicaciones creadas con estas tecnologías pueden ser ejecutadas desde gran variedad de plataformas.

En este capítulo vamos a distinguir tres tipos de aplicaciones según la tecnología que utilicen:

Aplicaciones móviles nativas: Son aplicaciones creadas de forma convencional. El lenguaje utilizado y la forma de trabajar pueden ser muy diferentes según desarrollemos en Android, iOS o Windows Phone. Es importante no confundir con el concepto de aplicaciones en código nativo.

Aplicaciones web optimizadas para móviles: Son aplicaciones creadas utilizando principalmente tecnología web. Tienen la gran ventaja de su fácil portabilidad. Además, este tipo de aplicaciones pueden ser incrustadas dentro de una aplicación convencional, de forma que el usuario apenas nota la diferencia con respecto a estar ejecutando una aplicación nativa.

Aplicaciones híbridas: Usar tecnología web en una aplicación Android tiene limitaciones. En una aplicación híbrida seguiremos el siguiente planteamiento: trataremos de escribir el núcleo de nuestra aplicación usando tecnología web y, aquellas partes a las que la API web no nos permita llegar, las desarrollaremos como aplicación nativa.

Veremos las ventajas y desventajas de cada tipo de aplicacióne incidiremos en el uso de tecnología web dentro de nuestras aplicaciones.

Aprenderemos a utilizar aplicaciones web alojadas en un servidor, Firebase Hosting, y empaquetadas en la aplicación.

No nos olvidaremos de las herramientas de creación de aplicaciones multiplataforma. A partir de una aplicación web, podremos crear aplicaciones capaces de comunicarse con el sistema operativo del dispositivo. Además, podremos crear fácilmente diseños web adaptados a dispositivos táctiles.

Uno de los componentes fundamentales de Firebase es Firebase Analytics. Una solución de medida que proporciona estadísticas sobre el uso de una aplicación y la participación de los usuarios. Captura automáticamente eventos y propiedades de usuario Los informes de Analytics ayudan a entrender cómo se comportan los usuarios, poder segmentarlos y, así, tomar decisiones y realizar acciones sobre segmentos concretos de usuarios.

Objetivos

- Describir las distintas formas de desarrollo para plataformas móviles.
- Enumerar las ventajas y desventajas de cada tipo de desarrollo.
- Introducir el hospedaje web con Firebase Hosting.
- Mostrar cómo se pueden integrar elementos web en aplicaciones nativas.
- Integrar aplicaciones web alojadas en un servidor o empaquetadas con la aplicación.
- Describir cómo se pueden comunicar la parte nativa de la aplicación y la parte web.
- Mostrar algunas herramientas de creación de aplicaciones, independientemente de la plataforma móvil, basadas en tecnología web.
- Introducir el uso de Firebase Analytics en aplicaciones Android.

4.1. Introducción a la tecnología web

Muchas empresas y particulares ya tienen una página, un juego o una aplicación empresarial publicados en la web. Pero no disponen de una aplicación para dispositivos móviles. No es necesario crear desde cero una aplicación para cada tipo de dispositivo (Android, iOS, Windows Phone…), incluso hay veces que no es necesario ni crear una aplicación como tal. Bastará con optimizarla para ser visualizada desde un dispositivo móvil. Por definición, estas aplicaciones serán ejecutadas por un navegador web o módulo de ejecución web. Aunque hay que resaltar que en muchos casos el usuario no tendrá que abrir el navegador web para ejecutar la aplicación. Esta podrá ser empaquetada en una aplicación convencional.

Se denomina aplicación web a aquella que ha sido desarrollada utilizando tecnología web; es decir: HTML, CSS y JavaScript. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web y en la que se confía la ejecución al navegador. Esta forma de trabajar tiene la principal ventaja de facilitar la migración de aplicaciones entre sitios web y las diferentes plataformas móviles.

Las aplicaciones web son populares debido a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales. Existen aplicaciones como los webmails, wikis, weblogs, tiendas en línea o Google Docs son ejemplos bien conocidos de aplicaciones web.

Una aplicación web suele alojarse en un servidor web. Pero también es posible alojarla de forma local en el dispositivo del cliente. Esto nos permitirá acceder a la aplicación de forma más rápida y sin tener acceso a Internet. Es importante mencionar que una aplicación web debe contener elementos que permitan una comunicación activa entre el usuario y la información.

4.1.1. Aplicación web

En este capítulo vamos continuar trabajando con aplicación “Eventos” de la unidad anterior. En este apartado, crearemos una aplicación web desde el inicio que ejecutaremos desde un navegador web; pero, en los siguientes apartados, vamos a insertarla dentro de la aplicación “Eventos” para Android que hemos creado en el tema anterior.

Se trata de una aplicación sencilla formada por un archivo HTML con la estructura visual de la aplicación, un fichero CSS para dar estilo y un fichero JavaScript para otorgar funcionalidad. La aplicación contiene una lista de eventos; cuando el usuario pulsa sobre un elemento de la lista, mostrará la información del evento:

Ejercicio: Crear una aplicación web

1. Crea una carpeta en tu ordenador llamada eventos.
2. Dentro crea un fichero de texto plano con un editor de texto; llámalo index.html y pega el siguiente código:

```
<!doctype html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Eventos</title>
      <link href="estilos.css" rel="stylesheet" type="text/css">
      <script src="funciones.js"></script>
   </head>
   <body>
      <a href="#" onclick="muestraEvento('carnaval');">Carnaval</a>
      <a href="#" onclick="muestraEvento('fallas');">Fallas</a>
      <a href="#" onclick="muestraEvento('nochevieja');">Nochevieja</a>
      <a href="#" onclick="muestraEvento('sanjuan');">
                                                Noche de San Juan</a>
      <a href="#" onclick="muestraEvento('semanasanta');">
                                                     Semana Santa</a>
      <a href="#" onclick="alert('Curso Android Avanzado de la UPV.');">
                                                     Acerca de</a>
      <h1 id="evento"></h3>
      <p id="wiki"><p>
   </body>
</html>
```

3. Crea también en la carpeta otro fichero con el editor de textos llamado estilos.css y pega el siguiente código:

```
h1 {
   color: #111;
   font-family: 'Helvetica Neue', sans-serif;
   font-size: 50px;
   font-weight: bold;
   letter-spacing: -1px;
   line-height: 1;
   text-align: center;
}
p {
   color: #685206;
   font-family: 'Helvetica Neue', sans-serif;
   font-size: 14px;
   line-height: 24px;
   margin: 0 0 24px;
   text-align: justify;
   text-justify: inter-word;
}
```

```
a {
   font-family: 'Droid serif', serif;
   background: #b48608;
   padding: 4px 7px;
   color: #000;
   text-decoration: none;
   margin: 14px 14px 14px 14px;
   width: 90%;
   display: block;
   text-align: center;
   font-style: italic;
   font-size: 32px;
}
a:hover {
   color: #000;
   background: #fff;
}
#imagen {
   text-align: center;
}
```

4. Por último, crea otro fichero de texto llamado funciones.js con el siguiente código JavaScript:

```
var evento="";
var wiki="";
function muestraEvento(mEvento){
   switch (mEvento){
      case "carnaval":
         evento ="Carnaval";
         wiki="El carnaval es una celebración que tiene lugar
inmediatamente antes del inicio de la cuaresma cristiana, que se inicia a
su vez con el Miércoles de Ceniza, que tiene fecha variable (entre febrero
y marzo según el año). El carnaval combina algunos elementos como
disfraces, desfiles, y fiestas en la calle.";
         break;
      case "fallas":
         evento="Fallas";
         wiki = "Las Fallas (Falles en valenciano) son unas fiestas que
van del 15 al 19 de marzo con una tradición arraigada en la ciudad de
Valencia y diferentes poblaciones de la Comunidad Valenciana. Oficialmente
empiezan el último domingo de febrero con el acto de la Crida.";
         break;
      case "nochevieja":
         evento="Nochevieja";
         wiki = "La Nochevieja, víspera de Año Nuevo, Año Viejo o fin de año,
es la última noche del año en el calendario gregoriano, comprendiendo desde
31 de diciembre hasta el 1 de enero (Año Nuevo). Desde que se cambió al
calendario gregoriano en el año 1582, se suele celebrar esta festividad,
aunque ha ido evolucionando en sus costumbres y supersticiones.";
         break;
      case "sanjuan":
         evento="Noche de San Juan";
```

```
         wiki ="La víspera de San Juan o noche de San Juan es una festividad
cristiana, de origen pagano (Litha) celebrada el 23 de junio,1 en la que se
suelen encender hogueras o fuegos y ligada con las celebraciones en las que
se festejaba la llegada del solsticio de verano, el 21 de junio en el
hemisferio norte, cuyo rito principal consiste en encender una hoguera.";
         break;
      case "semanasanta":
         evento="Semana Santa";
         wiki="La Semana Santa es la conmemoración anual cristiana de la
Pasión, Muerte y Resurrección de Jesús de Nazaret. Por eso, es un período de
intensa actividad litúrgica dentro de las diversas confesiones cristianas. Da
comienzo el Domingo de Ramos y finaliza el Domingo de Resurrección,1 aunque
su celebración suele iniciarse en varios lugares el viernes anterior (Viernes
de Dolores) y se considera parte de la misma el Domingo de Resurrección.";
         break;
      default:
         wiki ="No se encuentra el evento";
   }
   muestra(evento,wiki);
}
function muestra(mEvento, mWiki){
   document.getElementById("evento").innerHTML=mEvento;
   document.getElementById("wiki").innerHTML=mWiki;
}
```

5. Abre el archivo index.html con un navegador, pulsa sobre varios eventos y comprueba que puedes ver su descripción.

4.1.2. Aplicación web online y offline

En la mayoría de las aplicaciones web los ficheros se alojan en un servidor web. En este caso estamos hablando de una **aplicación web en línea (online)**. Esto obliga al usuario a disponer de acceso a Internet para poder usar nuestra aplicación. Pero tiene sus ventajas: puede ser ejecutada sin necesidad de instalación y la aplicación se actualiza de forma automática modificando los ficheros del servidor. Para conseguir que el ejemplo del ejercicio anterior fuera en línea, tendríamos que copiar los tres ficheros creados a una carpeta de un servidor web. Para acceder a la aplicación se utilizaría el URL del fichero HTML.

Vamos a utilizar Firebase Hosting para alojar la aplicación web.

4.1.2.1. Aplicación online: Firebase Hosting

Firebase Hosting proporciona un alojamiento estático y dinámico, rápido y seguro para una aplicación web. Ofrece la infraestructura, las funciones y herramientas orientadas a la implementación y administración de sitios web.

Las funciones clave de Firebase Hosting son:

- **Proporciona una conexión segura:** SSL se integra con Firebase Hosting sin necesidad de configuración para que el contenido se proporcione siempre de forma segura.

- **Entrega rápida:** Independientemente de dónde se encuentren nuestros usuarios, el contenido se entrega de forma rápida.
- **Implementación rápida:** Puodemos tener una aplicación funcionando en segundos. Las herramientas de Firebase CLI (Command Line Interface) facilitan la publicación.
- **Reversiones con un solo clic:** Podemos deshacer errores, Firebase Hosting proporciona administración completa de versiones, y realizar reversiones con un solo clic.

Proporciona un subdominio en el dominio firebaseapp.com a nuestro proyecto. Al usar Firebase CLI, podemos implementar archivos desde directorios locales de nuestro ordenador al servidor Firebase Hosting. Los archivos se envían a través de una conexión SSL.

Además de alojar contenido estático, ofrece opciones de configuración para que podamos desarrollar aplicaciones web sofisticadas de forma progresiva. Podemos rescribir fácilmente el URL para el enrutamiento del cliente o configurar encabezados personalizados.

Una vez que estemos listos para llevar un sitio a producción, podemos conectar nuestro nombre de dominio propio a Firebase Hosting. Proporciona automáticamente un certificado SSL para el dominio, de modo que todo el contenido se ofrezca de forma segura.

Los pasos a seguir para utilizar Firebase Hosting son los siguientes:

1. **Instalamos el Firebase CLI:** Firebase CLI facilita la configuración de un nuevo proyecto, ejecutar un servidor de desarrollo local e implementar contenido.
2. **Configuramos un directorio para el proyecto:** Agrega archivos a la aplicación web en la carpeta local. Puedes probar el sitio localmente.
3. **Implementamos el sitio:** Cuando consideres que todo está correcto, sube los archivos a Firebase Hosting. Las nuevas versiones se lanzan todas al mismo tiempo, por lo que nunca tendrás que preocuparte por las implementaciones a medio terminar. Sin embargo, si algo sale mal, puedes revertir acciones con un solo clic.

Ejercicio: Aplicación web en Firebase Hosting

1. Instala Firebase CLI:

   Firebase CLI (Command Line Interface) requiere Node.js y npm. Puedes instalar ambos siguiendo las instrucciones en https://nodejs.org. Al instalar Node.js también se instala npm. Firebase CLI necesita la versión de Node.js 0.10.0 o superior.

   Una vez instalado Node.js y npm, puedes instalar Firebase CLI vía npm. Ejecuta desde la línea de comandos:

```
npm install -g firebase-tools
```

Para actualizar a la última versión, simplemente ejecuta el mismo comando.

2. Loguéate en Firebase:

   Ejecuta el comando `firebase login`. Si tienes una sesión de Google activa y que se pueda conectar a la consola de Firebase, devolverá el mensaje: "Already logged in as xxxx@gmail.com". Si no es así, se lanzará el navegador predeterminado para que inicies la sesión en una cuenta de Google y autorices el acceso a Firebase desde Firebase CLI. Puedes cerrar la sesión ejecutando `firebase logout`.

3. Inicializa la aplicación.

   Sitúate, en el directorio del ordenador donde tienes o vas a desarrollar el proyecto, en la consola del sistema ejecutando el comando `cd`. En nuestro caso accede a la carpeta eventos creada anteriormente, por ejemplo, con `c:\eventos` en Windows. Posteriormente, el comando `firebase init`:

```
cd ruta_carpeta_proyecto
firebase init
```

   El comando `firebase init` lanza un asistente de configuración del proyecto:

```
Are you ready to proceed? (Y/n)
```

   Introduce `y` y pulsa la tecla `INTRO` para indicar que estás de acuerdo en proceder con la configuración del proyecto.

```
What Firebase CLI features do you want to setup for this folder? (Press
<space> to select)
>( ) Database: Deploy Firebase Realtime Database Rules
 ( ) Firestore: Deploy rules and create indexes for Firestore
 ( ) Functions: Configure and deploy Cloud Functions
 (*) Hosting: Configure and deploy Firebase Hosting sites
 ( ) Storage: Deploy Cloud Storage security rules
```

   Podemos implementar varias características de Firebase en el proyecto mediante Firebase CLI: reglas de Database, Firestore o Storage; configurar y desarrollar Functions o Hosting. Debemos seleccionar los componentes a utilizar en nuestro proyecto. Elegiremos configurar y desarrollar un sitio en Firebase Hosting. Puedes navegar entre las opciones mediante las teclas de cursor en el teclado y mediante la tecla espacio, seleccionar o deseleccionar la opción. Si no seleccionas ninguna opción, el asistente no continuará y no se inicializará el proyecto en Firebase CLI. Para nuestro proyecto, dejaremos marcadas las opciones Firestore y Hosting. Pulsa la tecla `INTRO` para continuar.

```
What Firebase project do you want to associate as default? (Use arrow
keys)
> [don't setup a default project]
  Eventos (eventos-e9d26)
  [create a new project]
```

Asocia el proyecto en Firebase CLI a un proyecto en Firebase. Mueve con el cursor la selección hasta seleccionar **Eventos**, que es el proyecto con el que nos vamos a asociar. Pulsa la tecla INTRO para continuar. Puedes no asociar un proyecto y hacerlo más adelante, o bien crear un proyecto nuevo.

```
Your public directory is the folder (relative to your project directory)
that will contain Hosting assets to uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? (public)
```

Indicamos cual será la carpeta que contendrá los archivos del proyecto y que se subirán a Firebase Hosting. Por defecto, su nombre será public. Dejarás este nombre y seguirás adelante, pulsando la tecla INTRO.

```
? Configure as a single-page app (rewrite all urls to /index.html)? (y/N)
```

Si lo deseamos, nos creará una única página por defecto en la carpeta del punto anterior, en un fichero llamado index.html. Si, por ejemplo, un usuario pide cargar una página que no existe, se deberá tratar en index.html. Por defecto, no lo harás; así que, pulsa la tecla INTRO para continuar. Creará un archivo index.html, para el funcionamiento normal de la aplicación, y otro fichero 404.html, por si el usuario pide un recurso que no existe.

```
+  Firebase initialization complete!
```

Hemos terminado la inicialización del proyecto en Firebase CLI. La ejecución del comando `firebase init` crea un archivo de configuración firebase.json en la raíz del directorio del proyecto, una carpeta donde desarrollar el proyecto, por defecto, llamada public. Además, si decidimos utilizar otras opciones de Firebase CLI, creará los ficheros correspondientes. Por ejemplo, para las reglas de Firebase Database, creará un fichero de configuración de las reglas llamado database.rules.json.

4. Agrega los archivos.

   Cuando inicializas la aplicación con `firebase ini`, se te solicita que proporciones un directorio para usar como raíz pública (el valor predeterminado es public). Es en esta carpeta donde tienes que colocar los archivos de la aplicación. Vamos a aprovechar los archivos que has creado en la aplicación web “Eventos”. Muévelos a la carpeta public del proyecto en Firebase CLI.

5. Implementa el sitio web.

   Para implementar el sitio web, ejecuta desde la línea de comandos en la carpeta del proyecto en Firebase CLI:

```
firebase deploy
```

Si todo está correcto, nos devolverá dos URL:

- URL al proyecto en la consola de Firebase:
  `https://console.firebase.google.com/project/<id_proyecto>/overview`

- URL a Firebase Hosting del proyecto:
  `https://<id_proyecto>.firebaseapp.com`

La aplicación se implementará en el dominio:

<id_proyecto>.firebaseapp.com.

Accede a la consola de Firebase, al apartado HOSTING y comprueba que aparece un nuevo dominio cuyo nombre es el subdominio de `firebaseapp.com` que nos ha devuelto la implementación del proyecto mediante Firebase CLI.

6. Accede a la aplicación en Firebase Hosting.

Abre el navegador web de tu móvil. Puedes acceder a la aplicación web mediante el URL que encontrarás en la sección Hosting de la consola de Firebase. El URL tiene el siguiente aspecto: https://eventos-xxxx.firebaseapp.com/. Observa cómo el navegador trata de visualizar la información de la mejor manera posible.

Firebase Hosting solo funciona en conexiones SSL, solo transmitirá contenido a través de HTTPS. Debes asegurarte de que todos los recursos externos que no estén alojados en Firebase Hosting se carguen a través de SSL (HTTPS), incluida cualquier secuencia de comandos. La mayoría de los navegadores no permiten que los usuarios carguen "contenido combinado" (tráfico SSL y no SSL).

El comando `firebase init` crea un archivo de configuración firebase.json en la carpeta raíz del proyecto. La configuración predeterminada de firebase.json es la siguiente:

```
{
   "hosting": {
      "public": "app",
      "ignore": [
         "firebase.json",
         "**/.*",
         "**/node_modules/**"
      ]
   }
}
```

Vamos a ver las propiedades `public` e `ignore`:

1. `public`

   `"public": "app"`: (Obligatorio) Indica qué directorio se debe cargar a Firebase Hosting. Este directorio debe encontrarse dentro del directorio del proyecto y debe existir. El valor predeterminado es un directorio llamado `"public"`.

2. `ignore`

```
"ignore": [
   "firebase.json",
   "**/.*",
```

```
   "**/node_modules/**"
]
```

(Opcional) Especifica los archivos que se deben ignorar en la implementación.

Vamos modificar la aplicación web añadiendo un enlace y una imagen, ambos elementos no van a proporcionar una funcionalidad importante, pero nos va a servir para comprender algunos conceptos de las aplicaciones web que explicaremos más adelante. Además, practicaremos cómo revertir implementaciones en Firebase Hosting.

Ejercicio: Histórico de implementaciones y reversiones

1. Accede al apartado Hosting en la consola de Firebase.

   Comprueba que puedes ver el historial completo de tus implementaciones. Para revertir una implementación previa, desplázate sobre su entrada en la lista, haz clic en el icono del menú ampliado y luego en Revertir. En nuestro caso solo tendremos una implementación en el historial marcada como "Actual", no podemos revertir ninguna implementación.

2. Añade debajo de la etiqueta `<body>` en el archivo index.html de la carpeta `public` del proyecto y guarda los cambios:

```
<a href="http://www.androidcurso.com">ANDROID CURSO</a>
<p id="imagen">
   <img src="http://www.androidcurso.com/images/certificado_upv.jpg">
</p>
```

3. Abre el archivo index.html en el navegador de forma local.

   Comprueba que aparece un nuevo botón encima de todo llamado **Android curso**. Al pulsarlo accederemos a la web del curso http://www.androidcurso.com. Además, vemos una imagen de un muñeco Android. Pulsa en **Android curso** y comprueba que puedes navegar a la web del curso.

   Ahora tenemos una versión diferente en desarrollo (local) y en producción (Firebase Hosting). Vamos a solucionarlo.

4. Ejecuta `firebase deploy` para actualizar la versión de Firebase Hosting.

   Recuerda que esta instrucción la debes ejecutar desde la consola de comandos y estar situado en la carpeta del proyecto. Accede a la dirección web de proyecto y comprueba que la versión local y la de Firebase Hosting coinciden, que muestran lo mismo.

5. Accede a la sección Hosting en la consola de Firebase.

   Observa que tienes una nueva implementación. Una marcada como actual y otra, como implementada.

6. Restaura la versión que tienes marcada como implementada mediante el menú **REVERTIR**.
7. Comprueba que ahora la aplicación muestra la versión anterior.
8. Vuelve a restaurar la última versión que has implementado.

Hasta el momento, la forma de acceder a la aplicación web es mediante un subdominio de firebase.com, pero te puede interesar que la aplicación tenga un dominio personalizado, como por ejemplo midominio.com.

Firebase Hosting puede conectarse a un dominio personalizado, como https://midominio.com o https://miaplicacion.midominio.com. Firebase Hosting proporciona un certificado SSL para tu dominio y transmite tu contenido a través de una CDN global. Se admiten subdominios de hasta un nivel de profundidad. Los subdominios más profundos, como eventos.miaplicacion.midominio.com, no están permitidos.

Para conectar con un dominio personalizado sigue los siguientes pasos:

1. Verifica la propiedad del dominio.

Antes de conectar un dominio personalizado, deberás implementar tu sitio en Firebase Hosting. En el panel de administración de Hosting, podrás ver el historial de implementaciones y administrar dominios personalizados.

En el panel HOSTING de la consola de Firebase, selecciona Conectar dominio personalizado.

Introduce el dominio o subdominio que quieres que se asocie al subdominio de firebase.com.

Deberás actualizar las entradas del DNS de tu dominio añadiendo dos entradas TXT. Las entradas estarán disponibles en el panel HOSTING después de que se compruebe que tu dominio personalizado es válido y aún no está en uso. Según el proveedor de tu dominio, este paso de verificación puede ser instantáneo o tardar 1 hora o más.

2. Esperar la emisión del certificado SSL.

Una vez que se verifique la propiedad del dominio, se te proporcionará un certificado SSL para tu dominio y se implementará en la CDN global. Este proceso puede tardar entre 24 y 48 horas.

3. Activar el dominio.

Debes apuntar los registros DNS de tu dominio al servicio Firebase Hosting. Los registros específicos del DNS A o CNAME se mostrarán en la consola de Firebase una vez que se active el certificado de tu dominio. Cuando los cambios realizados al DNS se hayan propagado, los usuarios podrán acceder a tu sitio Firebase Hosting mediante el dominio conectado.

4. Redireccionamiento de subdominios con dominio personalizado.

Si tienes un dominio personalizado habilitado, puedes configurar subdominios arbitrarios que apunten a él; para eso, debes agregar registros CNAME al servidor

de tu nombre de dominio. Todos los subdominios se redireccionarán al dominio personalizado seleccionado con el código de estado HTTP 301.

Por ejemplo, si el dominio personalizado es midominio.com y agregas un registro CNAME para que www apunte a Firebase Hosting, quienes visiten https://www.midominio.com serán redireccionados a https://midominio.com.

Otro ejemplo: si especificas un subdominio como parte de tu dominio, como www.midominio.com, y configuras el registro A de tu dominio para que apunte a Firebase Hosting, todas las solicitudes para https://midominio.com serán redireccionadas a https://www.midominio.com.

Puedes redireccionar todos los subdominios a tu dominio personalizado al configurar un registro comodín CNAME en tu servidor DNS.

4.1.2.2. Aplicación offline

Existe otra posibilidad: alojar la aplicación de forma local en el dispositivo del cliente, con lo que tendríamos una **aplicación web fuera de línea (offline)**. Esto nos permitirá acceder a la aplicación de forma más rápida y sin tener acceso a Internet. Para conseguir que la aplicación que estamos desarrollando fuera totalmente offline, tendríamos que copiar la imagen del munñeco Android a una carpeta local. Por supuesto, puede no ser factible implementar esta función en todas las páginas y para todos los recursos del sitio. En nuestro ejemplo, sería factible aplicarla para la imagen, pero no para el contenido del enlace al sitio web Android Curso.

También podemos contemplar una tercera posibilidad: que nuestra aplicación fuera en línea cuando tuviéramos acceso a Internet y fuera de línea cuando no lo tuviéramos. De esta forma tendríamos las ventajas de las dos alternativas. Para conseguirlo, podemos utilizar la caché del navegador.

Todos los navegadores poseen una caché o espacio en memoria donde guardar elementos de páginas visitadas. Su objetivo es que al volver a visitarlas no sea necesario volver a descargar el contenido.

De esa forma, se acelera la navegación y se consume menos ancho de banda. El tiempo que deben permanecer en dicha caché los elementos de una página lo establece el servidor web donde están almacenados. El servidor envía al navegador, en los encabezados, dicha información al realizar la petición de la página. En un servidor web se usan las cabeceras `Expires` y `Cache-Control: max-age` para definir el tiempo que deben perdurar en la caché del navegador los elementos de una página. Este intervalo también se conoce como tiempo de refresco (freshness lifetime).

Para lograr que el contenido de una página web se muestre offline, tras ser cargada por primera vez, se necesita implementar un archivo de manifiesto que acompañe a la página. En él se indica al navegador los archivos que debe descargar, guardar y, posteriormente, cargar de la caché.

El archivo de manifiesto es un archivo de texto plano con extensión .manifest. Debe declararse en la página HTML y estar en su misma ubicación.

Dicho archivo debe contener en la primera línea: `CACHE MANIFEST`. A continuación, dejar un espacio e ir relacionando todos los archivos que se deben guardar. Comienza por incluir la misma página, posteriormente, añade el resto de archivos. Por ejemplo, los archivos de estilo, scripts, imágenes, etc.

Ejercicio: Definición del manifest de una aplicación web offline

1. Dentro de la carpeta public en el proyecto Firebase CLI, crea un fichero de texto plano con un editor de texto, llamado eventos.manifest y pega el siguiente código:

```
CACHE MANIFEST

index.html
estilos.css
funciones.js
http://www.androidcurso.com/images/certificado_upv.jpg
```

2. Vas a declarar el archivo manifest dentro de index.html. Abre el archivo index.html con el editor de texto y sustituye la etiqueta `<html>` por:

```
<html manifest="eventos.manifest">
```

3. Desconecta tu ordenador de Internet y abre el fichero index.html con un navegador. Observa cómo la imagen no se visualiza.
4. Conecta tu ordenador a Internet y recarga la página (pulsa “F5”). Observa cómo la imagen se visualiza.
5. Desconecta tu ordenador de Internet y vuelve a abrir la página. Observa cómo la imagen se visualiza gracias a que ahora está en la caché.
6. Publica la aplicación en Firebase Hosting ejecutando `firebase deploy`.
7. En la consola de Firebase, en la sección HOSTING, comprueba que hay una nueva implementación en el historial.
8. Abre la aplicación web implementada en Firebase Hosting.
9. Desconecta tu ordenador de Internet y vuelve a abrir la página. Observa cómo se visualiza gracias a que ahora está en la caché.

Ya tenemos preparada la aplicación. Pero lo único que podemos hacer hasta el momento es abrirla en un navegador en un dispositivo móvil. Pasemos a ver cómo podemos integrarla en una aplicación Android.

Preguntas de repaso: Introducción a las aplicaciones web

4.2. Uso de WebView

Como hemos comentado, existen muchas posibilidades de incluir contenido web en nuestras aplicaciones Android. En este apartado vamos a estudiar algunos ejemplos.

Android siempre ha sido sinónimo de conectividad y de ofrecer una experiencia genial de navegación web. ¿Por qué no construir nuestra aplicación o parte de ella con tecnología web? No solo se puede integrar una aplicación web, sino que podemos optimizar nuestros diseños para los distintos tamaños y densidades de pantalla. Podemos incrustar contenido web en una aplicación utilizando el control `WebView`.

4.2.1. Mostrar contenido web usando una intención

Una opción para integrar contenido web en una aplicación es abrir un URL mediante una intención. Vamos a verlo con un ejemplo:

Ejercicio: Abrir contenido web mediante un intent

1. Crea un nuevo proyecto con los siguientes datos:

   ```
   Application name: EventosWeb
   Package name: org.example.eventosweb
   Target Android Devices: Phone and Tablet
   Minimum Required SDK: API 14 (4.0)
   Add an activity to Mobile: Empty Activity
   Activity name: ActividadPrincipal
   Layout name: actividad_principal
   ```

2. Reemplaza el código de la clase `ActividadPrincipal` por:

```
public class ActividadPrincipal extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.actividad_principal);
      Intent intent = new Intent(Intent.ACTION_VIEW);
      Uri uri = Uri.parse(
                    "https://eventos-xxxx.firebaseapp.com/index.html");
      intent.setData(uri);
      startActivity(intent);
   }
}
```

   Sustituye **eventos-xxxx** por tu subdominio en Firebase Hosting.

3. Ejecuta la aplicación. Se abrirá la página indicada en la variable `uri` en el navegador por defecto del dispositivo.

Hemos visto una forma sencilla de integrar contenido web en nuestra aplicación. Sin embargo, tiene sus inconvenientes: no podemos saber cuándo el usuario sale de la aplicación o cuándo vuelve. Además, no controlamos qué está viendo.

Android SDK proporciona un control denominado `WebView` para visualizar páginas web, ya sea obteniéndolas a través de un URL de Internet o en un fichero local, o bien generando el código HTML directamente.

`WebView` está basado en el proyecto Open Source WebKit. Incluye un motor de representación de HTML/CSS y un intérprete de JavaScript. WebKit es utilizado en numerosas aplicaciones y es la base de navegadores como Safari, Chromium – Google Chrome u Opera.

4.2.2. Uso de un WebView para mostrar contenido web

Para mostrar contenido web en una aplicación, debemos hacerlo utilizando la vista `WebView`. Permite visualizar páginas web como parte del diseño de una actividad. Por defecto, no incluye todas las características de un navegador web, como controles de navegación o una barra de direcciones.

Ejercicio: Abrir contenido web en línea en un WebView

1. Abre el proyecto "EventosWeb".
2. Vamos a añadir un componente `WebView` al código del layout actividad_principal.xml. Sustituye su código por el siguiente:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
  <WebView
      android:id="@+id/webkit"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent" />
</LinearLayout>
```

3. Añade la siguiente variable a la clase `ActividadPrincipal`:

```
WebView navegador;
```

4. Sustituye el código del método `onCreate` de `ActividadPrincipal` por el siguiente:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.actividad_principal);
   navegador = (WebView) findViewById(R.id.webkit);
   navegador.loadUrl("https://eventos-xxxx.firebaseapp.com/index.html");
}
```

Para mostrar contenido de una página web en una aplicación, tienes que añadir un control `WebView`. Será el espacio donde se visualizará el contenido. Indica qué página vas a cargar con `loadUrl`.

5. Necesitas solicitar el permiso para la conexión a Internet en el `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.INTERNET" />
```

En el ejemplo del `Intent` no hacía falta el permiso porque quien realmente accedía a Internet era el navegador.

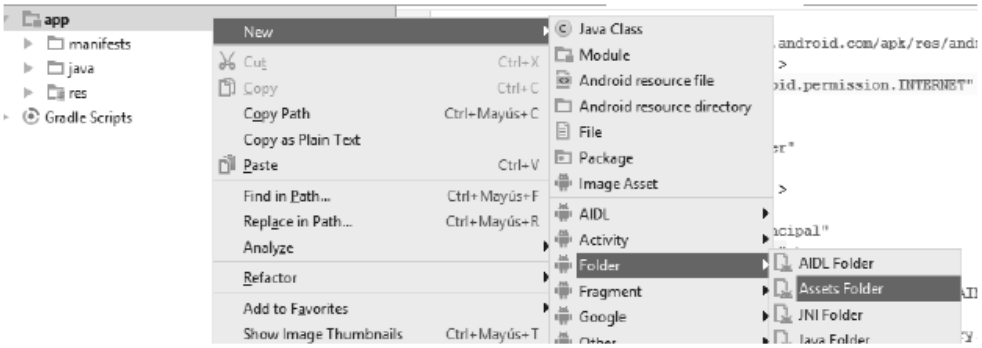
6. Ejecuta la aplicación y comprueba que se visualiza el contenido dentro de la aplicación.

Por defecto, en un `WebView` no puedes ejecutar JavaScript, puedes hacer zoom, los enlaces se abren en el navegador web… Por lo tanto, si pulsamos sobre el botón de un evento, no va a hacer nada. El enlace a la web del curso se abre fuera de la aplicación. Todas estas características las podemos cambiar.

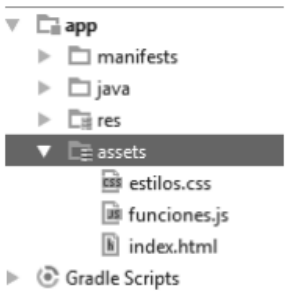
También podemos usar un `WebView` para mostrar contenido web fuera de línea integrada en la aplicación. Veamos el siguiente ejemplo.

Ejercicio: Abrir contenido web fuera de línea en un WebView

1. Abre el proyecto “EventosWeb”.
2. Si no existe, crea una carpeta llamada assets en la raíz del proyecto. Pulsa con el botón derecho sobre la carpeta app y en el menú flotante elije New / Folder / Assets folder.



3. Copia los ficheros que hemos creado anteriormente en la carpeta public del proyecto en Firebase CLI y pégalos dentro de la carpeta assets:



4. Cambia la dirección de `loadUrl` del `WebView` por `file:///android_asset/index.html` para que aceda a un fichero local.
5. Ejecuta la aplicación y observa que no se visualiza bien y que tampoco funcionan los botones.
6. Añade el siguiente código antes de la llamada a `loadURL` en el método `onCreate` de la clase `ActividadPrincipal`:

```
navegador.getSettings().setJavaScriptEnabled(true);
```

7. Ejecuta la aplicación y comprueba que ya funciona.
8. Pulsa el enlace ”Android Curso” y comprueba que se abre en el navegador.
9. Rota la pantalla. La actividad se reinicia y volvemos a la pantalla de inicio, perdiendo lo que tuviéramos en pantalla.
10. Quita el permiso de AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET" />
```

11. Verifica que la aplicación sigue funcionando.

En el ejercicio anterior hemos mostrado un ejemplo básico de cómo usar `WebView` para ejecutar una aplicación web fuera de línea. Sin embargo, esta aplicación tenía algunos inconvenientes. En el siguiente apartado trataremos de evitarlos y estudiaremos los aspectos básicos del control `WebView`.

4.2.3. Aspectos básicos de un WebView

Puedes consultar todos los métodos públicos de `WebView` en http://developer.android.com/reference/android/webkit/WebView.html. En este apartado vamos a describir unos cuantos métodos para desarrollar un navegador.

4.2.3.1. Evitar el reinicio de la actividad

Ejercicio: Evitar el reinicio de la actividad

1. Deshabilita las rotaciones sustituyendo en Android Manifest el código de la actividad `ActividadPrincipal` por el siguiente:

```
<activity
   android:name=".ActividadPrincipal"
   android:label="@string/app_name"
   android:configChanges="orientation|keyboardHidden"
   android:screenOrientation="portrait">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

2. Ejecuta la aplicación y comprueba que no se reinicia la actividad al cambiar la orientación de la pantalla.

Los eventos que pueden hacer que una actividad se reinicie son: un cambio de orientación de la pantalla o el uso del teclado en pantalla. Declarando `android:configChanges="orientation|keyboardHidden"` indicamos que *orientation* y *keyboardHidden* tienen su propio manejador; en este caso, vacío. A partir de Android 3.2 tenemos que utilizar `android:configChanges="orientation|keyboardHidden|screenSize"`. También podemos indicar la orientación de la pantalla con `android:screenOrientation` con los valores *portrait* o *landscape*.

4.2.3.2. Abrir los enlaces en el WebView

Ejercicio: Abrir los enlaces en el WebView

1. Vuelve a añadir el siguiente permiso en AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET" />
```

2. Añade al final del método `onCreate` de la clase `ActividadPrincipal` el siguiente código:

```
navegador.setWebViewClient(new WebViewClient()    {
   @Override
   public boolean shouldOverrideUrlLoading(WebView view, String url)
   {
      return false;
   }
});
```

3. Ejecuta la aplicación y observa que al pulsar un enlace no se abre en el navegador, sino en la aplicación.

Si queremos permanecer en nuestra aplicación al pulsar un enlace, hay que especializar la clase `WebViewClient` y sobrescribir el método `shouldOverrideUrlLoading` para que devuelva `false`. Este método recibe como parámetro el URL solicitado.

Puede ser interesante filtrar el contenido para que el usuario no se navegue a otra dirección web que no sea la indicada. Por ejemplo, vamos a impedir que un

usuario navegue a otra dirección que no sea http://www.androidcurso.com/. Si el usuario pulsa a un enlace dentro de la web, no va a poder acceder.

Ejercicio: Filtrar enlaces en un WebView

1. Ejecuta la aplicación y localiza un enlace dentro de la web “Android Curso” que varíe el URL. Puedes pulsar en uno de los vídeos que aparecen en la primera página. Te redirecciona a YouTube.
2. Inserta al principio del método `shouldOverrideUrlLoading` el siguiente código:

```
String url_filtro = "http://www.androidcurso.com/";
if (!url.toString().equals(url_filtro)){
   view.loadUrl(url_filtro);
}
```

3. Ejecuta la aplicación.

Pulsa el botón **Android curso**. Comprueba que al pulsar sobre el enlace que has localizado anteriormente, no carga el enlace definido en la web, sino que vuelve a cargar el mismo URL inicial.

4.2.3.3. Opciones de inicio

La clase `WebSettings` es usada para controlar los ajustes del `WebView`. Puedes consultar la documentación del objeto `WebSettings` en http://developer.android.com/reference/android/webkit/WebSettings.html.

Dos aspectos fundamentales en el uso de un `WebView` son: decidir si es capaz de ejecutar JavaScript y si el usuario puede hacer zoom en el contenido. Podemos controlarlo con las propiedades `setJavaScriptEnabled` y `setBuiltInZoomControls`.

Ejercicio: Habilitar JavaScript y deshabilitar el zoom

1. Añade el siguiente código antes de la llamada a `loadURL` en el método `onCreate` de la clase `ActividadPrincipal`:

```
navegador.getSettings().setJavaScriptEnabled(true);
navegador.getSettings().setBuiltInZoomControls(false);
```

2. Ejecuta la aplicación y comprueba que la aplicación funciona y no se puede hacer zoom.

4.2.3.4. Barra de progreso

El usuario no puede saber si una página se está cargando hasta que `WebView` la muestra. Puede dar la sensación de que la aplicación no responde. Es fácil de

comprobar si pulsas en el enlace de “Android Curso” de la aplicación que estamos desarrollando. Podemos proporcionar una barra de progreso con el widget `ProgressBar` y hacer uso de la clase `WebChromeClient` para saber el progreso de la carga de la página. Esta barra la podemos mostrar solamente cuando se esté cargando la página para ahorrar espacio en pantalla. Otra opción es crear un diálogo de progreso de carga, o bien, simplemente, mostrar el típico diálogo “Espere por favor…”.

Ejercicio: Añadir barra de progreso

1. Añade antes del `WebView`, la barra de progreso en el layout actividad_principal.xml:

```
<ProgressBar
   android:id="@+id/barraProgreso"
   style="?android:attr/progressBarStyleHorizontal"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content" />
```

2. Declara la barra de progreso en `ActividadPrincipal`:

```
private ProgressBar barraProgreso;
```

3. Añade el código que mostrará y ocultará la barra de progreso en la clase `ActividadPrincipal` al final del método `onCreate`:

```
barraProgreso = (ProgressBar) findViewById(R.id.barraProgreso);
navegador.setWebChromeClient(new WebChromeClient() {
   @Override
   public void onProgressChanged(WebView view, int progreso) {
      barraProgreso.setProgress(0);
      barraProgreso.setVisibility(View.VISIBLE);
      ActividadPrincipal.this.setProgress(progreso * 1000);
      barraProgreso.incrementProgressBy(progreso);
      if (progreso == 100) {
         barraProgreso.setVisibility(View.GONE);
      }
   }
});
```

4. Ejecuta la aplicación. Pulsa el enlace “Android Curso” para ver mejor la barra de progreso.

Hemos utilizado anteriormente `WebViewClient` y ahora `WebChromeClient`. Los dos son clientes web que pueden ser utilizados por `WebView`. Tienen diferentes métodos. Con `WebViewClient` podemos utilizar `onPageFinished`, `shouldOverrideUrlLoading`… Mientras que `WebChromeClient` permite alertas Javascript, `onProgressChanged`...

Lamentablemente, la barra de progreso no es muy visual para indicar a un usuario que espere. Vamos a sustituir la barra de progreso por un cuadro de diálogo con la frase “Cargando…”. Se mostrará cuando empiece la carga de la página y desaparecerá al final de la carga.

Ejercicio: Añadir diálogo de carga

1. Declara la siguiente variable en `ActividadPrincipal`:

```
ProgressDialog dialogo;
```

2. Reemplaza los métodos `onPageStarted` y `onPageFinished` de `setWebViewClient`:

```
@Override
public void onPageStarted(WebView view, String url, Bitmap favicon) {
   dialogo = new ProgressDialog(ActividadPrincipal.this);
   dialogo.setMessage("Cargando...");
   dialogo.setCancelable(true);
   dialogo.show();
}

@Override
public void onPageFinished(WebView view, String url) {
   dialogo.dismiss();
}
```

3. Ejecuta la aplicación y observa que aparece y desaparece el cuadro de diálogo en función del estado de la carga la página.

Con `ProgressDialog.show()` indicamos que aparezca un cuadro de diálogo en el contexto de `ActividadPrincipal`, que muestre “Cargando…” y añadimos `dialogo.setCancelable(true)` para que se pueda ocultar al pulsar sobre la pantalla. Utilizamos `dialogo.dismiss()` para ocultarlo.

Para mostrar la barra de estado hemos utilizado `WebChromeClient`. Hemos sobrescrito los métodos `onPageStarted` y`onPageFinished` de `WebViewClient`:

- `onPageStarted`: Se ejecuta al iniciarse la carga de una página.
- `onPageFinished`: Se invocará cuando haya terminado la carga de la página.

Utilizamos el cliente `WebChromeClient` o `WebViewClient` según los métodos que necesitemos.

4.2.3.5. Navegación

Vamos a ir completando un navegador web programando los botones **Sig**. (página siguiente), **Atrás** (página anterior) y **Stop** (detener la carga actual).

Ejercicio: Añadir funcionalidad página siguiente, anterior y detener carga

1. Inserta los botones **Stop**, **Atrás** y **Sig**. entre la barra de progreso y el `WebView` en el `layout` actividad_principal.xml:

```
<LinearLayout
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:orientation="horizontal" >
   <Button
      android:id="@+id/btnDetener"
      style="?android:attr/buttonStyleSmall"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="detenerCarga"
      android:text="Stop" />
   <Button
      android:id="@+id/btnAnterior"
      style="?android:attr/buttonStyleSmall"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="irPaginaAnterior"
      android:text="Atrás" />
   <Button
      android:id="@+id/btnSiguiente"
      style="?android:attr/buttonStyleSmall"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="irPaginaSiguiente"
      android:text="Sig." />
</LinearLayout>
```

2. Declara los botones en la clase `ActividadPrincipal`:

```
Button btnDetener, btnAnterior, btnSiguiente;
```

3. Inicializalos en `onCreate`:

```
btnDetener = (Button) findViewById(R.id.btnDetener);
btnAnterior = (Button) findViewById(R.id.btnAnterior);
btnSiguiente = (Button) findViewById(R.id.btnSiguiente);
```

4. Añade el siguiente código al final de la clase `ActividadPrincipal` para dotar de actividad a los botones:

```
public void detenerCarga(View v) {
   navegador.stopLoading();
}

public void irPaginaAnterior(View v) {
```

```
    navegador.goBack();
}

public void irPaginaSiguiente(View v) {
    navegador.goForward();
}
```

Utilizando los métodos `stopLoading()`, `goBack()` y `goForward()` detenemos la carga de la página, volvemos atrás o vamos a la página siguiente.

5. Sustituye los métodos `onPageStarted` y `onPageFinished` de `setWebViewClient`:

```
    @Override
    public void onPageStarted(WebView view, String url, Bitmap favicon) {
        dialogo = new ProgressDialog(ActividadPrincipal.this);
        dialogo.setMessage("Cargando...");
        dialogo.setCancelable(true);
        dialogo.show();
        btnDetener.setEnabled(true);
    }

    @Override
    public void onPageFinished(WebView view, String url) {
        dialogo.dismiss();
        btnDetener.setEnabled(false);
        if (view.canGoBack()) {
            btnAnterior.setEnabled(true);
        } else {
            btnAnterior.setEnabled(false);
        }
        if (view.canGoForward()) {
            btnSiguiente.setEnabled(true);
        } else {
            btnSiguiente.setEnabled(false);
        }
    }
```

Controlamos el estado de los botones de la siguiente forma: ”Atrás” estará habilitado si `canGoBack()` devuelve verdadero; «”Siguiente”» estará habilitado si `canGoForward()` devuelve verdadero; “Detener”» estará activo mientras dure la carga de la página. Al pulsar el botón **Stop** del diálogo de la carga se detendrá. Para pulsar el botón **Stop**, tendrás que pulsar una vez sobre la pantalla, para que se oculte el diálogo que indica que se está cargando la página, y después sobre el botón **Stop**.

6. Ejecuta la aplicación y comprueba que los botones añadidos funcionan correctamente.

Práctica: Añadir barra de direcciones

Modifica el layout actividad_principal.xml incluyendo un `EditText` llamado `txtDireccion` y, al lado y en la misma línea, un botón **Ir** llamado `btnIr`. Debe funcionar de la siguiente forma: introducir un URL en `txtDireccion` y posteriormente pulsar el botón `btnIr`. El `WebView` mostrará la página introducida. Recuerda que tienes que poner "http://" cuando escribas el URL. Recuerda que la aplicación tiene restringida cualquier dirección distinta de http://androidcurso.com en el método `shouldOverrideUrlLoading`. Quita la restricción o solo podrás probar el URL indicado.

4.2.3.6. Controlar el botón Volver

Es importante el control del botón **Volver** (BACK) del dispositivo. Por defecto, cuando lo pulsemos finalizaremos la actividad donde nos encontremos. Como en el proyecto que estamos desarrollando solo tenemos una actividad, finalizará la aplicación. Mientras que, en el navegador de Android, vuelve a la página anterior y la actividad no finaliza hasta que no pulsamos **Volver** desde la primera página mostrada. Este comportamiento puede ser implementado fácilmente capturando su pulsación con `onBackPressed`. Si podemos ir a la página anterior `canGoBack` ejecutamos `goBack`. Si no hay página anterior, entonces ejecutamos la acción por defecto del botón **Volver**.

Ejercicio: Controlar el botón **Volver** del dispositivo.

1. Sobrescribe el método `onBackPressed` añadiendo el siguiente código al final de la clase `ActividadPrincipal`:

```
@Override
public void onBackPressed() {
   if (navegador.canGoBack()) {
      navegador.goBack();
   }
   else {
      super.onBackPressed();
   }
}
```

2. Ejecuta la aplicación y comprueba su funcionamiento pulsando accediendo al enlace "Android Curso" y luego pulsando el botón **Volver** del dispositivo.

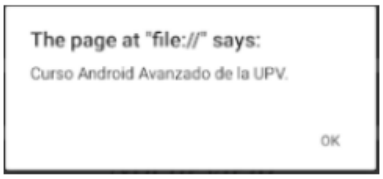
No funciona si se accede a la información de un evento y se pulsa el botón volver del dispositivo, ya que, aunque se muestre diferente contenido con la información detallada del evento, estamos en la misma página web. Para controlar este caso necesitaríamos realizar una programación especial que controlara en qué pantalla nos encontramos.

4.2.3.7. Capturar alertas JavaScript

`WebView` capturará alertas generadas con la instrucción alert («…») con JavaScript. Muchas páginas utilizan este sencillo cuadro de diálogo para comunicarse con el usuario. Si queremos modificar la forma con la que se muestran los mensajes, debemos sobrescribir el método `onJsAlert` de `setWebChromeClient`.

Ejercicio: Modificar alertas JavaScript

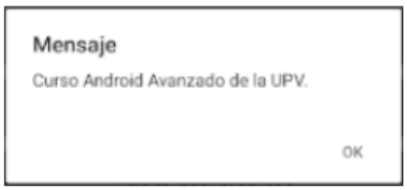
1. Ejecuta la aplicación y pulsa el botón **Acerca de**. Observa el diálogo que se muestra:



2. Sobrescribe el método `onJsAlert` de `setWebChromeClient` añadiendo el siguiente código:

```
@Override
public boolean onJsAlert(WebView view, String url, String message,
                  final JsResult result) {
   new AlertDialog.Builder(ActividadPrincipal.this).setTitle("Mensaje")
               .setMessage(message).setPositiveButton
                (android.R.string.ok,new AlertDialog.OnClickListener() {
      public void onClick(DialogInterface dialog,int which) {
         result.confirm();
      }
   }).setCancelable(false).create().show();

   return true;
}
```

3. Prueba a ejecutar el programa. Pulsa sobre el botón **Acerca de**. Observa el nuevo cuadro de diálogo que sustituye al anterior.



4.2.3.8. Gestión de errores

`WebView` mostrará en pantalla los errores que se produzcan. Además podemos recuperarlos para actuar en consecuencia, implementando el método `onReceivedError` de `WebViewClient`.

Ejercicio: Gestionar errores

1. Sobrescribe el método `onReceivedError` de `setWebViewClient` añadiendo el siguiente código:

```
@Override public void onReceivedError(WebView view, int errorCode,
                                      String description, String failingUrl) {

   AlertDialog.Builder builder =
         new AlertDialog.Builder(ActividadPrincipal.this);
   builder.setMessage(description).setPositiveButton("Aceptar",
                                         null).setTitle("onReceivedError");
   builder.show();
}
```

2. Prueba a ejecutar el programa, luego desactiva la conexión a Internet y, por último, pulsa el enlace a "Android Curso". Observarás que el error es capturado y mostrado como hemos definido.

4.2.3.9. Descargas

Otra funcionalidad proporcionada por `WebView` es el listener `DownloadListener`, que nos permitirá saber cuándo un URL solicitado es una descarga. Vamos a desarrollar la descarga de ficheros en la tarjeta SD con la aprobación del usuario.

Ejercicio: Descarga de un fichero en la tarjeta SD

1. Necesitas añadir el permiso en AndroidManifest.xml para escribir en la tarjeta SD:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

2. Inserta al final del método `onCreate` de la clase `ActividadPrincipal`:

```
ActivityCompat.requestPermissions(ActividadPrincipal.this,
        new String[]{android.Manifest.permission.WRITE_EXTERNAL_STORAGE},
        1);
```

3. Copia y pega al final de la clase `ActividadPrincipal` del siguiente código:

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[], int[]
grantResults) {
   switch (requestCode) {
      case 1: {
         if (grantResults.length > 0
               && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
```

```
            } else {
               Toast.makeText(ActividadPrincipal.this,
                     "Permiso denegado para escribir en el almacenamiento.",
                     Toast.LENGTH_SHORT).show();
            }
            return;
         }
      }
   }
```

4. Añade la clase asíncrona `DescargarFichero()` al final de la clase `ActividadPrincipal`, que nos permitirá guardar el fichero en la SD en todas las versiones de Android:

```
private class DescargarFichero extends AsyncTask<URL, Integer, Long> {
   private String mensaje;
   @Override
   protected Long doInBackground(URL... url) {
      String urlDescarga = url[0].toString();
      mensaje = "";
      InputStream inputStream =null;
      try {
         URL direccion = new URL(urlDescarga);
         HttpURLConnection conexion =
               (HttpURLConnection)direccion.openConnection();
         if (conexion.getResponseCode()== HttpURLConnection.HTTP_OK) {
            inputStream = conexion.getInputStream();
            String fileName = android.os.Environment
                  .getExternalStorageDirectory().getAbsolutePath() +
                  "/descargas";
            File directorio = new File(fileName);
            directorio.mkdirs();
            File file = new File(directorio, urlDescarga.substring(
                  urlDescarga.lastIndexOf("/"),
                  (urlDescarga.indexOf("?")==-1?
                  urlDescarga.length():urlDescarga.indexOf("?"))));
            FileOutputStream fileOutputStream = new
                  FileOutputStream(file);
            byte[] buffer = new byte[1024];
            int len = 0;
            int bytesRead = -1;
            while ((bytesRead = inputStream.read(buffer)) != -1) {
               fileOutputStream.write(buffer, 0, bytesRead);
            }
            fileOutputStream.close();
            inputStream.close();
            mensaje = "Guardado en: " + file.getAbsolutePath();
         } else {
            throw new Exception(conexion.getResponseMessage());
         }
      } catch (Exception ex) {
         mensaje = ex.getClass().getSimpleName() + " " + ex.getMessage();
      }
         finally {
```

```
            if (inputStream != null) {
               try {
                  inputStream.close();
               } catch (IOException e) {
               }
            }
         }
         return (long) 0;
      }
      protected void onPostExecute(Long result) {
         AlertDialog.Builder builder = new
               AlertDialog.Builder(ActividadPrincipal.this);
         builder.setTitle("Descarga");
         builder.setMessage(mensaje);
         builder.setCancelable(true);
         builder.create().show();
      }
}
```

5. Añade el siguiente código al final del método `onCreate` de la clase `ActividadPrincipal`:

```
navegador.setDownloadListener(new DownloadListener() {
   public void onDownloadStart(final String url, String userAgent, String
                contentDisposition, String mimetype,long contentLength) {
      AlertDialog.Builder builder =
            new AlertDialog.Builder(ActividadPrincipal.this);
      builder.setTitle("Descarga");
      builder.setMessage("¿Deseas guardar el archivo?");
      builder.setCancelable(false).setPositiveButton("Aceptar",
                                 new DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog, int id) {
            URL urlDescarga;
            try {
               urlDescarga = new URL(url);
               new DescargarFichero().execute(urlDescarga);
            } catch (MalformedURLException e) {
               e.printStackTrace();
            }
         }
      }).setNegativeButton("Cancelar",
               new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int id) {
          dialog.cancel();
      }
   });
   builder.create().show();
   }
});
```

Hemos añadido el escuchador `DownloadListener` a nuestro `WebView` para que detecte los enlaces que son descargas y los procese. Llamamos a una tarea asíncrona donde se procesa la descarga.

6. Ejecuta la aplicación.

   Para probar este ejercicio navega por el web de Android Curso donde podrás encontrar recursos descargables. Si no encuentras ningún recurso descargable, podrás realizar la prueba cargando directamente el siguiente URL `http://www.androidcurso.com/index.php/master/proyectos-finales/81-pf2017/proyectos-master-moviles-2017` en el `WebView`. Selecciona alguno de los proyectos para probar la descarga. Cuando termines recuerda volver a poner el siguiente URL en método loadUrl del `WebView`: `file:///android_asset/index.html`.

4.2.3.10.Conectividad

Siempre que en una aplicación necesitemos conectividad, debemos comprobar que la tenemos disponible antes de realizar cualquier solicitud web. Con `WebView` estas comprobaciones se hacen todavía más necesarias. Para ello hay que hacer uso de `ConnectivityManager` y `NetworkInfo`.

Ejercicio: Comprobación básica de conectividad

1. Necesitas añadir el permiso a AndroidManifest.xml para acceder al estado de la red:

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

2. Inserta al final del método `onCreate` de la clase `ActividadPrincipal`:

```
ActivityCompat.requestPermissions(ActividadPrincipal.this,
        new String[]{android.Manifest.permission.ACCESS_NETWORK_STATE},
        2);
```

3. Añade un nuevo caso en el método `onRequestPermissionResult` en la clase `ActividadPrincipal`:

```
case 2: {
   if (grantResults.length > 0
               && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
   } else {
      Toast.makeText(ActividadPrincipal.this,
                    "Permiso denegado para conocer el estado de la red.",
                     Toast.LENGTH_SHORT).show();
   }
   return;
}
```

4. Añade la siguiente función, al final de la clase `ActividadPrincipal`, que comprobará la conectividad:

```
private boolean comprobarConectividad() {
   ConnectivityManager connectivityManager =
         (ConnectivityManager) this.getSystemService(
         Context.CONNECTIVITY_SERVICE);
```

```
    NetworkInfo info = connectivityManager.getActiveNetworkInfo();
    if ((info == null || !info.isConnected() || !info.isAvailable())) {
        Toast.makeText(ActividadPrincipal.this,
                "Oops! No tienes conexión a internet",
                Toast.LENGTH_LONG).show();
        return false;
    }
    return true;
}
```

5. Sustituye `btnDetener.setEnabled(true)` de `onPageStarted` por lo siguiente:

```
if (comprobarConectividad()) {
    btnDetener.setEnabled(true);
} else {
    btnDetener.setEnabled(false);
}
```

6. Sustituye el código del método `irPaginaAnterior` por:

```
public void irPaginaAnterior(View v) {
    if (comprobarConectividad()) {
        navegador.goBack();
    }
}
```

7. Sustituye el código del método `irPaginaSiguiente` por:

```
public void irPaginaSiguiente(View v) {
    if (comprobarConectividad()) {
        navegador.goForward();
    }
}
```

8. Ejecuta la aplicación.

Puedes comprobar el funcionamiento, activando y desactivando la conexión a Internet del dispositivo o PC (si trabajas con un emulador). Tenemos, básicamente, todo lo necesario para controlar la información que se puede mostrar en un `WebView`: mostrar la información, navegar, controlar errores, comprobar la conectividad, etc.

Práctica: Integrar EventosWeb en Eventos

Complementa el proyecto `Eventos` con el proyecto `EventosWeb`.

Añade la siguiente opción de menú en `menu_detalles.xml`:

```
<item
    android:id="@+id/action_acercaDe"
    android:orderInCategory="104"
```

```
   android:title="Acerca de"
   app:showAsAction="never" />
```

Cuando se seleccione, se mostrará una nueva actividad llamada `EventosWeb` y se enviará el identificador del evento en los extras. Para ello añade el siguiente caso en el método `onOptionsItemSelected` de la clase `EventoDetalles`:

```
case R.id.action_acercaDe:
   Intent intentWeb = new Intent(getBaseContext(), EventosWeb.class);
   intentWeb.putExtra("evento", evento);
   startActivity(intentWeb);
   break;
```

Debes obtener el extra que has enviado en la nueva actividad `EventosWeb` declarando una variable llamada `evento` de tipo `String` y añadiendo en el método `onCreate`:

```
Bundle extras = getIntent().getExtras();
evento = extras.getString("evento");
```

Esta nueva actividad tendrá asociado el siguiente `layout` llamado `eventos_web.xml`:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
   <WebView
      android:id="@+id/webkit"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent" />
</LinearLayout>
```

Debe contener las siguientes características:

- Los archivos web deben estar alojados en Firebase Hosting.
- El área de visualización debe ser toda la pantalla.
- Se debe poder ejecutar.
- El usuario no podrá hacer zoom.
- No se debe reiniciar cuando esté activa (por ejemplo: por cambios de orientación de la pantalla).
- Controlar el botón **Volver** del dispositivo.
- Debe aparecer un diálogo que alerte al usuario cuando se esté cargando alguna página.
- Controla la conectividad del dispositivo.
- Todos los enlaces se abrirán en la aplicación.
- Personalizar el mensaje a mostrar cuando se llame al método `alert()` de JavaScript.
- No debe haber ni barra de progreso, ni botones de navegación, ni barra de dirección.

Preguntas de repaso: Uso de WebView

4.3. Diseño web en Android

El diseño en una aplicación web es muy importante para obtener buenos resultados. El usuario no debe tener la impresión de que es una página web.

Si queremos desarrollar una aplicación web para Android o rediseñar una existente, hay que tener en cuenta cómo se van a mostrar en diferentes tipos de pantallas. Cuando diseñemos páginas web para dispositivos móviles lo haremos con HTML5. Soporta funcionalidades creadas específicamente para adaptar contenido web a dispositivos móviles. Son dos los factores fundamentales a tener en cuenta: el tamaño del área de visualización (`viewport`) y el escalado, y la densidad de pantalla del dispositivo.

4.3.1. Área de visualización y escalado

Área de visualización (viewport)

El área de visualización (`viewport`) es el área en la cual una página web es mostrada. Aunque el área visible del `viewport` coincide con el tamaño de la pantalla, `viewport` tiene sus propias dimensiones, que determinan el número de píxeles disponibles para la página. Es decir, los móviles hacen un escalado para mostrar más píxeles de los reales, para que una página web diseñada para escritorio se vea completa. El ancho del `viewport` por defecto en Android es de 800 px.

Por eso cuando abrimos una página web con el navegador de un dispositivo Android, si no está adaptado, lo vemos más pequeño y encajado en pantalla.

En la figura anterior vemos una página web con una imagen de un ancho de 320 px donde no se ha establecido ninguna propiedad a `viewport`, por lo tanto el ancho de `viewport` es de 800 px.

Podemos controlar el `viewport` usando la propiedad `viewport` en HTML5 con la etiqueta `<meta name="viewport">`. Debe ser emplazada dentro de `<head>` en el documento HTML. Podemos establecer múltiples propiedades a `viewport` mediante el atributo `content`. Cada par propiedad-valor en el `content` debe ser separado por una coma.

La siguiente sintaxis muestra todas las propiedades soportadas por `viewport` y los tipos generales de valores aceptados por cada uno:

```
<meta name="viewport"
   content="
      height = [pixel_value | device-height] ,
      width = [pixel_value | device-width ] ,
      initial-scale = float_value ,
      minimum-scale = float_value ,
      maximum-scale = float_value ,
      user-scalable = [yes | no] ,
      target-densitydpi = [dpi_value | device-dpi |
         high-dpi | medium-dpi | low-dpi]
" />
```

Podemos establecer el ancho (`width`) y el alto (`height`) del `viewport` en un tamaño concreto en píxeles. Será el ancho y alto de que dispondrá la página web para ser mostrada. Por ejemplo, si establecemos el ancho del `viewport` de la página web de la imagen anterior en 400 px:

```
<meta name="viewport" content="width=400" />
```

En la figura anterior vemos la página web con un `viewport` de 400 píxeles y la imagen tiene un ancho de 320 píxeles.

Para el ancho, los valores mayores de 10 000 y menores de 320 son ignorados. Para el alto, valores mayores de 10 000 y menores de 200 son ignorados.

Como alternativa a especificar el ancho y el alto del `viewport` tenemos la posibilidad de establecer que su valor encaje con el tamaño de la pantalla con los valores `device-width` y `device-height`. Es apropiado utilizarlo cuando desarrollamos una aplicación web con un ancho fluido (no ancho fijo). Utilízalo si quieres que la página se visualice como si tuviera un ancho fijo. Se adapta perfectamente a todas las pantallas como si el ancho estuviera configurado para que coincida con cada pantalla.

Figura 1 Viewport = device-width.

En la figura anterior vemos la página web con una imagen de 320 píxeles que se adapta al ancho de la pantalla del dispositivo, ya que hemos establecido el `viewport` de la siguiente forma:

```
<meta name="viewport" content="width=device-width" />
```

Tienes que tener en cuenta que la imagen se ha escalado para que coincida con la anchura de la pantalla del dispositivo si la pantalla no es de densidadmedia.

4.3.2. Escalado

La escala del `viewport` define el nivel de zoom aplicado a la página web. Podemos especificar la escala en las propiedades del `viewport` con:

- `initial-scale`: Escala inicial de la página. Un valor tipo float que indica un multiplicador del tamaño de la página web con respecto al tamaño de la pantalla. Por ejemplo, si establecemos la escala inicial en 1.0 entonces la página web es mostrada para encajar con la resolución de la densidad objetivo 1:1. Si se establece a 2.0 la página se agranda con un factor de 2.

- `minimum-scale`: La escala mínima permitida. El valor que indica el multiplicador mínimo para el tamaño de la página web.
- `maximum-scale`: La escala máxima permitida para la página. El valor indica el valor máximo del multiplicador del tamaño de la página web relativo al tamaño de pantalla.
- `user-scalable`: Indica si el usuario puede cambiar la escala de la página (zoom in y out). Podemos establecer los valores: yes, para permitir hacer zoom al usuario, o no, para no permitirlo. Por defecto, su valor es yes.
  Si se establece a no, entonces `minimum-scale` y `maximum-scale` son ignorados.

Los rangos de valores de las escalas deben estar comprendidos entre 0.01 y 10.

4.3.3. Densidad de pantalla del dispositivo

La densidad de pantalla de un dispositivo es la relación entre la resolución de la pantalla y su tamaño, definida como el número de puntos por pulgada (dpi). Una pantalla con baja densidad es la que tiene disponibles menos píxeles por pulgada; sin embargo, una pantalla de alta densidad tiene más píxeles por pulgada. La densidad de una pantalla es importante porque, entre otras cosas, un elemento de la interfaz de usuario (como un botón), cuya altura y anchura son definidas en píxeles, aparecerá más grande en una pantalla de baja densidad y más pequeño en una de alta densidad. Para simplificar, Android engloba todas las densidades de pantalla actuales en tres densidades generales: alta, media y baja.

Por defecto, el navegador de Android y `WebView` escalan las páginas web como si estas se mostraran en una pantalla de densidad media. Así, aplica un escalado 1.5x en una pantalla de alta densidad (porque los píxeles son más pequeños) y un escalado 0.75x en una pantalla de baja densidad (porque los píxeles son mayores). Por esto, las imágenes de las figuras 1, 2 y 3 se muestran con el mismo tamaño, aunque las densidades de pantalla sean diferentes.

Controlar la densidad de pantalla mediante HTML

Utiliza la propiedad `target-densitydpi` de `viewport` para especificar la densidad para la cual la página web ha sido diseñada. Usa los siguientes valores:

- `device-dpi` : Usa la densidad nativa del dispositivo como densidad destino. Por defecto, nunca ocurre el escalado.
- `high-dpi` : Usa la alta densidad como densidad destino. Las pantallas de densidad media y baja se reducen proporcionalmente tanto como sea apropiado.
- `medium-dpi` : Usa la densidad media como densidad destino. Las pantallas de alta densidad aumentan el escalado y las de baja lo reducen. Este es el comportamiento por defecto.
- `low-dpi` : Usa la baja densidad como densidad destino. Las pantallas de media y alta densidad aumentan el escalado tanto como sea apropiado.

- Especifica un valor para usar como densidad destino. Los valores aceptados son 70-400.

Ejemplo de una etiqueta meta que especifica la densidad objetivo y el ancho:

```
<meta name="viewport" content="target-densitydpi=device-dpi, width=device-
width" />
```

Figura 2 Viewport =device −width y target −densitydpi = device −dpi.

En la figura anterior vemos cómo se ve la imagen con la densidad del dispositivo y el ancho del dispositivo.

Ejercicio: Establecer la propiedades de Viewport

1. En el proyecto Eventos modifica el fichero index.html que sea HTML5. Sustituye `<html>` por:

```
<!DOCTYPE HTML5>
<html lang="es">
```

2. En el mismo fichero, incluye la siguiente línea después de `<head>`:

```
<meta name="viewport" content="width=device-width, initial-scale=1,
target-densitydpi=medium-dpi">
```

3. Puedes comprobar cómo funciona `viewport` al cambiar el valor de las propiedades del punto 2. Por ejemplo, `initial-scale=2` o `densitydpi=high-dpi`.

Controlar la densidad de pantalla mediante JavaScript

También podemos controlar la densidad con la que se muestra una aplicación web haciendo uso de JavaScript y DOM. El valor de la propiedad de DOM window.devicePixelRatio especifica el valor por defecto de la escala usada por el dispositivo actual. Por ejemplo, si el valor de window.devicePixelRatio es 1.0, entonces el dispositivo es considerado como un dispositivo de densidad media (mdpi) y el escalado por defecto no es aplicado a la página web; si el valor es 1.5, entonces el dispositivo es considerado de alta densidad (hdpi) y el contenido de la página web es escalado 1.5x; si el valor es 0.75, entonces el dispositivo es considerado como un dispositivo de baja densidad (ldpi) y el contenido es escalado 0.75x. Sin embargo, si especificamos la propiedad de meta target-densitydpi, entonces este comportamiento de escalado no se aplica.

Controlar la densidad de pantalla mediante CSS

Usamos la consulta CSS `-webkit-device-pixel-ratio` para especificar las densidades de pantalla para las cuales una hoja de estilo será usada. El valor correspondiente debe ser 0.75, 1, o 1.5, para indicar que los estilos son para dispositivos con baja, media o alta densidad de pantalla, respectivamente. Por ejemplo:

```
<link rel="stylesheet" media="screen and (-webkit-device-pixel-ratio:1.5)"
href="hdpi.css" />
```

La hoja de estilo hdpi.css es usada solamente para dispositivos con una pantalla con una ratio de píxeles de 1.5, la cual es una ratio de píxeles de alta densidad.

4.3.4. Depuración remota en Android con Chrome

Para realizar diseños, ajustes en la visualización o para retocar código HTML o JavaScript, nos será útil poder depurar. Si dispones de un dispositivo con Android KitKat o superior conectado al ordenador de desarrollo mediante USB, puedes depurar el contenido de un `WebView` usando el navegador Chrome. La depuración se realiza de la misma forma que con una aplicación web desde un ordenador. Los cambios que realices mediante el navegador se verán reflejados automáticamente en el `WebView`.

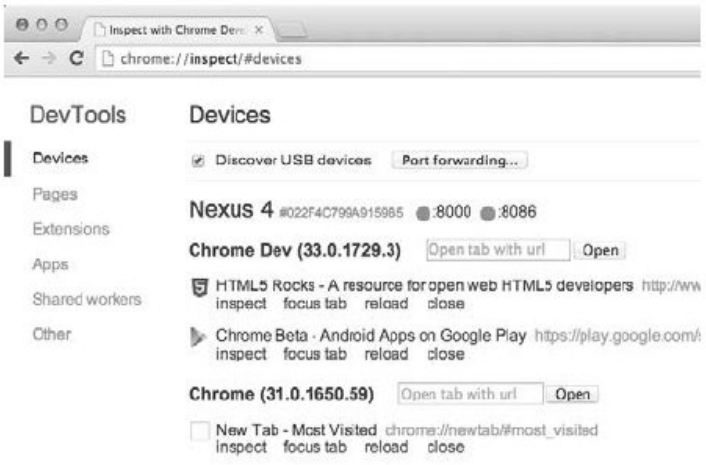
Si dispones de un dispositivo con la Android KiKat o superior podrás realizar el siguiente ejercicio paso a paso. Para depurar un WebView mediante el navegador sigue los siguientes pasos:

Ejercicio: Depuración remota de un WebView mediante Chrome

1. Activa la depuración mediante USB en el dispositivo. En Ajustes / Opciones de desarrollo, verifica que está marcada la opción Depuración USB
2. En tu equipo, accede a la página chrome://inspect con el navegador Chrome y asegúrate que está marcado Discover USB devices
3. Los WebView que quieras depurar tienen que estar habilitados para la depuración. Añade el siguiente código para habilitarlos:

```
if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
   WebView.setWebContentsDebuggingEnabled(true);
}
```

4. Conecta el dispositivo mediante USB al ordenador. Puede que en el dispositivo te pida que aceptes el permiso para realizar poder depurar mediante USB a través de tu ordenador.
5. Una vez que hayas configurado el dispositivo para la depuración por USB, la página chrome://inspect mostrará los dispositivos conectados, así como las pestañas del navegador abiertas y los WebView habilitados para la depuración:



6. Encuentra el WebView en el que estés interesado y pulsa sobre el enlace inspect para abrir las herramientas de desarrollo:

4.4. Aplicaciones híbridas

Si creamos una aplicación web adaptada a dispositivos móviles alojada en un servidor web, tenemos la ventaja de su fácil portabilidad. Pero también comporta que la aplicación no pueda acceder a los recursos del dispositivo "notificaciones de la barra de estado". ¿Por qué no aprovechar ambas ventajas? Para eso necesitamos comunicar la parte nativa de la aplicación con la parte web.

Para crear una aplicación híbrida (nativa-web), hemos visto que tenemos que utilizar `WebView` para mostrar la parte web. Para realizar la comunicación debemos permitir la ejecución de JavaScript en el `WebView` estableciendo `setJavaScriptEnabled` a `true`.

La comunicación desde la parte web a la nativa, la realizaremos mediante una **interfaz de comunicación**. Usaremos el lenguaje Java en la parte nativa y JavaScript en la parte web. Para la comunicación desde la parte nativa a la web, utilizaremos el método `loadUrl` del `WebView`.

Vamos a modificar el proyecto "Eventos" para que cierre la actividad donde muestra la información del evento desde la aplicación web mediante un botón html (comunicación desde `WebView` a Android) y modificaremos la aplicación web para que muestre la información del evento que está consultando el usuario (comunicación desde Android a `WebView`).

La aplicación web se encuentra en Firebase Hosting. Cualquier cambio va a necesitar que ejecutes `firebase deploy` para subir los cambios a Firebase Hosting, antes de comprobar el resultado en la aplicación. Es una buena práctica, si la aplicación web no es muy complicada, que durante el desarrollo utilices un servidor local o, si la aplicación contiene comunicación web a nativa, la integres dentro de la aplicación. Cuando la publiques, solamente tendrás que quitar la carpeta `assets` del empaquetado y cambiar la dirección a la que apunta el `WebView`. Te recomiendo que lo hagas. Copia el contenido de la carpeta donde tengas el proyecto (si estás utilizando Firebase CLI el contenido de la carpeta public) a la carpeta `assets` del proyecto y cambia el URL del `WebView`.

Ejercicio: Comunicación Android-WebView

1. Sustituye el contenido del fichero index.html por:

```
<!doctype html>
<html>
<head>
   <meta charset="utf-8">
   <title>Eventos</title>
   <link href="estilos.css" rel="stylesheet" type="text/css">
   <script src="funciones.js" align="center"></script>
</head>
<body>
   <h1 id="evento"></h3>
```

```
   <p id="wiki"><p>
   <a href="#" onClick="volver();">Volver</a>
</body>
</html>
```

Hemos quitado el enlace al sitio web del curso, la imagen y los botones. Hemos dejado el elemento "evento" (título) y "wiki" (texto), para mostrar la descripción de un evento. Además, hemos añadido un botón **Volver** para cerrar la actividad. El botón **Volver** llama a una función javascript llamada `volver()`.

2. Inserta al final del método `onPageFinished` de la clase `EventosWeb`:

```
navegador.loadUrl("javascript:muestraEvento(\""+evento+"\");");
```

3. Ejecuta la aplicación.

Comprueba que al acceder a la opción de menú `Acerca de` de un evento, se muestra la descripción del evento.

Para comunicarnos con una aplicación web de un `WebView`, necesitamos llamar a una función JavaScript desde la parte nativa. Para ello hemos de utilizar `loadUrl()` de `WebView`; pero en vez de pasarle un URL, le parasemos `javascript:nombre_funcion()`. Así, para mostrar la información de un evento ejecutamos `navegador.loadUrl("javascript:mostrarEvento(\""+evento+"\")")`. La función JavaScript necesita que le pasemos un parámetro, que en nuestro caso será el identificador del evento que queremos mostrar.

Ahora veamos la parte contraria, cómo nos comunicamos desde la parte web con la parte nativa Android. Para ello le otorgaremos funcionalidad al botón **Volver** que hemos creado en la actividad `EventosWeb`.

Ejercicio: Comunicación WebView-Android

1. Añade el siguiente código al final de la clase `EventosWeb`:

```
public class InterfazComunicacion {
   Context mContext;

    InterfazComunicacion(Context c) {
      mContext = c;
   }
   @JavascriptInterface
   public void volver(){
      finish();
   }
}
```

Hemos creado la interfaz de comunicación entre Android y el `WebView` mediante la interfaz `InterfazComunicacion`.

2. Declara un objeto de esta clase añadiendo el siguiente código:

```
final InterfazComunicacion miInterfazJava = new InterfazComunicacion(this);
```

3. Tienes que añadir la interfaz que acabas de crear al `WebView`; para ello, al final de `onCreate`, añade:

```
navegador.addJavascriptInterface(miInterfazJava, "jsInterfazNativa");
```

NOTA: Para la versión Android 4.2 (API 17) y posteriores añade `@SuppressLint("JavascriptInterface")` delante de `onCreate` en la clase `EventosWeb`.

4. Por último debes crear la función JavaScript `volver()` que se ejecuta cuando se pulsa el botón **Volver**. Añade al final del fichero funciones.js del proyecto Firebase Hosting:

```
function volver(){
   jsInterfazNativa.volver();
}
```

5. Ejecuta la aplicación.

Observarás que al pulsar sobre el botón **Volver** se cierra la actividad.

En este ejercicio hemos creado una interfaz de comunicación creando la clase `InterfazComunicacion`. En esta clase, tenemos que definir las funciones que podrán ser llamadas desde JavaScript y se ejecutarán en Android. En nuestro caso, tenemos el método `volver()`, que cierra la actividad.

Preguntas de repaso: Aplicaciones híbridas

4.5. Alternativas en la programación independiente de la plataforma para móviles

Hemos visto cómo crear aplicaciones web e integrarlas dentro de nuestra aplicación Android. También, qué tenemos que tener en cuenta a la hora de diseñar aplicaciones web para dispositivos móviles y cómo se comunica la parte nativa con la parte web. Es fácil pensar que utilizando HTML5, CSS y JavaScript, podemos encontrar herramientas que nos permitan acelerar el desarrollo de todo lo que hemos visto. Efectivamente, es así: en los últimos años han aparecido herramientas basadas en tecnología web para el desarrollo de aplicaciones móviles multiplataforma. Destacamos dos de estas herramientas: PhoneGap, que nos permitirá desarrollar aplicaciones multiplataforma sin necesidad de escribir código nativo; y jQuery Mobile para el desarrollo de interfaces móviles amigables y de rápida implementación. Vamos a ver una introducción a JQuery Mobile para saber cómo integrarlo en una aplicación Android.

En octubre de 2014 se liberó la versión definitiva de HTML5, por tanto, ya no es una recomendación, sino un estándar soportado por los navegadores. La gran ventaja que tiene HTML5 es el hecho de la evolución que se le ha imprimido con nuevos elementos y API de desarrollo, teniendo en algunos casos acceso al hardware del dispositivo. Pero no todos los dispositivos soportan por completo todas las novedades de HTML5.

4.5.1. jQuery Mobile

jQuery Mobile es un framework (entorno de trabajo) de desarrollo de interfaces de usuario en dispositivos móviles. Permite crear de una forma sencilla unas interfaces atractivas con multitud de componentes: botones, deslizadores, pestañas… Se visualizan en cualquier tipo de dispositivo y están preparadas para entornos táctiles. Se basa en jQuery.

Como habrás podido comprobar, Android Studio y Android no son el mejor sistema para desarrollar interfaces gráficas, sobre todo si lo comparamos con las herramientas y frameworks que existen para el diseño web. Pero también existen frameworks multiplaforma para el desarrollo web móvil que podemos utilizar en nuestras aplicaciones, como jQuery Mobile, Sencha, Titanium… por citar algunos. Estos frameworks permiten desarrollar de una forma rápida webs para dispositivos móviles mediante el uso de HTML5, JavaScript y CSS3. Nos permiten generar páginas cuya apariencia será muy similar, con independencia del dispositivo desde el que se acceda a ellas.

Hemos elegido jQuery Mobile porque es gratuito y con una curva de aprendizaje muy rápida para poder introducirnos en este tipo de frameworks y usarlo en nuestras aplicaciones.

Podemos utilizarlo para: el desarrollo de aplicaciones rápidas con un diseño y funcionalidad muy interesantes; integrarlo en nuestra aplicación nativa; o usarlo mediante PhoneGap y obtener rápidamente una aplicación multiplataforma sin programar prácticamente nada nuevo.

Para utilizar jQuery Mobile en una página web, tiene que soportar HTML5 y tenemos que hacer referencia a jQuery, jQuery Mobile y a la hoja estilos de jQuery Mobile. Tendremos que añadir la referencia al web de JQuery Mobile y JQuery o descargarlos e indicar que se encuentran en una carpeta local de nuestro proyecto. El siguiente código muestra cómo se realiza cuando hacemos referencia a la web:

```
<link rel="stylesheet"
  href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css"/>
<script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
<script
     src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
</script>
```

Luego, no nos queda más que utilizar etiquetas y atributos para dotar de diseño y funcionalidad a la página. Vamos a explicar un ejemplo básico de una página web realizada con jQuery Mobile.

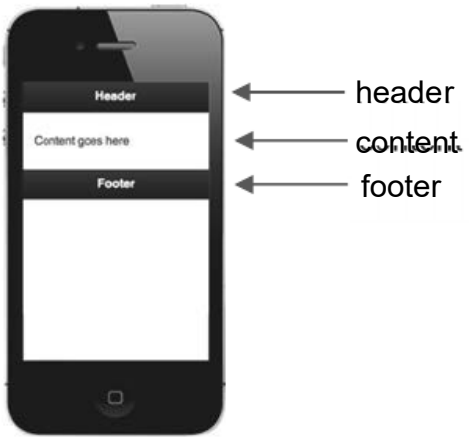
4.5.1.1. Crear una página básica

El siguiente código es un ejemplo de una página básica creada con jQuery Mobile:

```
<!DOCTYPE html>
<html>
<head>
   <title>Hola mundo</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href=
    "http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css"/>
   <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
   <script src=
     "http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
   </script>
</head>
<body>
<div data-role="page">
   <div data-role="header">
      <h1>Cabecera</h1>
   </div>
   <div data-role="content">
      <p>Contenido</p>
   </div>
</div>
</body>
</html>
```

Con este simple código hemos creado una página con una cabecera y un contenido. Vamos a explicar lo que contiene. En el `head`, la etiqueta meta `viewport` ajusta el ancho de la pantalla al ancho en píxeles del dispositivo y hacemos referencia a jQuery, jQuery Mobile y a la hoja de estilos.

En el `body`, con un elemendo HTML `div` con atributo `data-role="page"`, creamos la envoltura usada para delimitar una página. Dentro del `div` de la página incluimos otro `div` para la barra de cabecera. Lo identificamos con el atributo `data-role="header"`. También incluimos en la página otro `div` para la región de contenido con el atributo `data-role="content"`. Los divs de cabecera y contenido son opcionales; pero, como mínimo, incluye el contenido para no mostrar una página vacía. También se puede declarar la región del pie de una página con `data-role="footer"`. No lo hemos utilizado en el ejemplo. Los atributos `data-` son atributos HTML5 usados a través de jQuery Mobile para transformar una marca básica en un widget mejorado y con estilo.



Ejercicio: Crear una página web con jQuery Mobile

1. Crea una carpeta en tu ordenador llamada jquerymobile_basico.
2. Crea un fichero dentro de texto llamado index.html.
3. Pega dentro del fichero el código HTML anterior.
4. Abre el fichero con un navegador que soporte HTML5 y observa el resultado.

   NOTA: En las últimas versiones de Chrome existe un parche de seguridad que impide el visionado de páginas jQuery Mobile en local. No así las cargadas desde un servidor. Si te ocurre esto, utiliza Mozilla Firefox o Internet Explorer, o bien añade el siguiente código justo delante de: <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>

```
<script>
$(document).bind('mobileinit',function(){
    $.mobile.changePage.defaults.changeHash = false;
    $.mobile.hashListeningEnabled = false;
    $.mobile.pushStateEnabled = false;
});
</script>
```

5. Crea un nuevo proyecto con Android Studio:

```
Application name: EjemploJQueryMobile
Package name: org.example.ejemplojquerymobile
Target Android Devices: Phone and Tablet
Minimum Required SDK: API 14 (4.0)
Add an activity to Mobile: Blank Activity
Activity name: ActividadPrincipal
Layout name: activity_main
Title: Ejemplo JQuery Mobile
Menu Resource Name: menu_principal
```

6. Copia el fichero index.html que acabas de crear en la carpeta assets en el proyecto.

7. Reemplaza el código del layout activity_main.xml por:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
   <WebView
      android:id="@+id/webkit"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent" />
</LinearLayout>
```

8. Añade la siguiente variable a la clase `ActividadPrincipal`:

```
WebView navegador;
```

9. Sustituye el código del método `onCreate()` de `ActividadPrincipal` por el siguiente:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   navegador = (WebView) findViewById(R.id.webkit);
   navegador.getSettings().setJavaScriptEnabled(true);
   navegador.getSettings().setBuiltInZoomControls(false);
   navegador.loadUrl("file:///android_asset/index.html");
}
```

10. Deshabilita las rotaciones sustituyendo en Android Manifest el código de la actividad `ActividadPrincipal` por el siguiente:

```
<activity
   android:name="org.example.ejemplojquerymobile.ActividadPrincipal"
   android:label="@string/app_name"
   android:configChanges="orientation|keyboardHidden">
```

11. Como en index.html hemos definido las referencias a jQuery Mobile en Internet, necesitas solicitar el permiso de acceso a Internet. Añade el siguiente permiso en Android Manifest:

```
<uses-permission android:name="android.permission.INTERNET" />
```

12. Ejecuta la aplicación.

Comprueba que la aplicación se ejecuta y se visualiza correctamente. Si no lo hace, es posible que ocurra porque el dispositivo en el que se ejecuta no esté conectado a Internet. Como hemos definido las referencias a jQuery Mobile en la web, es necesario que el dispositivo tenga conexión a Internet para que la aplicación se visualice correctamente. Deshabilita la conexión a Internet y vuelve a ejecutar la aplicación. Observa lo que ocurre.

Como la aplicación se ejecuta en un dispositivo, no es posible garantizar que siempre esté conectado a Internet. Para solucionarlo, podemos descargar jQuery Mobile y empaquetarlo en la aplicación.

Vamos a ver cómo hacerlo:

Ejercicio: Utilizar jQuery Mobile offline en una aplicación Android

1. Descarga la última versión de jQuery Mobile desde https://jquerymobile.com/download.

   En el momento de escribir este manual la última versión estable es la 1.4.5. Descargamos el fichero zip cuyo enlace está en el apartado ZIP FILE de la versión elegida.

2. Descomprime el fichero descargado en la carpeta assets del proyecto.

   Observa que entre el contenido descomprimido se encuentra la carpeta demos. En ella encontrarás diferentes ejemplos con su código fuente. Pero no aporta nada a nuestro proyecto. Lo que hace es aumentar el tamaño del apk. Por lo tanto procederemos a eliminarla.

   También necesitamos el fichero jQuery que no está incluido en la descarga. Concretamente, para la versión 1.4.5 de jQuery Mobile necesitamos la versión 1.11.x de JQuery.

3. Accede a jquery.com y en la sección DOWNLOAD, descarga la versión comprimida. Guarda el fichero en la carpeta assets del proyecto.

   La última versión en el momento de escribir este manual es 1.11.3. Tienes que descargar el fichero al cual se accede desde Download the compressed, production jQuery 1.11.3.

4. En el fichero index.html del proyecto, sustituye el contenido código de `<head>…</head>` por:

```
<head>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="jquery.mobile-1.4.5.min.css"/>
   <script src="jquery-1.11.3.min.js"></script>
   <script>
      $(document).bind('mobileinit',function(){
         $.mobile.changePage.defaults.changeHash = false;
         $.mobile.hashListeningEnabled = false;
         $.mobile.pushStateEnabled = false;
      });
   </script>
   <script src="jquery.mobile-1.4.5.min.js"></script>
   <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
</head>
```

   Hemos sustituido las referencias a jQuery Mobile en la web por las locales, que hacen referencia a los ficheros descargados.

5. Ejecuta la aplicación.

   Comprueba que funciona, tanto si el dispositivo tiene conexión a Internet como si no.

4.5.1.2. Añadir contenido

Dentro del `div` de contenido (`data-role="content"`), puedes añadir algunos elementos HTML estándar: encabezamientos, listas, párrafos… Puedes escribir tus propios estilos personalizados añadiendo hojas de estilo adicionales en el `head` después de la hoja de estilo de jQuery Mobile.

4.5.1.3. Crear una lista

jQuery Mobile incluye un conjunto de listviews que son codificados como listas HTML. Si añadimos el atributo `data-role="listview"`, transformaremos rápidamente una simple lista en una lista con estilo preparada para dispositivos táctiles. Vamos a hacer que la lista tenga un sangrado con `data-inset="true"` y añadir una búsqueda dinámica con `data-filter="true"`:

```
<ul data-role="listview" data-inset="true" data-filter="true">
   <li><a href="#">Samsung</a></li>
   <li><a href="#">Apple</a></li>
   <li><a href="#">Blackberry</a></li>
   <li><a href="#">HTC</a></li>
   <li><a href="#">Nokia</a></li>
</ul>
```

Ejercicio: Listas con jQuery Mobile

1. Añade el código anterior en index.html. Dentro del `div` de contenido, justo debajo de Contenido .
2. Ejecuta la aplicación y realiza una búsqueda.

Ejercicio: Sustituir TextView por WebView en FotografiasDrive

En este ejercicio mejoraremos la aplicación `Eventos` creada en la unidad anterior. Vamos a sustituir el `TextView` que listaba los nombres de las fotografías subidas a Google Drive por una lista compuesta por una miniatura de la fotografía y su nombre. Para ello utilizaremos el componente `ListView` de jQuery Mobile.

1. En la aplicación `Eventos`, sustituye el código del `TexView txtDisplay` en el layout `fotografias_drive.xml` por el siguiente:

```
<WebView
   android:id="@+id/display"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content" />
```

   Para utilizar jQuery Mobile en la aplicación, necesitamos hacerlo mediante un `WebView`.

2. Declara el `WebView` en la clase `FotografiasDrive` sustituyendo `public TextView mDisplay;` por:

```
static WebView mDisplay;
```

No declaramos el `WebView` en la actividad, ya que lo llamamos desde dentro del hilo que utilizamos para la llamada HTTP a Google Drive. Necesitaremos una programación un poco especial.

3. Inicializa el `WebView` sustituyendo `mDisplay = (TextView) findViewById(R.id.txtDisplay);`, en el método `onCreate` en la clase `FotografiasDrive`, por el código siguiente:

```
mDisplay = (WebView) findViewById(R.id.display);
mDisplay.getSettings().setJavaScriptEnabled(true);
mDisplay.getSettings().setBuiltInZoomControls(false);
mDisplay.loadUrl("file:///android_asset/fotografias.html");
```

4. Dentro de la carpeta `assets`, crea un fichero llamado `fotografias.html` con el siguiente código:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="user-scalable=no, width=device-width" />
<link rel="stylesheet"
   href=
   "http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css" />
<script src="fotografias.js"></script>
<script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
<script
   src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
</script>
</head>
<body>
   <div data-role="page">
      <div data-role="content">
         <ul id="lista" data-role="listview" data-inset="true"></ul>
      </div>
   </div>
</body>
</html>
```

Contiene la inicialización de jQuery Mobile y una página cuyo contenido es una lista vacía. La lista <ul>. . . no contiene ningún elemento <il>…</il>. Identificamos la lista como lista con `id=”lista”`.

5. Crea el fichero `fotografias.js` en la carpeta `assets`:

```
function add(fichero, imagen){
   var listItem = '<li><img src="'+imagen+'" heigh="150px">' +
fichero + '</li>';
   $("#lista").append(listItem);
   $('#lista').listview('refresh');
}
```

```
function vaciar(){
       $("#lista").empty();
       $('#lista').listview('refresh');
}
```

Contiene dos funciones:

- `add`: Añade a la lista un elemento `<li>…</li>`. Cada elemento está formado por su nombre fichero y una imagen. Utilizamos la variable `listItem` para concatenar una cadena que contiene el código HTML de cada elemento. Cada elemento contendrá una imagen `<img src="…">` de 150 píxeles de alto y el nombre del fichero. Utilizando el jQuery, añadimos el elemento a la lista con `$("#lista").append(listItem)` y luego refrescamos la lista con `$('#lista').listview('refresh')`.
- `vaciar`: Mediante el comando jQuery `$("#lista").empty()` vaciamos la lista para posteriormente refescar su contenido en pantalla.

6. Añade al final de la clase `FotografiasDrive`:

```
static void addItem(final Context context, final String fichero,
final String imagen) {
   carga.post(new Runnable() {
      public void run() {
         mDisplay.loadUrl(
         "javascript:add(\"" + fichero + "\",\"" + imagen + "\");");
      }
   });
}

static void vaciarLista(final Context context) {
   carga.post(new Runnable() {
      public void run() {
         mDisplay.loadUrl("javascript:vaciar()");
      }
   });
}
```

Estos dos métodos son los que nos van a permitir interactuar con el `WebView` desde dentro de un hilo. Simplemente son llamadas a las funciones declaradas en `funciones.js` con los parámetros correspondientes.

7. En la clase `FotografiasDrive` dentro del método `listarFicheros`, añade, debajo de `mostrarCarga(FotografiasDrive.this,"Listando archivos...")`, el siguiente código:

```
vaciarLista(getBaseContext());
```

Antes de añadir nuevos elementos a la lista, vacíala.

8. En el mismo método `listarFicheros`:

Sustituye:

```
mostrarTexto(getBaseContext(), fichero.getOriginalFilename());
```

por:

```
addItem(FotografiasDrive.this, fichero.getOriginalFilename(),
fichero.getThumbnailLink());
```

Por cada fichero almacenado en la carpeta donde hemos subido las fotografías, añadimos un elemento a la lista. Además de utilizar el atributo `getOriginalFileName()`, que nos proporciona el nombre del fichero, utiliza `getThumbnailLink()`, que nos facilitará la ruta a una miniatura de la fotografía.

9. Modifica la declaración del `BroadcastReceiver mHandleMessageReceiver`, sustituyéndola por:

```
private final BroadcastReceiver mHandleMessageReceiver = new
BroadcastReceiver() {
   @Override
   public void onReceive(Context context, Intent intent) {
      String nuevoMensaje = intent.getExtras().getString("mensaje");
   }
};
```

10. Ejecuta la aplicación y comprueba que se obtiene una lista con la imagen y el nombre de cada fotografía subida a Google Drive en cada evento.

4.5.1.4. Añadir un deslizador

El framework contiene un conjunto completo de elementos de formulario que automáticamente mejoran el estilo de los widgets para que sean amigables y táctiles. jQuery Mobile posee un deslizador HTML5 que permite la introducción de un valor perteneciente a un rango; no se necesita la etiqueta `data-role`. Asegúrate de introducirlo como un elemento de un formulario.

```
<form>
  <label for="slider-0">Valora de 0 a 100:</label>
  <input type="range" name="slider" id="slider-0" value="50"
         min="0" max="100"/>
</form>
```

Ejercicio: Deslizador con jQuery Mobile

1. Añade el código anterior en index.html, debajo del código de la lista anterior.
2. Ejecuta la aplicación y prueba el deslizador.

4.5.1.5. Crear un botón

jQuery Mobile nos permite convertir un enlace en un botón, así es más fácil hacer clic. Para hacerlo, añadimos el atributo `data-role="button"` a un enlace. Podemos añadir un icono con el atributo `data-icon` y opcionalmente establecer su posición con el atributo `data-iconpos`.

```
<a href="http://www.androidcurso.com" data-role="button" data-icon="star">
Curso Android</a>
```

Ejercicio: Botón con jQuery Mobile

1. Añade el código anterior en index.html, debajo del código del deslizador.
2. Ejecuta la aplicación y pulsa el botón.

4.5.1.6. Temas

jQuery Mobile tiene un robusto framework de temas que soporta 26 conjuntos de barras de herramientas, contenido y colores de botones, llamado swatch. Cada tema viene identificado por una letra, desde la **a** a la **z**.

Añade el atributo `data-theme` a los widgets de la página: page, header, list, slider, o button para aplicarles un tema. Por defecto, las letras **a, b** y **c** son las que contienen los temas. El resto de letras hasta la z son iguales que el tema **b**.

Podemos crear o personalizar temas con la aplicación web ThemeRoller accediendo a http://themeroller.jquerymobile.com. Crea un nuevo tema o modifica uno existente, bien modificando un tema por defecto o importándolo. Posteriormente, descarga el fichero zip con todos los ficheros necesario para utilizarlo, además de una página HTML de ejemplo.

Si realizas búsquedas por Internet, podrás encontrar temas gratuitos que puedes incorporar a la aplicación y dotarla de un aspecto espectacular.

Ejercicio: Temas conjQuery Mobile

1. Sustituye en index.html:

```
<div data-role="page"> <!-- página -->
```

por:

```
<div data-role="page" data-theme="b"> <!-- página -->
```

2. Ejecuta la aplicación.

Comprueba que el aspecto de la aplicación ha cambiado al variar el valor de `data-theme`.

Además de estas nociones básicas, jQuery Mobile y los demás frameworks preparados para el desarrollo web para dispositivos móviles permiten el uso de transiciones en las páginas y el uso de muchos más componentes. Si quieres ampliar la información, visita http://www.jquerymobile.com.

Vamos a cambiar la interfaz de la aplicación Eventos usando jQuery Mobile. Observarás que con solo modificar el fichero HTML dotaremosa a la aplicación de una nueva interfaz y de transiciones en el cambio de pantallas.

Ejercicio: Utilizar jQuery Mobile en la aplicación "Eventos"

1. En el proyecto Eventos, copia en la carpeta assets los ficheros necesarios para que la aplicación se ejecute offline, si no lo has hecho ya. Copia los ficheros de la carpeta public del proyecto de Firebase CLI. No elimines los ficheros que existen en assets.
2. Sustituye el contenido de la etiqueta `<head>` por el siguiente:

```
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="jquery.mobile-1.4.5.min.css"/>
   <script src="jquery-1.11.3.min.js"></script>
   <script>
      $(document).bind('mobileinit',function(){
         $.mobile.changePage.defaults.changeHash = false;
         $.mobile.hashListeningEnabled = false;
         $.mobile.pushStateEnabled = false;
      });
   </script>
   <script src="jquery.mobile-1.4.5.min.js"></script>
   <link   href="estilos.css" rel="stylesheet" type="text/css">
   <script src="funciones.js"></script>
</head>
```

3. Ejecuta la aplicación.

   Observa que la aplicación se ejecuta perfectamente y que ciertos objetos, como los botones, han adquirido el aspecto de una aplicación jQuery Mobile. Seguidamente vamos a cambiar el aspecto de la página de inicio.

4. Reemplaza el código de la etiqueta `<body>`por:

```
<body>
   <div data-role="page" data-theme="b" id="pagina1">
      <div data-role="header">
         <a href="#" data-icon="home" onClick="volver();">Inicio</a>
         <h1 id="evento"></h1>
         <a href="#pagina2" data-role="button" data-icon="info"
                                    data-transition="popup">+ Info</a>
      </div>
      <div data-role="content">
         <p id="wiki"><p>
      </div>
      <div data-role="footer" data-position="fixed">
         <h1>DEU UPV</h1>
      </div>
   </div>
</body>
```

Hemos sustituido la etiqueta `body`, que mostraba el contenido de la página, por otra con atributos jQuery Mobile. Mediante el atributo `data-role="page"` indicamos que es una página. Con `data-theme="b"` hacemos que el tema jQuery Mobile utilizado sea el **b**. Y mediante `id="pagina1"`, identificamos la página en el documento.

Dentro de la página, observamos que hay tres etiquetas `div` con distintos atributos `data-role`: `header`, `content` y `footer`. Con ellos indicamos que se trata de la cabecera, contenido y pie de la página, respectivamente.

En la cabecera (`header`) hemos incluido el nombre del evento con la etiqueta `<h1>` y un botón con la etiqueta `<a>` para cerrar la actividad.

En el contenido (`content`) hemos añadido un párrafo `<p>` donde mostraremos la descripción del evento y un botón `<a>` que mostrará otra pantalla con información del curso. Un botón en jQuery Mobile se declara como un enlace en HTML con la etiqueta `<a>`. Con el atributo `href` indicamos qué página se va a mostrar al pulsar el botón. Con `data-role="button"` declaramos que es un botón. Utilizamos `data-transition="popup"` para indicar qué tipo de transición se tiene que realizar al mostrar la nueva página; en nuestro caso, popup.

En el pie de la página (`footer`) hacemos que se muestre al pie de la pantalla mediante el atributo `data-position="fixed"`. Si no se mostraría justo debajo de `content`. Si `content` ocupara la mitad de la pantalla, se mostraría a mitad. En el pie mostramos el texto “DEU UPV”.

Seguidamente vamos a crear la página de “+ Info curso”.

5. Inserta antes de la etiqueta `</body>` el siguiente código:

```
<div data-role="panel" data-theme="b" id="pagina2">
   <h3>Info</h3>
   <p>Curso DEU UPV</p>
   <p>Master UPV</p>
   <p>Curso Firebase UPV</p>
```

```
   <a href="#pagina2" data-role="button" data-icon="back"
                                    data-transition="overlay">Volver</a>
</div>
```

Hemos añadido una nueva página `pagina2`. Es llamada desde el botón **+ Info curso** de `pagina1`. La llamada se realiza con el atributo `href="pagina2"` del botón.

En este caso, hemos cambiado una página por un panel. Al pulsar sobre el botón se abrirá desde la derecha. Definimos el panel con `data-role="panel"` y lo identificamos con `id="pagina2"`. El identificador lo utilizaremos para realizar la llamada para abrirlo o cerrarlo. Dentro del `div` del panel insertamos el contenido del mismo, en nuestro caso, la información de los cursos donde se imparte esta unidad. Además hemos incluido un botón para cerrarlo.

Para abrir o cerrar un panel realizamos la misma llamada desde un botón. Recordemos que definimos un botón mediante una etiqueta HTML de enlace `<a>` y `data-role="button"`. Con el atributo `href` indicamos qué elemento vamos a llamar; en nuestro caso, el panel de información. Para ello definimos el atributo como `href="#pagina2"`. Adicionalmente, con el atributo `data-display="overlay"`, indicamos la transición a utilizar para abrir o cerrar el panel.

Si no utilizamos jQuery Mobile tenemos que controlar la visibilidad de cada pantalla. Al utilizar jQuery Mobile, esto ya no es necesario. jQuery Mobile solo muestra una página, permite transiciones al cambiar de página; no es necesario que lo programemos.

6. Ejecuta la aplicación y podrás observar que rápidamente hemos mejorado la aplicación mediante el uso de frameworks para desarrollo web en dispositivos móviles.

Preguntas de repaso: Alternativas a la programación independiente de la plataforma para móviles

4.6. Firebase Analytics

4.6.1. Introducción

Firebase Analytics permite monitorizar aplicaciones para proporcionar estadísticas de uso y la participación de los usuarios. Es uno de los componentes fundamentales de Firebase. Su utilización es ilimitada y gratuita para la generación de informes sobre un máximo de 500 eventos diferentes. Los informes de Analytics ayudan a entrender cómo se comportan los usuarios de una aplicación y así tomar decisiones de marketing y de optimización de rendimiento.

Las funciones clave de Analytics son:

- **Generación de informes:** Ofrece informes ilimitados de hasta 500 eventos distintos.

- **Segmentación del público:** Podemos definir segmentos de público (conjunto de usuarios). Por ejemplo, en función de: el dispositivo que usa, eventos personalizados, propiedades del usuario... Los segmentos pueden usarse en otros servicios de Firebase, como por ejemplo en notificaciones.

El SDK Firebase Analytics captura automáticamente eventos (permite eventos personalizados) y propiedades de usuario. Consulta los datos capturados en el panel de control de la consola de Firebase. Las estadísticas de los datos van desde resúmenes de datos, como los usuarios activos y los grupos demográficos a los que pertenecen, hasta información más detallada, como los artículos más comprados.

Analytics también se integra con otras características de Firebase. Por ejemplo, registra automáticamente los eventos que corresponden a las notificaciones enviadas desde la consola de Firebase y proporciona informes sobre el impacto de cada campaña. Ayuda a entender cómo se comportan los usuarios para poder tomar decisiones sobre cómo comercializar una aplicación.

Firebase Analytics se integra con otros servicios:

- **BigQuery:** permite realizar análisis personalizados de todo el conjunto de datos de Analytics y también importar datos de otras fuentes.
- **Firebase Crash Reporting:** registra eventos de cada error de bloqueo de la aplicación. Guarda información asociada, como la versión de la aplicación o zona geográfica donde estaba el usuario cuando se ha producido el error. Se pueden crear públicos con los usuarios afectados por varios bloqueos y responder con notificaciones dirigidas a ese público.
- **FCM:** registra automáticamente los eventos de las notificaciones enviadas desde la consola de Firebase. Permite generar informes sobre el impacto de cada campaña.
- **Firebase Remote Config:** cambia el comportamiento y el aspecto de la aplicación, según el público, sin distribuir distintas versiones de la aplicación.
- **Google Tag Manager:** si integramos Google Tag Manager con Firebase Analytics, podemos administrar Analytics de forma remota desde una interfaz web.

Para comenzar a utilizar Analytics en una aplicación Android, se debe agregar el SDK de Firebase a la aplicación para que la recopilación de datos comience automáticamente. Podremos consultar los resultados en la consola de Firebase al cabo de unas horas. Podemos definir eventos y públicos personalizados. Usaremos los datos para orientar mensajes, promociones o nuevas funciones de la aplicación a distintos públicos mediante otros componentes de Firebase, como FCM y Remote Config.

4.6.2. Analytics en Android

Firebase Analytics recopila datos de uso y comportamiento de una aplicación Android. Registra dos tipos de información:

- **Eventos:** qué está sucediendo, como acciones del usuario, eventos del sistema o errores.
- **Propiedades del usuario:** atributos de usuarios, como preferencia de idioma o ubicación geográfica.

Analytics registra automáticamente algunos eventos y propiedades de usuario; no necesitamos agregar ningún código para habilitarlos.

4.6.2.1. Eventos

Un evento es una ocurrencia importante en la aplicación a monitorizar. Podemos monitorizar hasta 500 tipos diferentes de eventos por aplicación. Tampoco podemos asociar más de 25 parámetros para cada evento. Una aplicación puede contenter eventos comunes o eventos personalizados. Cada evento es definido con un nombre único.

Vamos a ver en un ejercicio cómo utilizar Firebase Analytics en una aplicación Android.

Ejercicio: Eventos Firebase Analytics en Android

1. Agrega la dependencia de Firebase Analytics en el archivo build.gradle (Module:app).

```
compile 'com.google.firebase:firebase-core:11.8.0
```

2. Declara un objeto Firebase Analytics en la clase `Comun`:

```
static FirebaseAnalytics mFirebaseAnalytics;
```

3. Inicializa el objeto `Analytics` insertando el siguiente código al final del método `onCreate` en la clase `ActividadPrincipal`:

```
mFirebaseAnalytics = FirebaseAnalytics.getInstance(this);
```

Una vez creada una instancia de `FirebaseAnalytics`, podemos registrar eventos predefinidos o personalizados con el método `logEvent()`.

Vamos a implementar un evento `SELECT_CONTENT` cuando un usuario acceda al menú **SUSCRIPCIONES**, que se encuentra en la actividad `ActividadPrincipal`.

4. Sustituye el método `onOptionsItemSelected` de la clase `ActividadPrincipal` por el siguiente:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   int id = item.getItemId();
   if (id == R.id.action_temas) {
      Bundle bundle = new Bundle();
      bundle.putString(FirebaseAnalytics.Param.ITEM_NAME,
                                                    "suscripciones");
```

```
        mFirebaseAnalytics.logEvent("menus", bundle);
        Intent intent = new Intent(getBaseContext(), Temas.class);
        startActivity(intent);
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

El nuevo código nos va a permitir capturar un evento de Firebase Analytics. Llamamos al método `logEvent` del objeto `FirebaseAnalytics` para contabilizar el evento. `logEvent` necesita dos parámetros:

- **nombre:** El nombre del evento. Debe contener de 1 a 40 caracteres alfanuméricos o guiones bajos. Algunos nombres de eventos están reservados. Consulta la lista de nombres de eventos reservados. Los prefijos "firebase_", "google_" y "ga_" están reservados y no deben utilizarse. Los nombres de los eventos distinguen entre mayúsculas y minúsculas.
- **parámetros:** El mapa de los parámetros del evento. Si tiene un valor nulo, indica que el evento no tiene parámetros. Los nombres de los parámetros pueden tener hasta 40 caracteres de longitud, deben comenzar con un carácter alfabético y contener solo caracteres alfanuméricos o guiones bajos. Los tipos de datos compatibles son: `string`, `long` y `double`. Los valores del parámetro de cadena pueden tener hasta 100 caracteres de longitud. Los prefijos "firebase_", "google_" y "ga_" están reservados y no deben usarse para nombres de parámetros.

El código registra un evento cuando el usuario elige un menú. Para ello hemos asociado un evento a **SUSCRIPCIONES**. El nombre del evento es “menu” y como parámetro le pasamos un objeto `Bundle` con el nombre del menú.

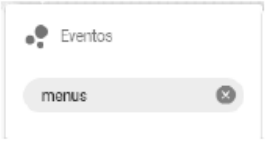
5. Para monitorizar el resto de menús, sustituye el método de la clase `EventoDetalles` por:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   View vista = (View) findViewById(android.R.id.content);
   Bundle bundle = new Bundle();
   int id = item.getItemId();
   switch (id) {
      case R.id.action_putData:
         bundle.putString(FirebaseAnalytics.Param.ITEM_NAME, "subir_imagen");
         mFirebaseAnalytics.logEvent("menus", bundle);
         subirAFirebaseStorage(SOLICITUD_SUBIR_PUTDATA,null);
         break;
      case R.id.action_streamData:
         bundle.putString(FirebaseAnalytics.Param.ITEM_NAME,
                                                    "subir_stream");
         mFirebaseAnalytics.logEvent("menus", bundle);
         seleccionarFotografiaDispositivo(vista,
                                          SOLICITUD_SELECCION_STREAM);
```

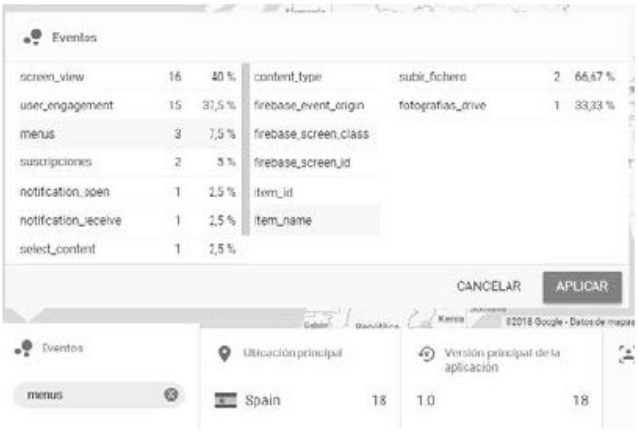
```
            break;
        case R.id.action_putFile:
            bundle.putString(FirebaseAnalytics.Param.ITEM_NAME,
                                                    "subir_fichero");
            mFirebaseAnalytics.logEvent("menus", bundle);
            seleccionarFotografiaDispositivo(vista,
                                            SOLICITUD_SELECCION_PUTFILE);
            break;
        case R.id.action_getFile:
            bundle.putString(FirebaseAnalytics.Param.ITEM_NAME,
                                                   "descargar_fichero");
            mFirebaseAnalytics.logEvent("menus", bundle);
            descargarDeFirebaseStorage(evento);
            break;
        case R.id.action_fotografiasDrive:
            bundle.putString(FirebaseAnalytics.Param.ITEM_NAME,
                                                   "fotografias_drive");
            mFirebaseAnalytics.logEvent("menus", bundle);
            Intent intent = new Intent(getBaseContext(),
                                                FotografiasDrive.class);
             intent.putExtra("evento", evento);
             startActivity(intent);
             break;
         case R.id.action_acercaDe:
            bundle.putString(FirebaseAnalytics.Param.ITEM_NAME,
                                                        "acerca_de");
            mFirebaseAnalytics.logEvent("menus", bundle);
            Intent intentWeb = new Intent(getBaseContext(),
                                                  EventosWeb.class);
            intentWeb.putExtra("evento", evento);
            startActivity(intentWeb);
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

6. Ejecuta la aplicación y selecciona cualquier menú de la pantalla principal o de un evento concreto.
7. Accede a la consola de Firebase y selecciona “StreamView” en el apartado ANALYTICS.
8. Asegurate de tener seleccionado **Eventos**.

9. Pulsa en **Eventos**.



10. Localiza el evento “menus”, en el desplegable selecciona “item_name”. Observa que aparece la cantidad de veces que has elegido cada menú en la aplicación.



Puede que los datos tarden un poco en aparecer. Comprueba que apareces geolocalizado en el mapa que mostrará StreamView.

4.6.2.2. Propiedades de usuario

Las propiedades del usuario describen a los usuarios de una aplicación. Son atributos que utilizamos para crear segmentos, como el idioma o la ubicación geográfica. Permiten analizar los comportamientos de diversos segmentos, podemos aplicar las propiedades como filtros en informes.

Analytics registra automáticamente algunas propiedades de usuario. Si necesitamos recopilar datos adicionales, podemos configurar hasta 25 propiedades de usuario personalizadas. Los nombres de las propiedades distinguen mayúsculas de minúsculas.

Para configurar una propiedad de usuario, debemos registrarla en la consola de Firebase. Cada usuario le otorgará un valor desde la aplicación Android. Para ello, nos valdremos de una instancia de `FirebaseAnalytics` y del método `setUserProperty`, al cual pasaremos como parámetros el nombre de la propiedad de usuario y el valor.

Ejercicio: Propiedades de usuario en Firebase Analytics

1. Accede a la consola de Firebase y selecciona **Propiedades de usuario** en el apartado ANALYTICS.
2. Pulsa el botón **Nueva propiedad de usuario**.
3. Proporciona un nombre y una breve descripción:

   Nombre de la propiedad de usuario: evento_detalle

   Descripción: Usuario interesado en un evento

4. Pulsa el botón **Crear**.
5. Añade el siguiente código al final del método `onCreate` de la clase `EventoDetalles`:

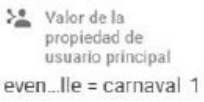
```
mFirebaseAnalytics.setUserProperty("evento_detalle", evento);
```

6. Ejecuta la aplicación.

   Accede a los detalles de algún evento. Comprueba que aparecen los datos de la propiedad de usuario en la consola de Firebase.

7. Accede a la consola de Firebase y selecciona **StreamView** en el apartado ANALYTICS.
8. Asegúrate de tener seleccionado **Usuarios**.

9. Pulsa en Valor de la propiedad de usuario principal.



10. Comprueba que aparece la propiedad “evento_detalle” y, al pulsar sobre ella, el nombre del primer evento que has pulsado tras añadir la propiedad.

11. Selecciona Notificaciones en el apartado GROW.
12. Duplica la última notificación que hayas enviado.
13. En el apartado DESTINO selecciona **Segmento de usuarios**, elige App con el valor `org.example.eventos` y añade la “Propiedad de usuario” de usuario “evento_detalle” que contenga el valor “todos”.
14. Pulsa el botón **Enviar mensaje**.

Comprueba que no recibes ninguna notificación. No hay ningún usuario con ese valor en la propiedad de usuario “evento_detalle”.

15. Duplica la última notifiación y cambia el valor de “evento_detalle” al nombre del evento que aparece en el apartado STREAMVIEW.

16. Pulsa el botón **Enviar mensaje**.

Comprueba que se recibe la notificación en el dispositivo.

4.6.3. Panel de control de Analytics

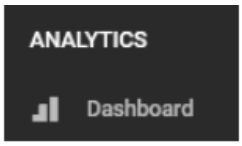
Además de definir eventos y propiedades de usuario, también tienes que saber cómo funciona el panel de control de Analytics en la consola de Firebase.

Para ver el panel de control es recomendable verlo con datos, la aplicación que estamos desarrollando, no tiene muchos. Se puede acceder a un proyecto demostración con datos desde la consola de Firebase. Accedemos a la consola y seleccionamos Explorar un proyecto de demostración.

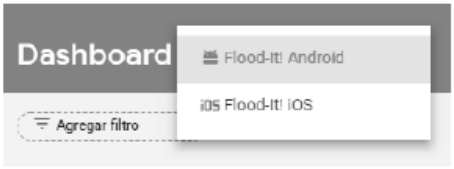
Podremos acceder al proyecto “Flood-It!” y ver sus estadísticas en Firebase Analytics. No podremos hacer cambios, ya que tenemos acceso de solo lectura.

“Flood-It” es un juego que se puede encontrar en Play Store. Para entender mejor los datos, instálalo y prueba a jugar unas partidas. Seguiremos la explicación del panel de control de Firebase Analytics con el proyecto de demostración.

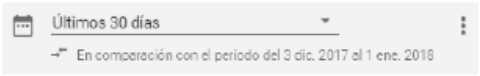
Para acceder al panel de control de Firebase Analytics seleccionamos **Dashboard** en el apartado ANALYTICS.



Un proyecto puede tener varias aplicaciones. En nuestro caso tenemos dos, una para Android y otra para iOS. Para seleccionar una u otra, lo haremos desde el desplegable situado a la derecha de **Dashboard**"



Por defecto, los datos mostrados son los de los últimos 30 días. Se puede cambiar el rango del periodo pulsando en el desplegable de periodo en la parte superior derecha de la pantalla. Cuando cambiemos el periodo a evaluar, provocaremos que todos los datos de la pantalla se cambien para mostrar dicho periodo.

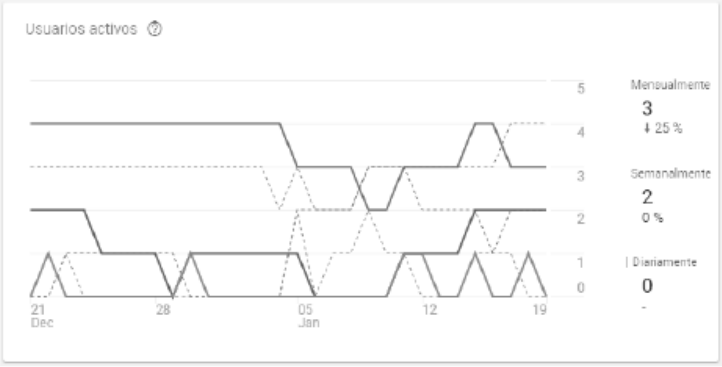


Filtramos contenido mediante Agregar filtro.

Podemos filtrar contenido mediante dos tipos de filtros “Nombre de la audiencia” y “Propiedad del usuario”. Se pueden utilizar los valores predeterminados de cada filtro o bien definir unos personalizados.

Vamos a ver las diferentes secciones del panel de control de Analytics en la consola de Firebase:

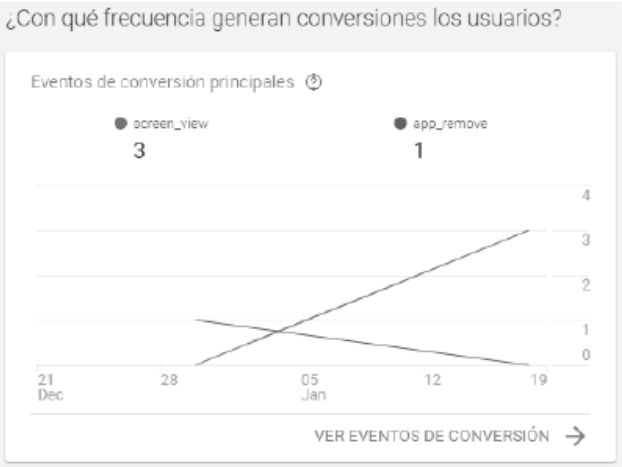
Usuarios activos: Muestra los usuarios activos durante un periodo de 1 día, 1 semana y 1 mes, representados a lo largo del tiempo. Movemos el puntero del ratón por el gráfico para ver los datos de usurio en un día concreto. Los valores de resumen a la derecha muestran la cantidad de usuarios activos en el periodo.



En tiempo real: muestra la cantidad de usuarios que han usado la aplicación en los últimos 30 minutos. El gráfico de barras indica los usuarios activos por minuto. Podemos mover el puntero del ratón sobre una barra para ver la cantidad

de usuarios registrados conectados para el minuto indicado. Para obtener más detalles sobre la actividad actual, pulsamos sobre **Streamview.**

Principales eventos de conversión: Los gráficos muestran los tres principales eventos de conversión recientes representados a lo largo del tiempo. Los valores de resumen (que se encuentran en la parte superior) muestran la cantidad de eventos y el porcentaje de incremento para el periodo de tiempo seleccionado. **NOTA:** No se muestran los eventos `first_open` y `session_start`.



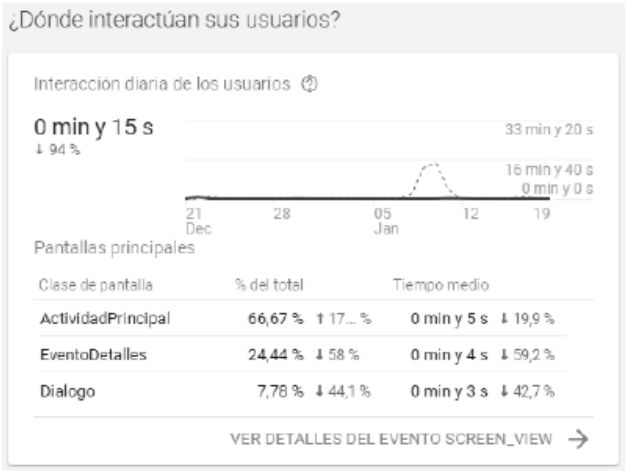
Interacción de los usuarios:

Interacción diaria: el valor del tiempo representa la suma del tiempo de participación de todos los usuarios en el periodo. El valor porcentual representa el aumento o la disminución en la participación respecto del periodo de comparación. La línea continua en el gráfico muestra las tendencias de participación diarias totales para el periodo de tiempo indicado, mientras que la línea de puntos representa el periodo de comparación.

Interacción diaria por usuario: el valor del tiempo representa la duración promedio de la interacción de los usuarios en el periodo. El valor porcentual muestra el aumento o la disminución en la participación diaria por usuario respecto del periodo de comparación. La línea continua en el gráfico muestra la participación diaria por usuario durante el periodo de tiempo indicado, mientras que la línea de puntos representa el periodo de comparación.

Sesiones por usuario: el valor numérico representa la cantidad total de sesiones dividida por la cantidad de usuarios activos del periodo. El valor porcentual representa el aumento o la disminución en la cantidad de sesiones por usuario en el periodo de comparación. La línea continua en el gráfico representa el cambio en las sesiones por usuario durante el periodo de tiempo indicado, mientras que la línea de puntos representa el periodo de comparación.

Usaremos el menú para seleccionar los datos de participación del usuario por nombre de pantalla o clase de pantalla.



Total de ingresos:

Ingresos totales: muestra el valor combinado de todas las fuentes de ingresos, incluidos los estimados de la red de AdMob. El valor porcentual indica el aumento o la disminución para el periodo de tiempo seleccionado. El gráfico muestra las tendencias de ingresos para el periodo de tiempo seleccionado.

Fuentes de ingresos: los ingresos por compras se combinan para todos los eventos ecommerce_purchase e in_app_purchase. Los ingresos de AdMob son ingresos estimados.

Ingresos por usuario: incluye el ingreso promedio por usuario (ARPU) y el ingreso promedio por usuario que paga (ARPPU) para el periodo de tiempo seleccionado. El ARPU incluye los ingresos estimados de la red de AdMob.

Los ingresos son la suma de los valores correspondientes a los eventos ecommerce_purchase y in_app_purchase. ARPU es el ingreso dividido por el total de usuarios para los periodos de tiempo dados. ARPPU es el ingreso dividido entre los usuarios que compraron durante los periodos de tiempo dados.

NOTA: Los ingresos informados pueden diferir de los valores que se muestran en la consola de Google Play.

Estabilidad: Es el porcentaje de usuarios que no experimentaron un bloqueo. Utiliza los datos de Firebase Crashlytics y Firebase Crash Reporting.

Versión más reciente: El gráfico muestra el porcentaje de usuarios activos para cada versión de la aplicación en el periodo seleccionado.

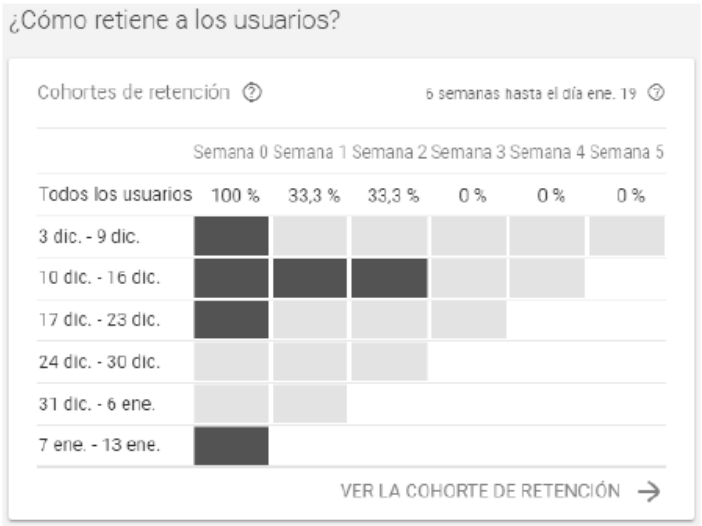
La tabla de versiones más recientes muestra una lista de las últimas versiones lanzadas para la aplicación y también indica el porcentaje de usuarios activos y de usuarios que no experimentaron bloqueos para cada versión.

Adquisición: Muestra, según el origen del usuario, la cantidad de veces que se abrió la aplicación por primera vez y el valor del ciclo de vida del cliente (LTV) correspondiente a los usuarios que la abrieron. Se muestran sus cinco principales redes publicitarias (fuentes).

El ciclo de vida de un cliente (LTV) es el valor neto de los ingresos que un cliente nos genera durante el tiempo que el cliente es cliente nuestro. Hay que tener en cuenta que el cálculo que obtenemos es una previsión.

Cohortes de retención: Una cohorte es un conjunto de usuarios que comenzaron a utilizar la aplicación al mismo tiempo (por ejemplo, el mismo día o durante la misma semana). El gráfico muestra el nivel de retención de los usuarios.

Cada fila representa una cohorte. La fila inferior representa la cohorte más reciente. La fila superior representa la cohorte más antigua. El sombreado más oscuro indica que un mayor porcentaje de usuarios de la cohorte volvió a utilizar la aplicación nuevamente. Podemos observar que el sombreado se vuelve más claro con el tiempo a medida que menos usuarios de cada cohorte vuelven a utilizar la aplicación.



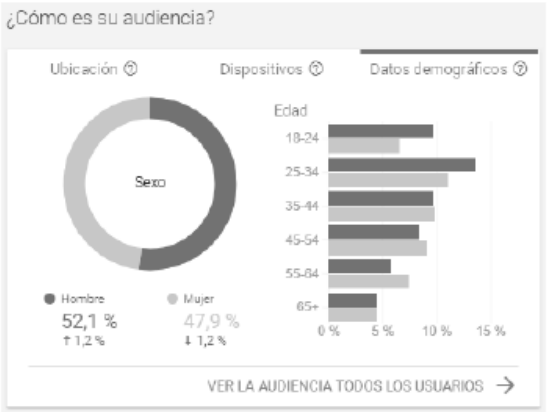
Audiencia:

La tarjeta AUDIENCIA incluye tres pestañas: **Ubicación**, **Dispositivos** y **Datos demográficos**:

Ubicación: mapa y tabla que indican el porcentaje de sesiones de cada uno de los principales países.

Dispositivos: "Modelo" indica el porcentaje de usuarios que utiliza cada uno de los tres principales modelos de dispositivos. Si existen más modelos, se englobarían en el item "Otro". "Versión del SO" indica el porcentaje de usuarios en cada una de las tres versiones principales del SO, si hay más versiones, se englobarían en el item "Otra".

Datos demográficos: el porcentaje de usuarios masculinos y femeninos, por edad.



4.6.4. StreamView

StreamView permite ver en vivo los eventos de Analytics. Los informes de StreamView están optimizados para conocer más detalles sobre los usuarios de la aplicación y qué hacen en tiempo real. Se enfoca a los eventos recopilados durante los últimos 30 minutos.

No se necesita realizar una configuración especial para que una aplicación use StreamView, solo debe integrar el SDK de Firebase.

Para acceder a los informes de StreamView, hacemos clic en StreamView en la consola de Firebase.

Veamos las diferentes vistas que tenemos en StreamView:

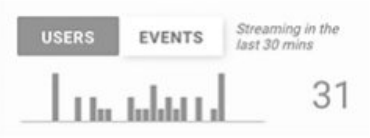
Vista geográfica

Pantalla que muestra el mapa con puntos activos donde los dispositivos registran eventos.

La vista predeterminada es una vista geográfica de la localización de los usuarios. Veremos los usuarios únicos cuyos dispositivos registraron eventos durante los últimos 30 minutos.

Alternar el tipo de métrica

Pantalla que muestra en primer plano la vista para cambiar el tipo de métrica. El botón alterna entre las métricas de **Usuarios** y **Eventos**.



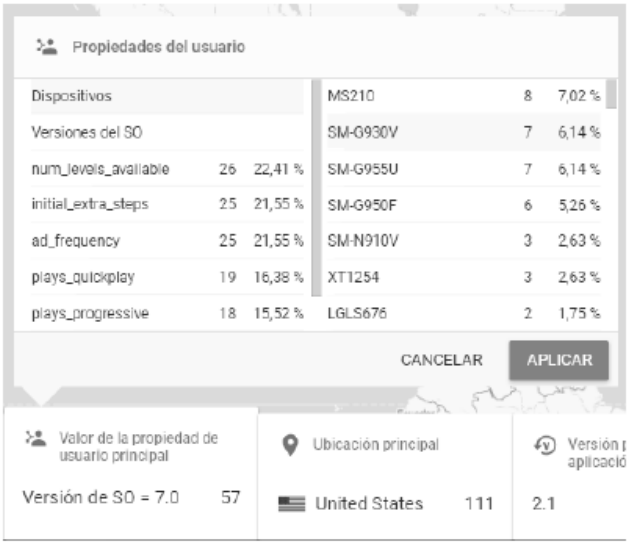
Si seleccionamos **Usuarios**, los informes mostrarán métricas relacionadas con los usuarios que estuvieron activos en los últimos 30 minutos. Si seleccionamos **Eventos**, se reflejarán los eventos que se registraron en los últimos 30 minutos.

Marquesina

A lo largo de la parte inferior de la pantalla hay una marquesina que tiene dos funciones. De forma predeterminada, la marquesina muestra las dimensiones principales de **Usuarios** o **Eventos**. Podemos hacer clic en cada dimensión para ver la información detallada de varios atributos.

Filtros

Se pueden aplicar filtros para responder preguntas, como "¿cuáles son los términos de búsqueda más populares en la aplicación en un país?". Para aplicar un filtro, hacemos clic en una dimensión de la marquesina. Luego, seleccionamos un valor (como por ejemplo: país o nombre de un evento) y clicamos en **Aplicar**. Esto creará un filtro en el resto de los informes de StreamView.



Detalles del cronograma

Podemos expandir un gráfico de barras de **Usuarios** o **Eventos**. Hacemos clic en el botón **Detalles del cronograma**, situado arriba a la izquierda de la marquesina.

El gráfico muestra la fluctuación del volumen de **Usuarios** o **Eventos** cada minuto en los últimos 30 minutos. Si nos desplazamos sobre cada barra, veremos un resumen de la información de ese minuto.

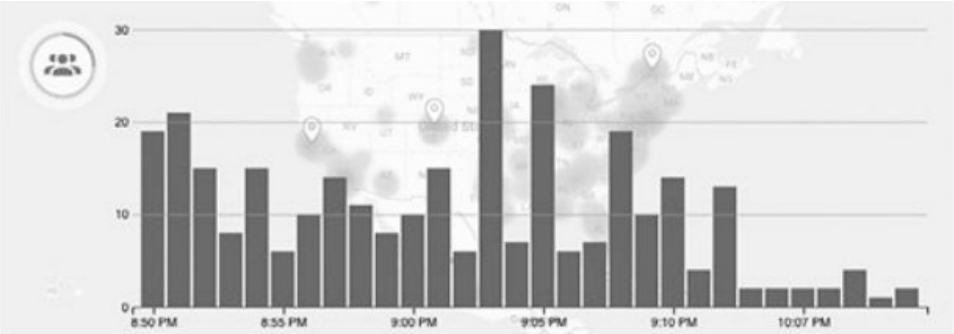


Tabla de tendencias

La “Tabla de tendencias” es una tabla con las propiedades de usuario o eventos principales registrados durante los últimos 30 minutos. Hacemos clic en el botón **Tendencias** situado arriba a la izquierda de la marquesina. Clicamos en el botón **Maximizar** para presentar una vista dinámica en pantalla completa. Permite crear una selección de las métricas más reveladoras, como los términos de búsqueda o los productos más populares en los últimos 30 minutos.

Instantáneas de usuario

Instantáneas de usuario son un informe detallado sobre un usuario. Permite ver la secuencia de eventos que registró un usuario individual en los últimos 30 minutos. La secuencia y el cronograma se muestran como un relato, lo que permite comprender de mejor manera cómo los usuarios de ciertos países, y en versiones específicas, navegan por la aplicación y cambian de una pantalla a otra. Podemos hacer clic en cada evento para examinar los parámetros y las propiedades de usuario que incluye cada uno, lo que nos permite hacer un análisis bastante profundo y ofrecer estadísticas que solo son posibles con una resolución detallada.

Antes de entrar a este modo, es recomendable seleccionar en el país y la versión de la aplicación para filtrar a los usuarios. Una vez que accedemos a Instantáneas de usuario, podremos usar las flechas para alternar entre diferentes usuarios a fin de realizar comparaciones rápidas.

4.6.4.1. Funnels

Usaremos embudos (Funnels) para visualizar y optimizar la tasa de finalización de una serie de pasos (eventos). Por ejemplo, podemos crear un embudo que contenga los pasos necesarios para crear una cuenta, y luego ver la tasa de finalización de cada paso. Podemos detectar un paso del proceso de creación de una cuenta donde los usuarios abandonan. Podemos filtrar los informes y así detectar qué segmento de usuarios completan los embudos a un ritmo mayor. Obtenemos información de un Funnel y luego realizamos cambios en la aplicación para mejorar la retención.

Funnels no es restrictivo en los pasos, lo que significa que los usuarios no tienen que completar un paso anterior (por ejemplo, el Paso 1) para ser incluidos en las métricas para un paso posterior (por ejemplo, el Paso 2).

Utilizamos Funnels para secuencias estrictas de eventos que ocurren naturalmente dentro de la aplicación.

Ejercicio: Crear Funnels

1. Accede a la consola de Firebase y selecciona **Eventos**.
2. Selecciona la opción **Funnels** en el apartado "ANALYTICS".
3. Pulsa en el botón **Agregue su primer embudo de conversión**.
4. Introduce el nombre y una descripción del embudo:

   Nombre: Notificacion

   Descripción: Recibir y abrir una notificación

5. Selecciona los dos primeros eventos como pasos en el embudo: notification_receive y notification_open.
6. Pulsa en el botón **Crear y ver**.
7. Duplica la última notificación enviada.
8. Ejecuta la aplicación “Eventos” y ponla en segundo plano.
9. Envía la notificación.

Comprueba que al recibir y abrir una notificación aparece en las estadísticas del embudo que has creado el mismo valor en los eventos notification_reveive y notification_open. Envia una nueva notificación, pero no la abras. Comprueba que aparece un valor superior en notification_receive.

Preguntas de repaso: Firebase Analytics

CAPÍTULO 5.

Firebase Functions, Enlaces dinámicos, Stability y más

VICENTE CARBONELL

En este capítulo vamos a seguir viendo características de Firebase. Controlaremos los eventos producidos en los servidores de Firebase mediante el uso de funciones con Firebase Functions. Aprenderemos a variar el comportamiento y el aspecto de una aplicación de forma remota con Firebase Remote Config. Conoceremos qué es y cómo utilizar enlaces dinámicos y enviar invitaciones mediante Firebase en Android con Firebase Dynamic Links. Veremos cómo otorgar estabilidad a una aplicación mediante Firebase. Para ello, veremos los servicios: Crashlitycs (para capturar errores de la aplicación que se producen en los dispositivos de los usuarios), Performance (evaluaremos el rendimiento de la aplicación en distintos puntos para mejorarlo) y Test Lab (nos permitirá realizar pruebas de una aplicación en dispositivos reales con distintas configuraciones).

Por último, aprenderemos a usar el servicio de copia de seguridad con Android Backup Service. Imaginemos que para que nuestra aplicación funcione, le pedimos al usuario una serie de datos o que el funcionamiento de la misma genere datos que vamos guardando de forma local en nuestro dispositivo. Si por cualquier motivo, el usuario hiciera un reset de fábrica, cambiara de dispositivo, etc., perdería todos los datos recopilados por nuestra aplicación, lo que sería muy frustrante para el usuario. Google ofrece un servicio para guardar todos estos datos en la nube de forma transparente, de tal forma que, al volver a instalar la aplicación, los tendríamos disponibles.

Objetivos

- Ejecutar funciones en los servidores de Google que son activadas por medio de ciertos eventos.
- Utilizar Firebase Remote Config en aplicaciones Android para variar su aspecto o comportamiento de forma remota.
- Compartir nuestra aplicación o contenido de la misma utilizando enlaces dinámicos.
- Capturar errores en los dispositivos de los usuarios con Firebase Crashlitycs.
- Evaluar el rendimiento de una aplicación con Firebase Performance.
- Realizar pruebas de una aplicación en dispositivos reales con Firebase Test Lab.
- Crear copias de seguridad en línea usando Android Backup Service.

5.1. Cloud Functions

5.1.1. Introducción

Cloud Functions para Firebase permite ejecutar automáticamente código JavaScript en los servidores de Google en respuesta a eventos en los servidores de Firebase. Los eventos son activados por desencadenadores de Firebase y solicitudes HTTP/S. El código a ejecutar se almacena en la nube de Google y se ejecuta en un entorno administrado.

Los distintos tipos de eventos que dispararán los desencadenadores y ejecutarán las funciones de Firebase son eventos de Cloud Firestore, Realtime Database, Firebase Authentication, Google Analytics para Firebase, Cloud Storage, Pub/Sub de Cloud o llamadas HTTP/S.

Implementamos el código en los servidores de Firebase desde una consola en tu ordenador.

Firebase gestiona los recursos de servidor de forma automática según los patrones de uso de los usuarios. No tendremos que preocuparnos de las credenciales, la configuración, mantenimiento o aprovisionamiento de servidores.

En muchos casos, los programadores prefieren ejecutar parte de la lógica de la aplicación en un servidor para evitar alteraciones del código en la parte cliente y liberar al dispositivo cliente de consumo de recursos. Así, aunque se aplicara ingeniería inversa a la aplicación, parte del código estaría protegido por estar en el servidor. El código a ejecutar en Cloud Functions está completamente aislado del cliente.

Después de implementar una función, los servidores de Google comienzan a administrarla. Detectan eventos y ejecutan la función cuando se activa. A medida que la carga aumenta o disminuye, la respuesta de Google es escalar con rapidez la cantidad de instancias del servidor virtual necesarias para ejecutar la función.

El ciclo de vida de una función es:

1. Se implementa una nueva función seleccionando un proveedor de eventos y se definen las condiciones según las cuales debe ejecutarse la función.
2. Firebase la conecta al proveedor de eventos seleccionado.
3. Cuando el proveedor de eventos genera un evento que coincide con las condiciones de la función, se invoca el código.
4. Si la función está ocupada con muchos eventos, Google crea más instancias para controlar el trabajo con más rapidez. Si la función está inactiva, las instancias se borran.
5. Cuando se actualiza una función, todas las instancias de la versión antigua se borran y se reemplazan por instancias nuevas.
6. Cuando un programador borra la función, se borran todas las instancias y se quita la conexión entre la función y el proveedor de eventos.

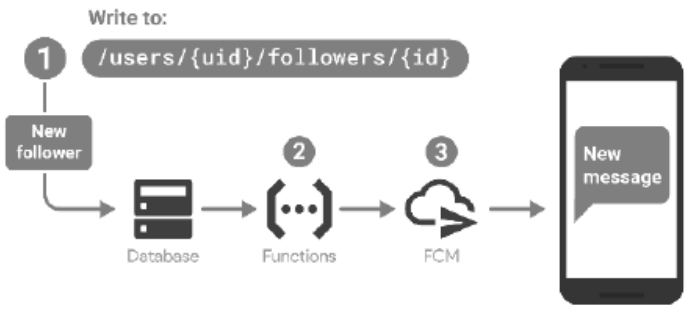
Para utilizar Firebase Functions, debemos instalar Firebase CLI e inicializar Firebase Functions en el proyecto de Firebase. Posteriormente, podremos escribir el código JavaScript de la función. Por último, implementamos la función en los servidores de Firebase con Firebase CLI. Podemos utilizar la consola de Firebase para ver y buscar en los registros de funciones implementadas.

Cloud Functions permite que los programadores accedan a eventos de Firebase y Google Cloud y brinda la capacidad de procesamiento escalable para ejecutar código en respuesta a eventos. Los casos típicos en los que se usa son:

- Notificar a los usuarios cuando ocurre algo interesante.
- Ejecución de limpieza y mantenimiento en Realtime Database.
- Ejecución de tareas intensivas en la nube en lugar de en la aplicación.
- Realizar integraciones con API y servicios de terceros.

Notificar a los usuarios cuando ocurre algo interesante

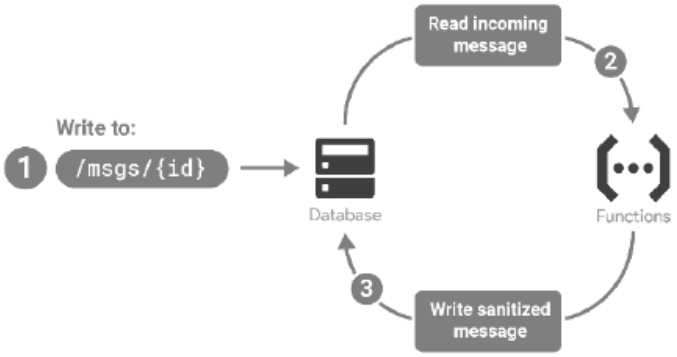
Podemos utilizar Cloud Functions para mantener a los usuarios interesados y actualizados con información relevante. Por ejemplo, una función que se activa cuando se escribe en Realtime Database para almacenar nuevos registros, podría generar notificaciones de Firebase Cloud Messaging (FCM) para informar a los usuarios.



La función se activa cuando se escribe en una ruta de acceso de Realtime Database que el programador ha definido. La función crea un mensaje para enviarlo a través de FCM. FCM envía el mensaje de notificación al dispositivo del usuario.

Ejecución de limpieza y mantenimiento en Realtime Database

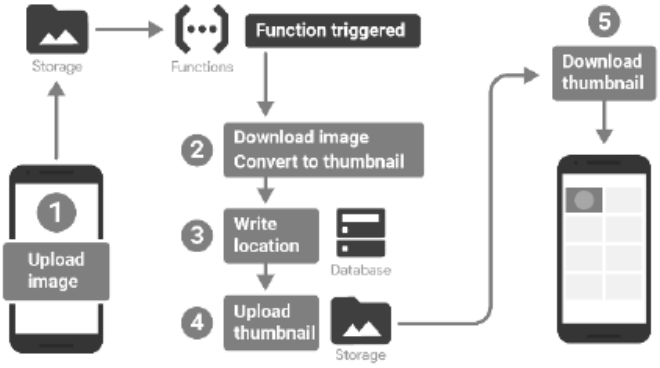
Con Cloud Functions, podemos modificar Realtime Database para mantener el sistema actualizado y limpio. Por ejemplo, en una aplicación que admita comentarios y que sean almacenados en Realtime Database, se podrían supervisar los eventos de escritura y quitar el texto inapropiado.



El controlador de eventos de la base de datos detecta eventos de escritura en una ruta de acceso específica y recupera datos del evento con el texto insertado. La función procesa el texto para detectar y quitar el lenguaje inapropiado. Vuelve a escribir el texto actualizado en la base de datos.

Ejecución de tareas intensivas en la nube en lugar de en la aplicación

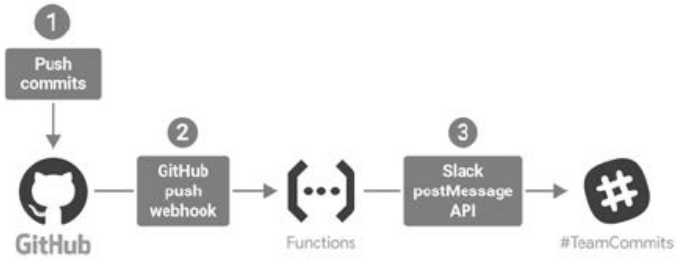
Podemos aprovechar Cloud Functions para ejecutar, en la nube de Google, trabajo que requiere una gran cantidad de recursos (uso intensivo de CPU o de red) y que no sería práctico ejecutar en el dispositivo de un usuario. Por ejemplo, una función que detecte cuándo se suben imágenes a Firebase Storage, descargue la imagen a la instancia que ejecuta la función, la modifique y la vuelva a subir a Storage. Se podría modificar el tamaño, recortar o convertir la imagen.



La función se activa cuando un archivo de imagen se sube a Firebase Storage. La función descarga la imagen y crea una versión en miniatura. La función escribe la ubicación de esa miniatura en la base de datos, de manera que una aplicación cliente pueda encontrarla y usarla. La función vuelve a subir la miniatura a Firebase Storage en una ubicación nueva. La aplicación descarga el vínculo de la miniatura.

Realizar integraciones con API y servicios de terceros

Cloud Functions puede ayudar a que una aplicación funcione con servicios de terceros a través de llamadas API web. Por ejemplo, una aplicación que se usa para la programación colaborativa podría notificar las confirmaciones de GitHub a un grupo de trabajo.



Un usuario envía confirmaciones a un repositorio de GitHub. Una función HTTPS se activa a través de la API de webhook de GitHub. La función envía una notificación de la confirmación al equipo de trabajo.

5.1.2. Configurar Cloud Functions

Firebase Cloud Functions utiliza Firebase CLI para programar funciones. Se realiza de forma parecida a como lo utilizamos con Firebase Hosting. También hay que inicializar Cloud Functions en el proyecto de Firebase.

En primer lugar, instala Firebase CLI, si has realizado el apartado de Firebase Hosting de la unidad anterior, ya debes tenerlo instalado.

Ejercicio: Inicializar un proyecto Cloud Functions en Firebase CLI

1. Instala Firebase CLI.

   Firebase CLI (Command Line Interface) requiere Node.js y npm. Puedes instalar ambos siguiendo las instrucciones en https://nodejs.org. Instalando Node.js también se instala npm. Firebase CLI necesita la versión de Node.js 0.10.0 o superior.

   Una vez instalado Node.js y npm, puedes instalar Firebase CLI vía npm, ejecuta desde la línea de comandos:

```
npm install -g firebase-tools
```

Para actualizar a la última versión, simplemente ejecuta el mismo comando.

2. Loguéate en Firebase.

   Ejecuta el comando `firebase login`. Si tienes una sesión de Google activa y que se pueda conectar a Firebase Console, debe devolver el mensaje: "Already logged in as xxxx@gmail.com". Si no es así, se lanzará el navegador predeterminado para que inicies la sesión en una cuenta de Google y autorices el acceso a Firebase desde Firebase CLI. Puedes cerrar la sesión ejecutando `firebase logout`.

3. Inicializa la aplicación.

   Abre la consola del sistema operativo y sitúate en el directorio del ordenador donde tienes o vas a desarrollar el proyecto de Firebase, ejecutando el comando `cd`. En nuestro caso, accede a la carpeta eventos creada anteriormente, por ejemplo, con `cd c:\eventos` en Windows. Posteriormente, ejecuta el comando `firebase init`:

```
cd ruta_carpeta_proyecto
firebase init
```

El comando `firebase init` lanza un asistente de configuración del proyecto. Hemos de tener en cuenta que ya hemos inicializado el proyecto para Firebase Hosting. Vamos a ver cómo añadir una nueva función a un proyecto ya inicializado:

```
You're about to initialize a Firebase project in this directory:
  C:\eventos
Before we get started, keep in mind:

  * You are currently outside your home directory
  * You are initializing in an existing Firebase project directory

? Are you ready to proceed? (Y/n)
```

El asistente nos advierte de que ya existe un proyecto Firebase en el directorio. Introduce `y` y pulsa la tecla `INTRO` para indicar que estás de acuerdo en proceder con la configuración del proyecto.

```
? Which Firebase CLI features do you want to setup for this folder? Press
Space to select features, then Enter to confirm your choices.
 ( ) Database: Deploy Firebase Realtime Database Rules
 ( ) Firestore: Deploy rules and create indexes for Firestore
>(*) Functions: Configure and deploy Cloud Functions
 (*) Hosting: Configure and deploy Firebase Hosting sites
 ( ) Storage: Deploy Cloud Storage security rules
```

Nos preguntan qué características de Firebase queremos activar. Tenemos que elegir las que ya tenemos inicializadas (Hosting) y la nueva (Functions). Marca las dos opciones. Pulsa la tecla INTRO para continuar.

```
=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

i  .firebaserc already has a default project, skipping
```

Como ya tenemos un proyecto de Firebase asociado al proyecto en Firebase CLI, no nos pregunta qué proyecto queremos asociar. Ya lo hemos hecho en el apartado de Firebase Hosting.

```
=== Functions Setup

A functions directory will be created in your project with a Node.js
package pre-configured. Functions can be deployed with firebase deploy.

+  Wrote functions/package.json
+  Wrote functions/index.js
? Do you want to install dependencies with npm now? (Y/n)
```

Este es el único apartado específico para configurar Firebase Functions, el resto, salvo los generales del proyecto, nos aparecen porque vamos a sobrescribir la configuración de un proyecto ya inicializado. Seguidamente, nos pregunta por la configuración específica de Firebase Functions. El asistente creará un directorio nuevo llamado functions y dos ficheros: package.json e index.js. Nos pregunta si queremos instalar las dependencias, le indicamos que sí, introduciendo Y y pulsando INTRO. Esperamos un momento mientras se instalan las dependencias.

```
=== Hosting Setup

Your public directory is the folder (relative to your project directory)
that will contain Hosting assets to be uploaded with firebase deploy. If
you have a build process for your assets, use your build's output
directory.

? What do you want to use as your public directory? (public)
```

Indicamos cuál será la carpeta que contendrá los archivos que se subirán a Firebase Hosting. Por defecto, su nombre será public. Dejarás este nombre y seguirás adelante, pulsando la tecla INTRO. Es lo que hemos indicado en la anterior instalación, y es conveniente mantener la misma configuración:

```
? Configure as a single-page app (rewrite all urls to /index.html)? (y/N)
```

Tal y como has hecho en la anterior inicialización, elegirás que no quieres configurar una sola página para la aplicación. Pulsa la tecla INTRO para continuar.

```
? File public/404.html already exists. Overwrite? (y/N)
```

Indica que no quieres sobrescribir la página 404.html.

```
? File public/index.html already exists. Overwrite? (y/N)
```

Tampoco queremos sobrescribir el fichero index.html. Ya lo tenemos implementado desde el apartado de Hosting.

```
+  Firebase initialization complete!
```

Hemos terminado la inicialización del proyecto en Firebase CLI. La ejecución del comando `firebase init` crea varios archivos y carpetas en el directorio del proyecto. Sirven para configurar e implementar el proyecto.

Después de inicializar el proyecto para Functions, la estructura del proyecto será similar a la siguiente:

```
proyecto
 +- .firebaserc # Archivo oculto que ayuda a cambiar de forma rápida
 |              # entre proyectos con: firebase use
 |
 +- firebase.json # Describe propiedades del proyecto
 |
 +- functions/ # Directorio que contiene el código de las funciones.
      |
      +- package.json  # Fichero del paquete npm que
      |                # describe el código de Cloud Functions
      +- index.js   # Fichero principal de código de Cloud Functions
      |
      +- node_modules/ # Directorio donde estan instaladas
                       # las dependencias (declaradas en package.json)
```

Nuestra carpeta de proyecto contendrá más directorios y archivos, ya que hemos configurado una nueva caraterística de Firebase en el proyecto. Ya podemos comenzar a agregar código. Lo haremos en el fichero index.js que encontrarás en la carpeta functions. El lenguaje de programación que debemos utilizar es JavaScript.

Una vez implementadas las funciones en el fichero index.js, deberemos subir la función a los servidores de Firebase. Para ello utilizaremos el comando `firebase deploy --only functions`. `firebase deploy` ya lo hemos utilizado en Firebase Hosting para sincronizar el proyecto local con Firebase. Añadimos `–only functions` si queremos que soólo sincronice las funciones.

Para programar las funciones, debemos importar los módulos necesarios. Vamos a ver cómo escribir el código de las funciones.

5.1.3. Escribir funciones

Las funciones las escribiremos en el fichero index.js del proyecto.

Debemos importar los módulos que se necesiten mediante el método `require`. Será obligatorio incluir `require` para el SDK de Firebase Cloud Functions. Nos permite crear funciones Cloud Functions y configurar los desencadenadores. Lo haremos incluyendo la siguiente línea en el fichero index.js:

```
const functions = require('firebase-functions');
```

Además, dependiendo de la funcionalidad de las funciones a programar, deberemos importar diferentes módulos. Por ejemplo, `require('firebase-admin')` será necesario si queremos ejecutar acciones como leer o escribir datos en Realtime Database, enviar mensajes de Firebase Cloud Messaging, generar y verificar tokens de autentificación de Firebase o para crear una consola de administrador simplificada.

Firebase CLI instala de forma automática los módulos de Node de Firebase y del SDK de Firebase para Cloud Functions cuando se inicializa un proyecto. Para agregar bibliotecas de terceros al proyecto, podemos modificar el fichero functions/package.json, ejecutar `npm install` y realizar solicitudes.

Nos quedaría agregar el código propio de las funciones. Distinguimos tres partes en cada función: nombre de la función, desencadenador asociado y código a ejecutar.

```
exports.nombreFuncion = functions.desencadenador((req, res) => {
   //Código a ejecutar
});
```

Definiremos el nombre de las funciones con la instrucción `exports.` seguido del nombre que queramos.

Tenemos que asociar un objeto a cada desencadenador. También tendremos que indicar qué evento del objeto va a ser el que va a activar la función. Por ejemplo, asociamos a una función un objeto Realtime Database e indicamos que se va a activar cuando se produzca un evento de escritura. En la siguiente tabla, puedes ver los desencadenadores y los objetos asociados; posteriormente, veremos los eventos de cada objeto:

Desencadenadores	Objeto
Cloud Firestore	`functions.firestore`
Realtime Database	`functions.database`
Authentication	`functions.auth`
Analytics	`functions.analytics`
Crashlytics	`functions.crashlytics`
Cloud Storage	`functions.storage`
HTTP	`functions.https`
Cloud Pub/Sub	`functions.pubsub`

5.1.3.1. Activadores de Cloud Firestore

Los eventos que se disparan cuando realizamos cambios en los datos de Cloud Firestore, son los que ejecutan las funciones de Firebase Functions.

El ciclo de vida de una función Cloud Firestore es el siguiente:

1. Espera a que ocurran cambios en un documento en particular.
2. Se activa cuando ocurre un evento implementado en la función.
3. Recibe un objeto de datos del evento que contiene dos instantáneas de los datos almacenados: una con los datos originales previos al cambio y una con los datos nuevos.

Un objeto `functions.firestore` para Cloud Firestore admite los siguientes eventos:

Evento	Descripción
`onCreate`	Se activa cuando se escribe en un documento por primera vez
`onUpdate`	Se activa cuando un documento ya existe y se cambia uno de sus valores
`onDelete`	Se activa cuando se borra un documento con datos
`onWrite`	Se activa cuando se activa `onCreate`, `onUpdate` u `onDelete`

Los eventos de Cloud Firestore se activan solo con los cambios en documentos. No es posible agregar eventos a campos específicos.

Por ejemplo, para activar un evento para cualquier cambio en un documento específico, podemos usar la función `functions.firestore.document('…').onWrite(…)`.

```
// Escuchador para cualquier cambio en el documento 'fichero'
// de la colección 'empresa'
exports.nombreFuncion = functions.firestore
                        .document('empresa/fichero').onWrite((event) => {
   // ... Aquí el código
  });
```

Cuando se activa una función, es posible que sea interesante obtener los datos actualizados del documento o los datos antes de la actualización. Para obtener los datos, hay que usar `event.data`. Los datos actuales del documento los obtendremos mediante `event.data.data()` y los datos anteriores a la actualización, con `event.data.previous.data()`.

```
exports.actualizarUsuario = functions.firestore.document('users/{userId}')
                                                    .onUpdate(event => {
    // Obtenemos el objeto que contiene el documento actual
    var newValue = event.data.data();
    // Obtenemos el objeto que contiene el documento anterior
    var previousValue = event.data.previous.data();
   });
```

Con un objeto `event.data` podemos acceder a las propiedades del mismo modo que con cualquier otro objeto. También podemos utilizar la función `get` para

acceder a campos específicos, por ejemplo: `event.data.data().edad` o `event.data.data()['nombre']`.

Cada invocación de función está asociada a un documento específico en la base de datos de Cloud Firestore. Se puede acceder a ese documento con `DocumentReference` llamando a `event.data.ref`. `DocumentReference` incluye métodos como `update()`, `set()` y `remove()` para que podamos modificar con facilidad el documento que activó la función. Cada vez que se escribe en el mismo documento que activó una función, se corre el riesgo de crear un bucle infinito. Hay que tener cuidado y asegurase de salir de forma segura de la función cuando no necesitemos hacer cambios.

Si no se conoce el `ID` del documento específico al que deseamos adjuntar un activador, se puede usar un `{wildcard}`.

```
exports.useWildcard = functions.firestore.document('users/{userId}')
                                                    .onWrite((event) => {
    // Código a ejecutar
  });
```

En este ejemplo, cuando se cambia algún campo en cualquier documento dentro de `users`, la función será activada y en `userId` se nos indicará el nombre del documento. Si un documento de `users` tiene subcolecciones y se cambia un campo en uno de los documentos de esas subcolecciones, la función no se activa.

Las coincidencias de comodines se extraen de la ruta del documento y se almacenan en `event.params`. Se pueden definir tantos comodines como deseemos para sustituir los `ID` explícitos de colección o de documento.

```
exports.useMultipleWildcards = functions.firestore
            .document('users/{userId}/{messageCollectionId}/{messageId}')
            .onWrite((event) => {
    //Código a ejecutar
  });
```

Si modificáramos en el documento `"users/maria/mensajes/134"` el campo `{texto: "Hola"}`, obtendríamos los siguientes valores en `event.params` y `event.data`:

```
event.params.userId == "maria";
event.params.messageCollectionId == "mensajes";
event.params.messageId == "134";
event.data.data() == {texto: "Hola"}
```

El activador siempre debe apuntar a un documento, incluso si usamos un comodín. Por ejemplo, `users/{userId}/{messageCollectionId}` no es válido porque `{messageCollectionId}` es una colección. Sin embargo, `users/{userId}/{messageCollectionId}/{messageId}` es válido porque `{messageId}` siempre apuntará a un documento.

Vamos a ver un ejemplo en la aplicación “Eventos”. Al modificar una imagen de un evento, se lanzará una notificación Firebase Cloud Messaging a los usuarios de la aplicación anunciando el cambio. Utilizaremos Firebase Functions para Firestore, ya que, al subir una nueva imagen para un evento, se actualiza la ruta a la nueva imagen en la base de datos y eso será el evento activador.

Ejercicio: Activador Firestore en Firebase Functions

1. Abre el fichero index.js del directorio functions del proyecto “Eventos” que has creado con Firebase CLI con un editor de texto y elimina todas las líneas comentadas (las que empiezan por //).

   El fichero contendrá solamente la línea: `const functions = require('firebase-functions');`. De esta forma, importamos el SDK de Firebase para Cloud Functions para crear funciones y configurar los desencadenadores.

2. Añade el siguiente código al final del fichero:

```
const admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);
```

   Importamos el SDK de Firebase Admin para acceder a Firestore y Firebase Cloud Messaging y lo inicializamos.

3. Inserta la siguiente función al final del fichero:

```
var tema = "Todos";
exports.notificaFoto = functions.firestore.document('eventos/{evento}')
                                                    .onWrite(event => {
   if (event.data.data()['imagen']!=event.data.previous.data()['imagen']){
      const datos = {
         notification: {
            title: 'Nueva imagen de '+ event.data.data()['evento'],
               body: "Se ha cambiado la imagen del evento "
               + event.data.data()['evento']
         }
      };
      admin.messaging().sendToTopic(tema, datos)
         .then(function(response) {
            console.log("Envío correcto:", response);
         })
         .catch(function(error) {
            console.log("Envío erróneo:", error);
         });
   }
});
```

4. Accede a la consola de comandos y, posicionado en la carpeta del proyecto, ejecuta la siguiente instrucción para implementar la función en Firebase:

```
firebase deploy --only functions
```

5. Accede a la consola de Firebase. En el apartado DEVELOP selecciona Functions. Comprueba que la función está dada de alta.
6. Cambia la imagen de un evento desde la aplicación y comprueba que recibes una notificación.

   Comprueba que has recibido una notificación al realizar el cambio en la imagen de un evento. Al tener la aplicación abierta verás un cuadro de diálogo, no una notificación, en el área de notificaciones del dispositivo.

Hemos creado una función de Firebase llamada `notificaFoto`. Se activará cuando cuando se haga un cambio en el campo `imagen` de un documento de Firestore, para ello indicamos que `functions.firestore.document` será el objeto que desencadenará la función. Con `'eventos/{evento}'` expresamos que cualquier cambio en un documento de la colección `eventos` activará la función. Con `onWrite` implementamos que será cualquier cambio el que desencadenará la ejecución del código de la función. Solo queremos que se lance la notificación cuando se cambie el campo `imagen` de cualquier documento, si se cambia cualquier otro campo no queremos que avise. La comparación del valor actual del campo (`event.data.data()['imagen']`) con el anterior (`event.data.previous.data()['imagen']`) determinará si se lanza o no la notificación.

Enviamos las notificaciones a un tema determinado mediante el método `admin.messaging().sendToTopic`. Necesitamos especificar el tema al cual enviar la notificación (vamos a enviarlo al tema `'Todos'`) y los datos de la notificación. Los datos de la notificación los especificamos mediante un objeto `Notification` rellenando los campos `title` y `body`.

Otra forma de probar que funciona la función en Firestore es cambiar el valor del campo imagen en algún evento. Realiza un cambio y comprueba que al guardar los cambios recibes una notificación. Vuelve a poner el valor original o no se verá la imagen en la aplicación.

5.1.3.2. Activadores de Realtime Database

Permiten ejecutar operaciones en los servidores de Google que se activan cuando ocurren ciertos eventos en Firebase Realtime Database.

Para crear funciones `functions.database` para eventos Realtime Database, debemos especificar qué evento activa la función y la ruta de acceso a la base de datos.

Las funciones permiten controlar los eventos específicos de creación, actualización o eliminación, o detectar cambios de cualquier tipo. Los eventos disponibles son:

Evento	Descripción
`onCreate`	Se activa cuando se crean nuevos datos

Evento	Descripción
`onUpdate`	Se activa cuando se actualizan datos
`onDelete`	Se activa cuando se borran datos
`onWrite`	Se activa cuando se activa `onCreate`, `onUpdate` u `onDelete`

Para controlar cuándo se debe activar la función, implementamos `ref(path)`. Este método especifica la ruta de acceso dentro de la base de datos que activará la función. Por ejemplo:

```
functions.database.ref('/foo/bar')
```

Los eventos de una ruta afectan a la ruta, incluidos aquellos que ocurren en cualquier lugar dentro de ella. Por ejemplo, si configuramos la ruta de acceso `/foo/bar` para la función, se muestran coincidencias de eventos en estas dos ubicaciones:

```
 /foo/bar
 /foo/bar/baz/really/deep/path
```

En ambos casos, Firebase interpreta que el evento ocurre en `/foo/bar`. Los datos del evento incluyen los datos antiguos y actualizados en `/foo/bar`. Si la cantidad de datos que disparan el evento son grandes, es mejor usar varias funciones con rutas de acceso más profundas en lugar de una sola función cerca de la raíz. Para mejorar el rendimiento, solicitamos que los datos que disparen la función estén en el nivel más profundo posible.

Se puede especificar una ruta de acceso con comodines. Encerramos entre corchetes el comodín, por ejemplo: `ref('foo/{bar}')`, que coincide con cualquier elemento secundario de `/foo`. Los valores comodín están disponibles en la función con del objeto `event.params`. En este ejemplo, podemos consultar el valor llamando a `event.params.bar`.

Las rutas de acceso con comodines pueden coincidir con varios eventos de una misma escritura. Por ejemplo, la inserción de

```
{
  "foo": {
    "hola": "mundo",
    "firebase": "functions"
  }
}
```

coincide con la ruta de acceso `"/foo/{bar}"` dos veces: una vez con `"hola": "mundo"` y otra con `"firebase": "functions"`.

Al igual que con Firestore, podemos recuperar los datos originales y los modificados mediante un objeto `event.data`. El objeto recuperado es un `DeltaSnapshot`. Para leer los datos antes de la modificación, usamos `event.data.previous` y para leer los datos tras la modificación, usamos `event.data.current`, o simplemente `event.data`.

En algunos casos, no necesitamos el valor antiguo y solo precisamos saber si los datos se cambiaron. Para esto usamos el método `changed()` del objeto `DeltaSnapshot`.

5.1.3.3. Activadores de Firebase Authentication

Podemos activar una función en respuesta a la creación o la eliminación de cuentas de usuario de Firebase Authentication.

Los eventos disponibles para el objeto `functions.auth.user` son:

Evento	Descripción
`onCreate`	Se crea un usuario
`onDelete`	Se elimina un usuario

Por ejemplo, cuando se crea un usuario podríamos enviar un correo electrónico de bienvenida. Para ello usaríamos el evento `onCreate()`:

```
exports.enviarEmailBienvenida = functions.auth.user().onCreate(event => {
  // Código a ejecutar
});
```

Firebase activará eventos de creación de usuarios en los siguientes casos:

- Un usuario crea una cuenta de correo electrónico y una contraseña.
- Un usuario accede por primera vez con un proveedor de identidad federada.
- El programador crea una cuenta con el SDK de administrador de Firebase.
- Un usuario accede a una sesión de autenticación anónima por primera vez.

Cloud Functions no activa ningún evento cuando un usuario accede por primera vez con un token personalizado.

A través de `event.data`, podemos acceder a los atributos de un usuario recién creado con el objeto `UserRecord`. Por ejemplo, con `event.data.email` obtenemos el correo electrónico.

Cuando se elimina un usuario, se activan los eventos de eliminación de usuarios. Debemos usar el controlador de eventos `functions.auth.user().onDelete()`:

```
exports.enviarEmail = functions.auth.user().onDelete(event => {
  // Código a ejecutar
});
```

5.1.3.4. Activadores de Google Analytics para Firebase

Google Analytics para Firebase proporciona informes de eventos que nos ayudan a comprender la forma en que los usuarios interactúan con una aplicación. Con Cloud Functions, podemos acceder a los eventos de conversiones que registramos y activar funciones basadas en esos eventos.

Los eventos disponibles son `functions.analytics.event`:

Evento	Descripción
`onLog`	Se produce una conversión

Actualmente, Cloud Functions solo admite eventos marcados como eventos de conversión. Podemos especificar cuáles son los eventos de conversión en la pestaña **Eventos** del panel ANALYTICS de la consola de Firebase.

Cloud Functions admite ejecutar funciones con la activación de eventos `AnalyticsEvent` de Google Analytics. Se activan cada vez que la actividad del usuario genera un evento de conversión. Por ejemplo, podríamos recibir un e-mail cuando se genere el evento `in_app_purchase`, para indicar que se ha producido una compra desde la aplicación. Debemos especificar el evento de Analytics que deseamos que active la función con el método `functions.analytics.event()` y supervisar el evento dentro del controlador de evento `onLog()`:

```
exports.enviarEmailCompra =
            functions.analytics.event('in_app_purchase').onLog(event => {
     // Código a ejecutar
   });
```

Con cada evento de Analytics, tenemos acceso a todos los parámetros y las propiedades de usuario relevantes. Incluye información sobre el usuario, el dispositivo, la aplicación y la información geográfica del evento. Para obtener una lista completa de parámetros y propiedades del usuario, consultamos la referencia `functions.analytics`.

Observa este ejemplo, de una función activada por una compra, que recoge diversos parámetros del usuario, aplicación y dispositivo:

```
exports.enviarCuponPorCompra =
           functions.analytics.event('in_app_purchase').onLog(event => {
      const user = event.data.user;
      // Id de usuario.
      const uid = user.userId;
      // Cantidad compra en $.
      const purchaseValue = event.data.valueInUSD;
      // Lenguaje del usuario
      const userLanguage = user.deviceInfo.userDefaultLanguage;
      // Enviar cupon via notificación FCM
      return enviarCuponViaFCM(uid, userLanguage);
});
```

Seguidamente vamos a implementar en la aplicación “Eventos” una función que nos avise por e-mail cuando un usuario desinstale la aplicación.

Ejercicio: Activador Google Analytics en Firebase Functions

1. Accede a la consola de Firebase y selecciona el proyecto Eventos.
2. En el apartado ANALYTICS selecciona la opción **Events**.
3. Activa como un "Evento de conversión" el evento `app_remove`.

   Recuerda que solamente se pueden asociar funciones a eventos de conversión.

4. Abre el fichero index.js del directorio functions del proyecto "Eventos" que has creado con Firebase CLI con un editor de texto y añade el siguiente código:

```
const nodemailer = require('nodemailer');
exports.enviarEmailDesinstalacion =
                functions.analytics.event('app_remove').onLog(event => {
   const gmailEmail = functions.config().gmail.email;
   const gmailPassword = functions.config().gmail.password;
   const mailTransport = nodemailer.createTransport({
      service: 'gmail',
      auth: {
         user: gmailEmail,
         pass: gmailPassword
      }
   });
   const opcionesEmail = {
      from: '${APP_NAME} <noreply@firebase.com>',
      to: 'tucorreo@correo.com',
      subject: 'Desinstalación aplicación Eventos',
      text: 'Un usuario ha desinstalado la aplicación Eventos'
   };
   return mailTransport.sendMail(opcionesEmail);
});
```

   Sustituye `tucorreo@correo.com` por la dirección del e-mail que va a recibir los correos.

   Hemos creado una función llamada `enviarEmailDesinstalacion` y asociado a un evento de desinstalación de la aplicación mediante `functions.analytics.event('app_remove')`. Envía un correo electrónicio desde la función, para ello utiliza `nodemailer`. Envía los e-mails mediante una cuenta de Gmail, pero debes tener en cuenta las limitaciones de envío de Gmail. Si tienes unas necesidades más grandes, considera utilizar cuentas profesionales como Sendgrid, MailChimp…

5. Para habilitar el envío de correo desde una cuenta de Gmail, tienes que permitir el acceso a la cuenta desde aplicaciones menos seguras y

desbloquear el captcha de la cuenta. Para ello, accede a las siguientes direcciones y realiza las acciones pertinentes:

Habilita el acceso de aplicaciones menos seguras:
https://www.google.com/settings/security/lesssecureapps

Desbloquea el captcha de la cuenta:
https://accounts.google.com/DisplayUnlockCaptcha

6. Abre una ventana con la consola de comandos y sitúate en el directorio functions de la carpeta del proyecto "Eventos" que has implementado en Firebase CLI.

7. Ejecuta la siguiente instrucción en la consola de comandos para actualizar todas las dependencias del proyecto:

```
npm install
```

Asegúrate de estar situado en la carpeta functions cuando ejecutes la instrucción.

8. Configura el valor de las variables Google Cloud (gmail.email y gmail.password) con el e-mail y la contraseña de la cuenta de Gmail que será usada para enviar los correos electrónicos. Ejecuta la siguiente instrucción en la consola de comandos:

```
firebase functions:config:set gmail.email="cuenta@gmail.com"
gmail.password="contraseña"
```

Sustituye `"cuenta@gmail.com"` por la cuenta de Gmail que enviará los correos y `"contraseña"` por la contraseña de la cuenta.

9. Despliega el proyecto en la consola de Firebase con la siguiente instrucción:

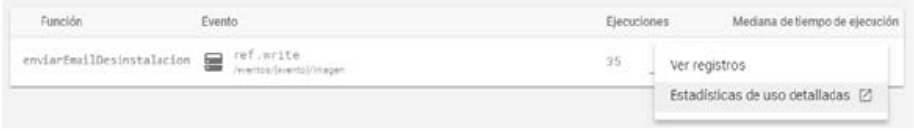
```
firebase deploy --only functions
```

Si devuelve el siguiente error: "`Error: Error parsing triggers: Cannot find module 'nodemailer'`", ejecuta la siguiente instrucción: `npm install nodemailer --save`.

10. Prueba el desencadenador.

Puedes probar el desencadenador de dos formas: haciendo una prueba desde la consola de Firebase o desinstalando la aplicación en un dispositivo.

Se puede activar la función desde la consola de Firebase para realizar pruebas. Desde la consola de Firebase, selecciona **Functions**. En la función enviarEmailDesinstalacion, selecciona la opción **Estadísticas de uso detalladas**.



Selecciona la pestaña **Prueba** en la nueva ventana que se ha abierto. Asegúrate de que la aplicación no se encuentre en primer plano. Pulsa en el botón **Prueba la función**.

Esta técnica te puede servir para probar de forma rápida algunas funciones y sus efectos en la aplicación.

Para probar desde el dispositivo simplemente desinstala la aplicación “Eventos” desde los ajustes del dispositivo.

En ambos casos, vas a recibir un correo electrónico que te avisa de que un usuario ha desinstalado la aplicación.

5.1.3.5. Activadores de Firebase Crashlytics

Podemos implementar funciones en respuesta a eventos Crashlytics. Estos eventos pueden ser por nuevos problemas, problemas regresivos o alertas de velocidad. Por ejemplo, podríamos notificar a nuestro equipo de desarrollo cuando se produzca un problema de velocidad.

Los eventos disponibles son:

Evento	Descripción
`onNewDetected`	Se produce un problema por primera vez
`onRegressed`	Se experimenta un problema que ya se ha producido
`onVelocityAlert`	Un dispositivo presenta problemas de velocidad

Para crear una función Crashlytics, primero generaremos un objeto `IssueBuilder` con `functions.crashlytics.issue()`, entonces llamaremos a la función apropiada.

En el siguiente ejemplo, vemos cómo la función se activa cuando la aplicación experimenta un problema por primera vez:

```
exports.nuevoError=functions.crashlytics.issue().onNewDetected(event => {
   // Código a ejecutar
});
```

También podemos capturar eventos que ya se han producido con `onRegressed`:

```
exports.errorConocido = functions.crashlytics.issue()
                                                .onRegressed(event => {
   // Código a ejecutar
});
```

Por último, indicar que también podemos capturar eventos de velocidad. Se producen cuando el dispositivo presenta problemas de velocidad:

```
exports.alertaVelocidad = functions.crashlytics.issue()
                                             .onVelocityAlert(event => {
   // Código a ejecutar
});
```

Cada evento activado creado con `IssueBuilder` retorna un problema. Un problema tiene propiedades, que incluyen: nombre del problema, un identificador, información relevante y mucho más. Por ejemplo, para obtener el identificador de un problema utilizaríamos `event.issueId`, y para el título `event.issueTitle`.

5.1.3.6. Activadores de Cloud Storage

Activan una función en respuesta a la carga, actualización o eliminación de archivos y carpetas en Cloud Storage. Podemos definir el alcance de la función para un depósito específico de Cloud Storage o utilizar el depósito predeterminado.

Los eventos disponibles son:

Evento	Descripción
`onChange`	Se activa cuando se produce cualquier cambio sobre un elemento del almacenamiento

Veamos un ejemplo del alcance de una función:

- `functions.storage.object()` detecta cambios de objetos en el depósito de almacenamiento predeterminado.
- `functions.storage.bucket('depositoA').object()` detecta los cambios de objetos en un depósito específico.

El siguiente código muestra una función que se activa si se produce un evento en el depósito de almacenamiento predeterminado:

```
exports.nombreFuncion = functions.storage.object().onChange(event => {
   // Código a ejecutar
});
```

En el ejemplo anterior hemos utilizado el evento `onChange`. Se activa cuando cambia un objeto dentro de un depósito (cuando se crea, modifica o borra un objeto).

Cloud Functions obtiene varios atributos del objeto modificado en `event`, como, por ejemplo, `timestamp`, `resource`, `size` y `contentType`. El atributo `resourceState` tiene el valor `"exists"` (si se ha creado o modificado un objeto) o `"not_exists"` (si se ha eliminado o movido). El atributo `resourceState` debe sincronizarse con el atributo `metageneration` para saber si un objeto acaba de crearse. Es un contador que aumenta cuando se produce un cambio en los metadatos del objeto. En un objeto nuevo, su valor es 1.

Veamos un ejercicio en el que vamos a generar imágenes en miniatura a partir de las imágenes subidas desde la aplicación "Eventos".

Ejercicio: Crear miniatura al subir imagen a Cloud Storage

1. Abre el fichero index.js del directorio functions del proyecto Eventos y añade la siguiente función al final del fichero:

```
exports.crearMiniatura = functions.storage.object().onChange(event => {
   // Código a ejecutar
});
```

2. Declara las siguientes variables debajo de la línea `// Codigo a ejecutar` :

```
const objecto = event.data;
const deposito = objecto.bucket;
const ruta = objecto.name;
const tipo = object.contentType;
const estado = object.resourceState;
const metageneration = object.metageneration;
```

La variable `objeto` [`event.data`] contiene el objeto Storage que ha activado la función. El depósito que contiene el fichero lo guardamos en la variable `deposito` [`event.data.bucket`]. Guardamos en la variable `ruta` [`event.data.name`], donde se encuentra el fichero. `tipo` [`event.data.contentType`] contiene el tipo de fichero modificado. `estado` [`event.data.resourceState`] nos informa de si es un alta o modificación (`'exists'`), o bien de un borrado o que se ha movido el fichero (`'not_exists'`). En la variable `metageneration` [`event.data.metageneration`] guardamos el número de veces que han cambiado los metadatos del archivo, siendo 1 el valor para objetos nuevos.

3. Añade debajo del código anterior el siguiente:

```
// Salimos del desencadenador si el fichero no es una imagen.
if (!tipo.startsWith('image/')) {
  console.log('No es una imagen.');
  return;
}
```

```
// Obtenemos el nombre del fichero.
const nombre = path.basename(ruta);
// Salimos de la función si es una miniatura.
if (nombre.startsWith('thumb_')) {
  console.log('Existe una miniatura.');
  return;
}
// Salimos de la función si se trata de un evento mover o borrado.
if (estado === 'not_exists') {
  console.log('Evento de borrado.');
  return;
}
// Salimos si no es un fichero nuevo. Sólo ha cambiado los metadata.
if (resourceState === 'exists' && metageneration > 1) {
  console.log('Cambio de metadata.');
  return;
}
```

Hemos definido todos los casos en los que no se va a generar la miniatura de una imagen: el fichero no sea una imagen, si es una miniatura, si es un borrado… Lo que haremos será transformar una imagen en miniatura si se trata de un fichero imagen nuevo en Cloud Storage.

4. Para descargar y volver a subir objetos con facilidad en Cloud Storage, desde la instancia del evento, instala el paquete de Google Cloud Storage ejecutando la siguiente instrucción en la consola de comandos:

```
npm install --save @google-cloud/storage
```

5. Inserta al inicio del archivo las siguientes importaciones:

```
const gcs = require('@google-cloud/storage')();
const spawn = require('child-process-promise').spawn;
const path = require('path');
const os = require('os');
const fs = require('fs');
```

En un ejercicio anterior ya hemos importado el requerimiento más importante de las funciones (`'firebase-functions'`). Para usar promesas de JavaScript importamos `'child-process-promise'`. También importaremos `'@google-cloud/storage'`, `'path'`, `'os'` y `'fs'`, para descargar y realizar el procesamiento de ficheros.

6. Transforma una imagen subida en una miniatura en la función, para ello debes insertar el siguiente código a continuación, al final de la función `crearMiniatura`:

```
const bucket = gcs.bucket(deposito);
const ficheroTemporal = path.join(os.tmpdir(), nombre);
return bucket.file(ruta).download({
      destino: tempFilePath
```

```
   }).then(() => {
      return spawn('convert', [ficheroTemporal, '-thumbnail',
         '200x200>', tempFilePath]);
   }).then(() => {
   const thumbFileName = 'thumb_${fileName}';
   const thumbFilePath = path.join(path.dirname(filePath), thumbFileName);
   return bucket.upload(tempFilePath, {destination: thumbFilePath});
   }).then(() => fs.unlinkSync(tempFilePath));
```

7. Ejecuta desde la consola, posicionado en la carpeta del proyecto, la siguiente instrucción para implementar la función en Firebase:

```
firebase deploy --only functions
```

8. Accede a la consola de Firebase. En el apartado DEVELOP selecciona **Functions**. Comprueba que la función está dada de alta.
9. Cambia la imagen de un evento desde la aplicación.

Comprueba que recibes una notificación en la aplicación y que la imagen subida ha reducido su tamaño.

Hay casos en que no será necesario descargar archivos desde Cloud Storage. Sin embargo, para realizar tareas intensivas, como generar una miniatura, debes descargar el archivo en la instancia de la función (en la máquina virtual que ejecuta el código).

Utiliza `gcs.bucket.file(filePath).download` para descargar un archivo en un directorio temporal en la instancia de Cloud Functions. En esta ubicación, puedes procesar el archivo según sea necesario y luego subirlo a Cloud Storage. Cuando ejecutes tareas asíncronas, asegúrate de mostrar una promesa de JavaScript.

Cloud Functions proporciona un programa de procesamiento de imágenes llamado `ImageMagick`. Puede manipular archivos de imágenes.

El código ejecuta el método `convert` del programa `ImageMagick` para crear una miniatura de 200 × 200 píxeles de la imagen original en un directorio temporal y después la sube a Cloud Storage.

5.1.3.7. Activadores HTTP

Podemos activar una función a través de una solicitud de HTTP mediante `functions.https`. Permite invocar una función síncrona a través de los siguientes métodos HTTP admitidos: `GET`, `POST`, `PUT`, `DELETE` y `OPTIONS`.

El evento disponible es:

Evento	Descripción
`onRequest`	Llega una solicitud HTTP

Vamos a ver un ejemplo de una función `GET` de HTTP. La función recupera la hora actual del servidor, formatea el tiempo según se especifica en un parámetro de consulta URL y envía el resultado en la respuesta HTTP.

Usaremos `functions.https` para crear una función que controle eventos HTTP. El controlador de eventos de una función HTTP detecta el evento `onRequest()` y admite dos argumentos específicos de HTTP: solicitud (`Request`) y respuesta (`Response`). El objeto `Request` da acceso a las propiedades de la solicitud HTTP que envió el cliente y el objeto `Response` proporciona una forma de enviar una respuesta al cliente.

```
exports.fecha = functions.https.onRequest((req, res) => {
   res.status(200).send(fechaFormateada);
});
```

Siempre terminamos una función HTTP con `send()`, `redirect()` o `end()`. De lo contrario, la función podría seguir ejecutándose y el sistema podría forzar su finalización. En el ejemplo, la función devuelve una fecha formateada en una variable llamada `fechaFormateada`, terminamos la función de la siguiente forma:

```
res.status(200).send(fechaFormateada);
```

Invocamos la función a través de su URL. El URL es único. Incluye lo siguiente (en ese orden):

- La región en la que se implementa la función. Por ahora solo se admite la región us-central1.
- El ID del proyecto de Firebase.
- cloudfunctions.net.
- El nombre de la función.

Por ejemplo, el URL para invocar `fecha()` tiene el siguiente aspecto:

```
https://us-central1-<project-id>.cloudfunctions.net/fecha
```

Podremos ver la dirección de la función desde la consola de Firebase una vez que esté implementada. La podremos consultar, desde el apartado FUNCTIONS, en la lista de funciones o en las “Estadísticas de uso detalladas” de la función, en la pestaña **Activador**.

Veamos cómo llamar a una función pasándole parámetros. Vamos a utilizar dos métodos para realizar la llamada: `GET` y `POST`.

- `GET`: Realizaremos una llamada mediante el método `GET`: añadiremos al URL de la función los parámetros y sus valores.

Para realizar una llamada `GET` bastará con crear un formulario HTML:

```
<form action="/url_funcion" method="get">
   Parámetro#1<input type="text" name="param1"><br>
   Parámetro#2<input type="text" name="param2"><br>
   <input type="submit" value="Enviar">
</form>
```

O podemos contruir directamente el URL para llamar a la función con los valores de los parámetros:

```
http://url_funcion?param1=valor1&param2=valor2&...&paramN=valorN
```

Añadiremos al URL los pares nombre del parámetro y valor que sean necesarios. El primer parámetro irá precedido por un símbolo ¿ y los siguientes por &. Indicaremos el valor del parámetro con el símbolo = seguido del valor.

- POST: Realizaremos una llamada mediante el método POST, añadiendo a la llamada de la función. Pero estos no estarán visibles en el URL de la llamada.

  Bastará con crear un formulario HTML que utilice el método POST:

```
<form action="/url_funcion" method="post">
   Parámetro#1<input type="text" name="param1"><br>
   Parámetro#2<input type="text" name="param2"><br>
   <input type="submit" value="Enviar">
</form>
```

Veamos un ejemplo de cómo llamar a una función y pasarle un mensaje simple:

```
curl -X POST -H "Content-Type:application/json" -d '{"mensaje":"Hola mundo"}' URL_FUNCION
```

El cuerpo de la solicitud se analiza de forma automática según el tipo de contenido y se llena en el cuerpo del objeto `request`. Por ejemplo:

Tipo de contenido	Cuerpo de la solicitud	Comportamiento
`application/json`	'{"nombre":"Pepe"}'	`request.body.nombre` equivale a "Pepe"
`application/octet-stream`	''texto'	`request.body` equivale a "6d792074657"
`text/plain`	'texto'	`request.body` equivale a "texto"
`application/x-www-form-urlencoded`	'nombre=Pepe'	`request.body.nombre` equivale a "Pepe"

Supongamos que se usa la siguiente solicitud para llamar a la función:

```
curl -X POST -H "Content-Type:application/json" -H "X-cabecera: 123" URL_FUNCION?foo=baz -d '{"texto":"algún texto"}'
```

Los datos enviados se materializarían en:

Propiedad/Método	Valor
`request.method`	“POST”
`request.get('x-cabecera')`	“123”
`request.query.foo`	“baz”
`request.body.texto`	“algún texto”

Podemos conectar una función HTTP con Firebase Hosting. Las solicitudes en un sitio de Firebase Hosting se pueden dirigir a funciones HTTP específicas.

Ejercicio: Crear un servicio Web con un activador HTTP

1. Abre el fichero index.js del directorio functions del proyecto Eventos con un editor de texto y añade la siguiente declaración al final del fichero:

```
const db = admin.firestore();
```

Para que Firebase Functions trabaje con Firestore, tenemos que utilizar una instancia de Firestore.

2. Añade el código de la función al final del fichero:

```
exports.mostrarEventos = functions.https.onRequest((req, res) => {
   var evento= req.query.evento;
   db.collection('eventos').doc(evento).get().then(doc => {
      if (doc.exists) {
         res.json(doc.data());
      } else {
         res.send("No se encuentra el evento "+evento+"");
      }
   }).catch(reason => {
      res.send("Error");
   })
})
```

Hemos implementado una función para que recoja los valores de los parámetros mediante el método `GET`. Los valores de los parámetros los obtendremos con la propiedad `query` de `Request`. Conseguimos el valor del parámetro `evento` con `req.query.evento`. Consultamos los datos de Firestore mediante `db.collection('eventos').doc(evento).get()` y los devolvemos en formato JSON con `res.json`.

3. Desde la consola del sistema operativo ejecuta la siguiente instrucción situándote en la carpeta functions del proyecto:

```
firebase deploy --only functions
```

4. Accede a la consola de Firebase, comprueba que se ha subido la función y copia la dirección URL que se ha generado. La encontrarás debajo del nombre de la función.

5. Abre un navegador web, pega en la barra de direcciones el URL de la función y añade `?evento=fallas`.

Observa que el resultado devuelto es la información en formato JSON del evento `"fallas"` guardada en Firestore. Además de información JSON o de otro tipo para ser tratada desde la aplicación de la que se realiza la llamada, las funciones HTTP son capaces de generar páginas dinámicas.

6. Añade la función `mostrarEventosHtml` al final del fichero:

```
exports.mostrarEventosHtml = functions.https.onRequest((req, res) => {
   var evento= req.query.evento;
   db.collection('eventos').doc(evento).get().then(doc => {
      if (doc.exists) {
         res.status(200).send(`<!doctype html>
           <head>
             <link rel="stylesheet"
                    href="https://fonts.googleapis.com/css?family=Ranga">
           </head>
           <body>
              <span style="font-family: 'Ranga',serif;
                                                    font-size:medium;">
                 El evento ${doc.data().evento} se realiza en
                 la ciudad de ${doc.data().ciudad} el dia
                 ${doc.data().fecha}.
              </span>
           </body>
           </html>`);
        } else {
           res.send("No se encuentra el evento "+evento+"");
        }
   }).catch(reason => {
      res.send("Error");
   })
})
```

Utilizamos el método `res.status(200).send` para devolver una página dinámica. La devolución es una cadena con el código de una página HTML. En nuestro caso, generamos un mensaje que indica que el evento, que hemos pasado como parámetro, se realiza en una ciudad en una determinada fecha. Los datos son extraidos de Firestore. Para incluir el valor de un campo de la base de datos en una cadena utilizamos `${doc.data().nombre_campo}`. Por ejemplo, para utilizar el valor del campo ciudad utilizaríamos `${doc.data().evento}`.

7. Desde la consola del sistema operativo ejecuta la siguiente instrucción situándote en la carpeta functions del proyecto:

```
firebase deploy –only functions
```

8. Abre un navegador web, pega en la barra de direcciones el URL de la función y añade `?evento=fallas`.

   Comprueba que aparece la frase: “El evento Carnaval se realiza en la ciudad de Rio de Janeiro el dia 21/02/2017”.

Práctica: Información del día del evento

Vamos a informar al usuario de la ciudad y día de un evento con la frase que hemos creado en el ejercicio anterior. En la actividad `EventosWeb` añade un menú Android o un botón jQuery Mobile, lo que prefieras, que, al pulsarlo, hará que se cargue la página web creada en el ejercicio anterior del evento que estamos viendo.

5.1.3.8. Activadores de Pub/Sub de Cloud

Pub/Sub de Google Cloud es un servicio que permite enviar mensajes entre aplicaciones. Una aplicación puede enviar mensajes a un tema, y otras aplicaciones pueden suscribirse a dicho tema para recibirlos. Utilizamos `functions.pubsub` para crear funciones que controlen eventos `Pub/Sub` de Google Cloud.

El evento disponible es:

Evento	Descripción
`onPublish`	Se envía un mensaje

Cloud Functions admite el evento `Message`. Este evento se activa cada vez que un mensaje de Pub/Sub se envía a un tema específico. Debemos especificar el nombre del tema que deseamos que active la función y configurar el evento dentro del controlador de eventos `onPublish()`:

```
exports.funcion = functions.pubsub.topic('tema').onPublish(event => {
  // Código a ejecutar
});
```

Podemos acceder a la carga útil del mensaje de Pub/Sub desde `event.data` como un objeto `Message`. En el caso de los mensajes con JSON en el cuerpo del mensaje Pub/Sub, el SDK de Firebase para Cloud Functions tiene una propiedad de ayuda que decodifica el mensaje. Por ejemplo, a continuación, se muestra un mensaje publicado con una carga útil JSON simple:

```
gcloud beta pubsub topics publish new_topic '{"name":"Xenia"}'
```

Podemos acceder a una carga de datos JSON como la anterior a través de la propiedad `json`:

```
const pubSubMessage = event.data;
let name = null;
try {
   name = pubSubMessage.json.name;
} catch (e) {
   console.error('El mensaje PubSub no está en JSON', e);
}
```

Las cargas útiles que no son JSON se incluyen en el mensaje de Pub/Sub como un `string` codificado en base 64 en `event.data`. Para leer un mensaje como el siguiente, debemos decodificar el `string` codificado en base 64 como se muestra:

```
gcloud beta pubsub topics publish new_topic 'MyMessage'
const pubSubMessage = event.data;
const messageBody = pubSubMessage.data ?
             Buffer.from(pubSubMessage.data, 'base64').toString() : null;
```

El mensaje de Pub/Sub se puede enviar con atributos de datos configurados en el comando de publicación. Por ejemplo, podríamos publicar un mensaje con un atributo `name`:

```
gcloud beta pubsub topics publish new_topic --attribute name=Xenia
```

Podemos leer estos atributos desde `event.data.attributes`:

```
const pubSubMessage = event.data;
const name = pubSubMessage.attributes.name
```

Preguntas de repaso: Firebase Functions

5.2. Enlaces dinámicos

5.2.1. Dynamic Links

Cuando nos pasan un enlace a un vídeo de YouTube, la forma habitual de abrirlo es a través del navegador Web. Sin embargo, es posible que tengamos instalada la aplicación de YouTube, que nos aportaría una mejor experiencia de usuario. Gracias a Dynamics Links, vamos a poder definir URL asociados a nuestra aplicación, de forma que el usuario pueda abrir contenido. En caso de no tenerla instalada se le puede sugerir que la instale, o si no está interesado, o estás en un sistema no Android, abrir el contenido desde la Web.

Firebase Dynamic Links son URL inteligentes que cambian su comportamiento de forma dinámica para proporcionar la mejor experiencia en diferentes plataformas.

Un mismo enlace puede apuntar a contenidos direfentes según la plataforma en la que es abierto. Si un usuario abre un enlace dinámico y no tiene la aplicación instalada, se le puede solicitar que la instale; luego, después de la instalación, se iniciará la aplicación y podrá acceder al vínculo. Cuando un usuario abre un Dynamic Link, si aún no tiene instalada la aplicación, se le dirige a Play Store o App Store para que la instale (a menos que especifiquemos algo diferente), y luego se abre la aplicación. Entonces, podremos recuperar el vínculo que se le pasó a la aplicación.

Los enlaces dinámicos son duraderos y sobreviven a las instalaciones de la aplicación. Evitan perder conversiones cuando los usuarios potenciales aún no tengan la aplicación instalada. Funcionan en iOS, Android y en aplicaciones web de escritorio y móvil.

Los usaremos en campañas de comercialización y para compartir contenido a fin de saber exactamente qué campañas y qué contenido están impulsando el crecimiento. Los análisis en la consola de Firebase permiten comprender qué vínculos generan la instalación y el uso de la aplicación.

Usaremos Dynamic Links en promociones físicas, web, por correo electrónico, en redes sociales y de recomendación para aumentar la captación y retención de usuarios. Combinaremos su uso con Firebase Analytics.

Funciona con o sin la aplicación instalada. Si es necesario que esté instalada para abrir un vínculo, la aplicación abre automáticamente el vínculo después de la instalación. Si es necesario actualizar la aplicación para abrir un vínculo, el vínculo se abrirá automáticamente después de actualizarse. Esta característica solo está disponible en Android.

Podemos llevar a los usuarios directamente a contenido vinculado dentro de la aplicación. Los vínculos se abren directamente en la aplicación, sin necesidad de diálogo de desambiguación ni rebote a través de un navegador.

Consultamos un análisis básico de cada enlace en el panel de Dynamic Links de la consola de Firebase o uno avanzado usando Firebase Analytics.

Podemos crear un Dynamic Link desde la consola de Firebase o desde programación.

El uso de Dynamic Link es gratuito e ilimitado.

Veamos cómo crear un enlace dinámico en la consola de Firebase.

Ejercicio: Crear un vínculo dinámico en Firebase

1. Abre el proyecto Eventos en la consola de Firebase.

   Si realizas un nuevo proyecto, asegúrate de haber proporcionado la huella digital SHA1 de la aplicación Android en la página de configuración del proyecto.

2. Abre la página de Dynamic Links en el menú de la barra lateral.

   En esta página podrás crear y ver tus enlaces creados, así como su tasa de clics.

3. Para crear el primer Dynamic Link haz clic en el botón **Nuevo vínculo dinámico**.

4. No vamos a personalizar el URL corto que nos propone Google para el vínculo dinámico. Pulsa en el botón **Siguiente**.

5. Proporciona la información básica de un enlace dinámico:

   URL de vínculo directo: https://eventos-xxxx.firebaseapp.com/

   Nombre: Vinculo básico

   Un vínculo dinámico es un vínculo directo a una aplicación. Funciona sin importar si está instalada. En los ordenadores de escritorio, se dirigirá al URL de vínculo directo. Será el vínculo que abrirá tu aplicación. El vínculo debe ser un URL válido que utilice el esquema HTTPS o HTTP.

   Nombre del vínculo dinámico, será al que harás referencia cuando realices el seguimiento de datos, como los clics en este vínculo

6. Pulsa en el botón **Siguiente**.

7. Nos pide que definamos el comportamiento en iOS. Dejarás la opción por defecto, **Abrir el vínculo directo en un navegador**.

   Existe la opción de abrir el vínculo en la aplicación iOS.

8. Pulsa en el botón **Siguiente**.

9. En la siguiente pantalla, tendremos que definir el comportamiento en Android. Marca la opción **Abrir el vínculo directo en la app de Android**.

10. Nos pedirá que elijamos el paquete de la aplicación que deseamos que abra el vínculo. Selecciona `org.example.eventos`.

    Al igual que en iOS tenemos la posibilidad de elegir cómo se va a comportar el vínculo cuando sea abierto por un dispositivo Android. En iOS hemos elegido que se abra en un navegador, por lo tanto, se abrirá el URL que hemos indicado en el vínculo directo, aunque la aplicación esté instalada en el dispositivo. Sin embargo, en Android hemos seleccionado la opción para que la aplicación sea la que abra el vínculo.

11. Elige la opción **App instantánea o URL personalizada** en "Si no está instalada la app, envía el usuario a".

    Hemos indicado que la aplicación sea la que abra el vínculo. Pero tenemos que señalar cómo se va a comportar el vínculo si el usuario no tiene la aplicación instalada. Tenemos dos opciones:

    - **Página de Google Play para tu app:** envía al usuario a Google Play para que instale la aplicación.
    - **App instantánea o URL personalizado:** tenemos la posibilidad de enviar al usuario a un URL determinado o bien, si la aplicación se configuró con Apps instantáneas, este vínculo abrirá y ejecutará la aplicación nativa de forma instantánea, sin necesidad de realizar ninguna instalación.

12. Introduce el siguiente URL en el campo de App instantánea o URL personalizado: https://www.androidcurso.com/

13. Haz clic en el botón **Siguiente**.

14. De forma opcional, podemos unir el vínculo a una campaña para poder seguirla y evaluar los resultados. No lo vamos a utilizar.
15. Pulsa en el botón **Siguiente**.
16. Puedes personalizar el vínculo para cuando se comparta mediante las redes sociales. Puedes introducir un título, una descripción y una imagen, que acompañarán al vínculo en las redes sociales. No lo vamos a utilizar.
17. Pulsa en **Crear vínculo dinámico**.
18. Copia el URL del vínculo que se ha creado y que obtendrás en la columna URL de la lista de vínculos creados.
19. Comparte el vínculo y ábrelo en tu ordenador y en un dispositivo Android.

Una forma bastante fácil es autoenviarse un e-mail a la misma cuenta que se ejecuta en el dispositivo Android. A estas cuentas de correo, normalmente, podemos acceder desde el dispositivo Android y desde el ordenador. Mediante el envío del e-mail, simulamos que el usuario ha recibido el vínculo como parte de una campaña de e-mail marketing.

Abre el vínculo en tu ordenador desde el e-mail que te has enviado. Comprueba que abre el URL de vínculo directo. Luego abre el vínculo desde el dispositivo Android donde tienes la aplicación "Eventos" instalada, comprueba que se abre la aplicación. Desinstala la aplicación en el dispositivo y vuelve a abrir el vínculo, observa que se abre el URL www.androidcurso.com. Si la aplicación no está instalada, puedes indicar qué acción se va a realizar al abrirlo. Lo normal, es pedir al usuario que instale la aplicación redirigiéndolo a Google Play.

Una de las características más importantes de los vínculos dinámicos es que podemos dirigir al usuario una parte concreta de la aplicación o mostrar un contenido concreto. Incluso si el usuario no tiene instalada la aplicación, cuando se la instale, el usuario será redirigido al contenido concreto dentro de la aplicación cuando la abra.

Por ejemplo, supongamos que enviamos una campaña de e-mail sobre el evento "Fallas". Introducimos en el e-mail un vínculo dinámico para que los usuarios que lo pulsen vean la información del evento en nuestra aplicación.

Vamos a modificar la aplicación para que esto se produzca y crearemos un vínculo en Firebase para que muestre un evento concreto.

Ejercicio: Abrir un vínculo dinámico en una actividad

1. Añade el siguiente filtro en la declaración de la actividad `EventosDetalles` en el manifiesto:

```
<intent-filter>
   <action android:name="android.intent.action.VIEW"/>
   <category android:name="android.intent.category.DEFAULT"/>
   <category android:name="android.intent.category.BROWSABLE"/>
```

```
   <data android:host="eventos-xxxx.firebaseapp.com"
                                                android:scheme="https"/>
</intent-filter>
```

Sustituye `eventos-xxxx.firebaseapp.com` por la dirección de Firebase Hosting de la aplicación. Debes declarar este filtro en cada una de las actividades que quieras que se abran desde un vínculo dinámico. La declaración de `action` y `category`, la tiene que realizar tal y como indica el paso del ejercicio. En la declaración de `data` es donde vas a personalizar la declaración. Lo que va a recibir la aplicación va a ser el URL que se introduzca en el URL de vínculo directo. El URL declarado en nuestro ejemplo es la siguiente: `https://eventos-xxxx.firebaseapp.com`. Distinguimos el dominio (`eventos-xxxx.firebaseapp.com`) y el esquema (`https`).

2. Si has desinstalado la aplicación, vuelve a instalarla. Ejecuta la aplicación y ciérrala. Si no la has desinstalado, revisa que la aplicación se encuentre cerrada.

3. Abre el vínculo del ejercicio anterior en el dispositivo.

   Revisa que cuando se abre la aplicación no se abre la actividad inicial, sino que se abre la actividad `EventosDetalles`. Muestra la actividad en blanco, ya que no le hemos indicado que muestre ningún evento. Lo resolveremos más adelante.

   Podemos abrir distintas actividades según el URL que enviemos a la aplicación. En la declaración de cada actividad en el manifiesto, deberemos declarar filtros distintos. Por ejemplo, si tenemos una aplicación de cocina con recetas paso a paso y vídeos, podríamos declarar un filtro para el dominio `recetas.miaplicacion.com` para la actividad de las recetas y otro filtro `videos.miaplicacion.com` para la actividad que muestra los vídeos. También deberemos declarar el esquema, es decir, si la llamada se va a hacer mediante HTTP o HTTPS. El parámetro data del filtro de recetas quedaría de la siguiente forma: `<data android:host="recetas.miaplicacion.com" android:scheme="http"/>`. Si las recetas recibieran URL HTTPS y HTTP, podríamos declarar más parámetros dato en el mismo filtro. Añadiríamos el filtro del ejemplo pero con `android:scheme="https"`.

Ejercicio: Abrir un vínculo dinámico con sobrecarga de datos

1. Crea un vínculo dinámico desde la consola de Firebase con las mismas opciones que el vínculo creado anteriormente, pero con los siguientes cambios:

   URL de vínculo directo: https://eventosz-xxxx.firebaseapp.com?evento=fallas

   Nombre: Vinculo con sobrecarga

2. Una vez hayas creado el vínculo, reenvíatelo mediante e-mail.

3. Sustituye, en el método `onCreate` de la clase `EventoDetalles`, la instrucción:

```
if (evento==null) evento = "";
```

por:

```
if (evento==null) {
    android.net.Uri url = getIntent().getData();
    evento= url.getQueryParameter("evento");
}
```

4. Ejecuta la aplicación en el dispositivo y ciérrala.
5. Pulsa sobre el vínculo “Vínculo con sobrecarga” que acabas de enviarte por e-mail.

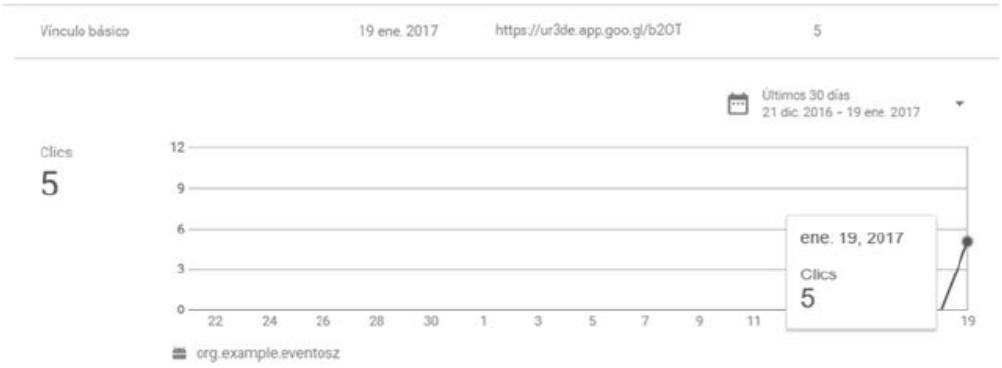
Comprueba que se abre la aplicación y se muestra directamente la información sobre el evento “Fallas”. Si un usuario no tuviera instalada la aplicación, lo podríamos redirigir a Google Play para que la instalara. Cuando la terminara de instalar y la abriera, mostraría el evento “Fallas”, ya que la instalación se realizó mediante un vínculo dinámico.

Hemos indicado que Dynamic Links se integra con Firebase Analytics. Por eso, sin realizar ninguna programación especial, podemos disponer de una estadística de clics sobre el enlace. Pulsa sobre un enlace en la lista que aparece en Dynamic Links.

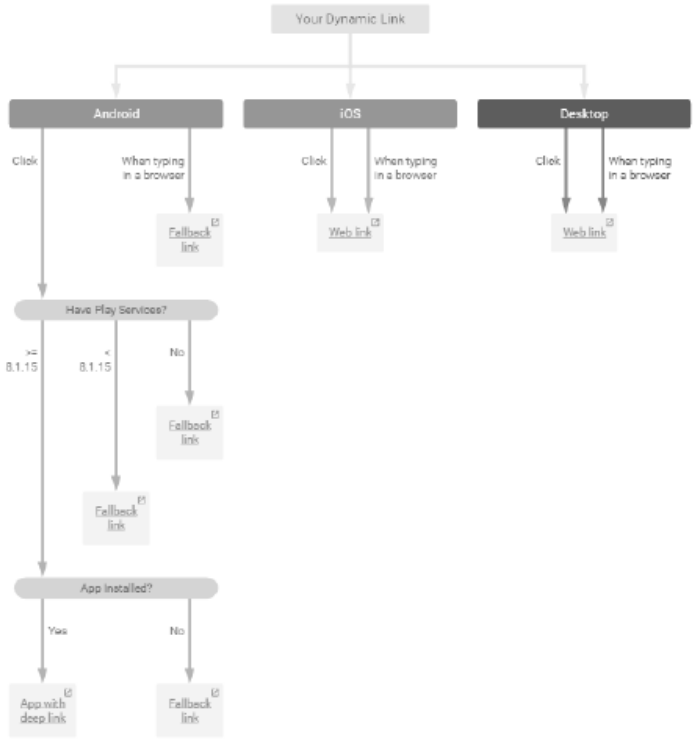
Para cada vínculo de la lista, podemos ver la siguiente información utilizando su menú individual. Tenemos tres elementos en el menú:

- **Detalles del vínculo:** Muestra la información del vínculo: nombre, enlace directo, el URL del vínculo dinámico corto y largo…

- **Estadísticas del vínculo:** Muestra los clics que se han hecho sobre el vínculo de forma temporal.



- **Flujo de vínculos:** Revisa el flujo del vínculo dependiendo del sistema operativo del dispositivo donde abra el vínculo el usuario, de la versión de Google Play Services, de si la aplicación está instalada… Además, podemos verificar los links de cada caso.



Si observas los detalles del vínculo, verás que nos muestra un “Vínculo dinámico corto” (es el que hemos utilizado hasta ahora, el que hemos enviado por e-mail) y “Vínculo dinámico largo”. Es el vínculo dinámico largo el que procesa Android, pero es encapsulado en un vínculo corto.

Firebase ha encapsulado el siguiente vínculo: https://ur3de.app.goo.gl/?link=https://eventosz-aacec.firebaseapp.com?evento%3Dfallas&apn=org.example.eventosz&afl=https://www.androidcurso.com/ en el siguiente vínculo corto: https://ur3de.app.goo.gl/PeSx. También, podemos crear un Dynamic Link mediante programación. Crearíamos un vínculo largo. Debemos construir un URL con la siguiente forma:

```
https://domain/?link=link&apn=package_name[&amv=minimum_version][&ad=1]
                [&al=android_link][&afl=fallback_link]
```

- *domain*: El dominio Dynamic Links del proyecto en Firebase.
- link: El vínculo que abrirá la aplicación. Podemos especificar cualquier URL compatible con la aplicación, como un vínculo al contenido o un URL que inicie una lógica específica. Este enlace debe ser un URL con formato correcto, debe estar correctamente codificado y debe usar un esquema HTTP o HTTPS.
- apn: El nombre del paquete de la aplicación Android que abrirá el vínculo. La aplicación debe estar conectada al proyecto de Firebase. Es obligatorio para que Dynamic Links abra la aplicación Android.
- amv: (opcional) La versionCode de la versión mínima de la aplicación que puede abrir el vínculo. Si la aplicación instalada es una versión anterior, se dirige al usuario a Play Store para que la actualice.
- ad (opcional) Declara que el Dynamic Link se está utilizando en un anuncio.
- al: (opcional) El vínculo que se debe abrir en Android. Este vínculo puede usar cualquier esquema y, si se especifica, tiene prioridad sobre el parámetro link en Android.
- afl: (opcional) El vínculo que se debe abrir cuando no esté instalada la app. Se usa para especificar que haga algo diferente de instalar la aplicación desde Play Store cuando no esté instalada, como abrir la versión web móvil del contenido o mostrar una página promocional para tu app.
- utm_source, utm_medium, utm_campaign, utm_term, utm_content, gclid: (opcional) Parámetros de una campaña personalizada. Estos parámetros se pasan a vínculos profundos para que la aplicación registre resultados para su análisis.

Hasta el momento hemos visto cómo enviar vínculos dinámicos desde la consola de Firebase que pueden ser abiertos por la aplicación para mostrar un contenido específico. También se pueden crear vínculos dinámicos mediante programación que podrán ser creados desde fuentes externas a la consola de Firebase, por ejemplo, un servidor propio.

Práctica: Campaña de descuento en evento

Imaginemos que realizamos una campaña de marketing por e-mail en la que se ofrecen descuentos para asistir a un evento. Enviaremos un e-mail con un enlace dinámico para el evento "Fallas" con un descuento del 25 %. Al pulsar sobre el enlace se abrirá la actividad EventosWeb de la aplicación, donde se mostrará la

información del evento y el descuento conseguido. Deberás mostrar el descuento en la aplicación web de la actividad.

5.2.2. Firebase Invites

Los vínculos dinámicos otorgan un extra a nuestra aplicación para captar y retener usuarios. Pero, está demostrado que la mejor forma de que un usuario instale una aplicación es mediante una recomendación de un familiar o un conocido. Podemos potenciar esta alternativa de promoción con otra funcionalidad de Firebase llamada Firebase Invites.

Firebase Invites es una solución multiplataforma para enviar invitaciones personalizadas por correo electrónico y SMS, incorporar usuarios y medir el impacto de las invitaciones.

El boca a boca es una de las formas más eficaces de lograr que los usuarios instalen nuestra aplicación. Firebase Invites facilita la conversión de los usuarios de una aplicación en sus principales promotores.

Firebase Invites se basa en Firebase Dynamic Links. Si bien Dynamic Links garantiza que los destinatarios de vínculos tengan la mejor experiencia posible en su plataforma y las aplicaciones que tienen instaladas, Firebase Invites garantiza la mejor experiencia posible en el envío de vínculos.

Cuando un usuario presiona un botón para compartir la aplicación y selecciona el canal de Firebase Invites —generalmente llamado "Correo electrónico y SMS"— se abre la pantalla para compartir. En pantalla, el usuario selecciona destinatarios entre sus contactos, puede personalizar el mensaje de la invitación, y envía las invitaciones. Las invitaciones se envían por correo electrónico o SMS, según la información de contacto disponible y contienen un Dynamic Link a la aplicación.

Cuando los destinatarios de la invitación abren el Dynamic Link de la invitación, se los envía a la Play Store en caso de que necesiten instalar la aplicación; luego, se abre la aplicación y puede recuperar y controlar el vínculo.

Procedamos a modificar la aplicación “Eventos” para que permita enviar invitaciones y procesar las invitaciones recibidas.

Ejercicio: Enviar invitaciones con Firebase Invites

1. Conecta Firebase Invites con la aplicación. Añade la siguiente dependencia en el fichero build.gradle (Module:app):

```
implementation 'com.google.firebase:firebase-invites:11.8.0'
```

2. Declara el cliente `GoogleApiClient` para conectar con Firebase y el código de la respuesta al lanzar la actividad de la invitación en la clase `ActividadPrincipal`:

```
private GoogleApiClient mGoogleApiClient;
private static final int REQUEST_INVITE = 0;
```

3. Inicializa `GoogleApiClient` con la API de Firebase Invites, añade el siguiente código al final del método `onCreate` de la clase `ActividadPrincipal`:

```
mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addApi(AppInvite.API)
        .enableAutoManage(this, this)
        .build();
```

4. Añade la siguiente implementación a la clase `ActividadPrincipal`:

```
implements GoogleApiClient.OnConnectionFailedListener
```

5. Inserta el siguiente método al final de la clase `ActividadPrincipal`:

```
@Override
public void onConnectionFailed(ConnectionResult connectionResult) {
   Toast.makeText(this, "Error al enviar la invitación", Toast.LENGTH_LONG);
}
```

6. Inserta el siguiente elemento de menú en `menu_principal`, delante del elemento `action_temas`:

```
<item
   android:id="@+id/action_invitar"
   android:orderInCategory="50"
   android:title="Invitar"
   app:showAsAction="always" />
```

7. Inserta dentro del método `onOptionsItemSelected` de la clase `ActividadPrincipal`, delante de `return super.onOptionsItemSelected(item);`:

```
if (id == R.id.action_invitar){
   invitar();
}
```

8. Agrega las siguientes cadenas al fichero `strings.xml` que encontrarás en la carpeta `values` en recursos:

```
<string name="invitation_title">
                    Envio de una invitación de la app Eventos</string>
<string name="invitation_message">
                 No te pierdas nunca más un evento interesante.</string>
<string name="invitation_deep_link">
                 https://principal.eventos-xxxx.firebaseapp.com</string>
<string name="invitation_custom_image">
https://www.google.com/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png
</string>
<string name="invitation_cta">Instalar!!</string>
```

Cambia `eventos-xxxx` por el dominio de tu aplicación en Firebase. Vamos a distinguir entre los vínculos dinámicos enviados desde la consola de Firebase, donde el enlace contiene la información de un evento, y las invitaciones, en las que simplemente vamos a precisar que se abra la aplicación. En el primer caso, se abrirá la actividad `EventoDetalles` y en el caso de las invitaciones, se abrirá la actividad `ActividadPrincipal`. Para ello tendremos que especificar dos filtros diferentes a definir en la declaración de las actividades en el manifiesto. Distinguiremos dos `host` diferentes, para `EventoDetalles` será `eventos-xxx.firebaseapp.com` y para `ActividadPrincipal` será `principal.eventos-xxxx.firebaseapp.com`.

9. Inserta el siguiente filtro en la declaración de `ActividadPrincipal` en el manifiesto:

```
<intent-filter>
   <action android:name="android.intent.action.VIEW"/>
   <category android:name="android.intent.category.DEFAULT"/>
   <category android:name="android.intent.category.BROWSABLE"/>
   <data android:host="principal.eventos-xxxx.firebaseapp.com"
         android:scheme="http"/>
   <data android:host="principal.eventos-xxxx.firebaseapp.com"
         android:scheme="https"/>
</intent-filter>
```

10. Añade la final de la clase `ActividadPrincipal`:

```
private void invitar() {
   Intent intent = new AppInviteInvitation.IntentBuilder(getString(
                                          R.string .invitation_title))
   .setMessage(getString(R.string.invitation_message))
   .setDeepLink(Uri.parse(getString(R.string.invitation_deep_link)))
   .setCustomImage(Uri.parse(getString(R.string.invitation_custom_image)))
   .setCallToActionText(getString(R.string.invitation_cta))
   .build();
   startActivityForResult(intent, REQUEST_INVITE);
}
@Override
protected void onActivityResult(int requestCode, int resultCode,
                                                        Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   if (requestCode == REQUEST_INVITE) {
     if (resultCode == RESULT_OK) {
       String[] ids=AppInviteInvitation.getInvitationIds(resultCode,data);
     } else {
       Toast.makeText(this, "Error al enviar la invitación",
                                                    Toast.LENGTH_LONG);
     }
   }
}
```

11. Ejecuta la aplicación.

Comprueba que puedes enviar una invitación desde el menú **INVITAR** en la pantalla inicial. Al pulsar sobre el botón **INVITAR** aparecerá una nueva pantalla donde podrás seleccionar a cuál de tus contactos enviar la invitación. Si te fijas, apareceren tanto los correos electrónicos como los números de teléfono, ya que la invitación se puede enviar por e-mail o por SMS. Introduce tu dirección de correo electrónico y pulsa enviar. Revisa tu bandeja de entrada. Abre el e-mail que has recibido con el asunto "No te pierdas nunca más un evento interesante" y pulsa en el enlace "Install this aplication". Comprueba que se abre la aplicación.

En resumen, hemos conseguido enviar un vínculo dinámico con el propósito de dar a conocer nuestra aplicación de una forma más eficiente.

Un usuario puede que envíe una invitación más fácilmente si tiene una motivación. Por ejemplo, un descuento. Para ello necesitamos pasar la información del descuento en el URL del enlace dinámico, al igual que hemos hecho con el enlace que hemos creado en la consola de Firebase que nos permitía abrir un evento en concreto. Para ello, necesitaremos tratar la invitación que nos llega. Será tratada cuando se abra la aplicación, incluso al abrirla después de instalarla.

Ejercicio: Enviar invitaciones con sobrecarga de datos

1. Modifica el siguiente `string` en strings.xml, que encontrarás en la carpeta values, en los recursos, para que contenga el parámetro descuento:

```
<string name="invitation_deep_link">
   https://principal.eventos-xxxx.firebaseapp.com?descuento=15</string>
```

2. Inserta el siguiente código al final del método `onCreate` de la clase `ActividadPrincipal`:

```
boolean autoLaunchDeepLink = true;
AppInvite.AppInviteApi.getInvitation(mGoogleApiClient, this,
                                                    autoLaunchDeepLink)
   .setResultCallback(
      new ResultCallback<AppInviteInvitationResult>() {
         @Override
         public void onResult(AppInviteInvitationResult result) {
            if (result.getStatus().isSuccess()) {
               Intent intent = result.getInvitationIntent();
               String deepLink = AppInviteReferral.getDeepLink(intent);
               String invitationId = AppInviteReferral
                                                .getInvitationId(intent);
               android.net.Uri url = Uri.parse(deepLink);
               String descuento = url.getQueryParameter("descuento");
               mostrarDialogo(getApplicationContext(),
```

```
                                        "Tienes un descuento del "+descuento
                                +"% gracias a la invitación: "+ invitationId);
                }
            }
        });
```

3. Ejecuta la aplicación y envía una invitación a tu e-mail.
4. Cierra la aplicación.
5. Pulsa sobre el enlace que has recibido en el e-mail de invitación.

   Comprueba que al pulsar sobre el enlace y abrir la aplicación, se muestra un mensaje que indica que se ha conseguido un 15 % de descuento gracias a un identificador de invitación. Faltaría procesar la invitación con el descuento. Es decir, que el usuario pueda aplicarse el descuento en la próxima compra y que no se pueda beneficiar más de una vez.

   Cuando un usuario recibe una invitación, si aún no ha instalado la aplicación, podrá hacerlo desde Google Play Store. Una vez la haya instalado, o si ya estaba instalada, se inicia y recibe el URL de su contenido, si envió una. Para recibir el URL que apunta al contenido de la aplicación, llamamos al método `getInvitation`. Se requiere un `GoogleApiClient` conectado con `AppInvite.API` habilitada.

   Si el parámetro `launchDeepLink` es `true`, la aplicación se reinicia automáticamente con el URL que lleva al contenido de la aplicación. Si el parámetro `launchDeepLink` es `false`, puedes iniciar manualmente la intención devuelta por `getInvitationIntent` para administrar el URL cuando sea necesario.

   Debemos llamar a `getInvitation()` en cada una de las actividades que el vínculo puede iniciar. El vínculo podría estar disponible en la intención mediante el uso de `getIntent().getData()`. La llamada a `getInvitation()` recupera el vínculo y el identificador de la invitación, y borra esos datos, de modo que la aplicación solo los procese una vez.

   Generalmente, se llama a `getInvitation()` en la actividad principal y en las actividades iniciadas por los filtros de intenciones que coinciden con el vínculo.

Preguntas de repaso: Enlaces dinámicos

5.3. Configuración remota con Firebase

Firebase Remote Config es un servicio en la nube que nos permite cambiar el aspecto y el comportamiento de una aplicación sin que los usuarios necesiten descargarla y actualizarla. Para usar Remote Config, debemos crear valores predeterminados en la aplicación. Luego, podemos usar la consola de Firebase

para cambiar estos valores predeterminados para todos o para unos segmentos de usuarios. La aplicación busca frecuentemente actualizaciones y las aplica con un impacto insignificante en el rendimiento.

Sus funciones clave son:

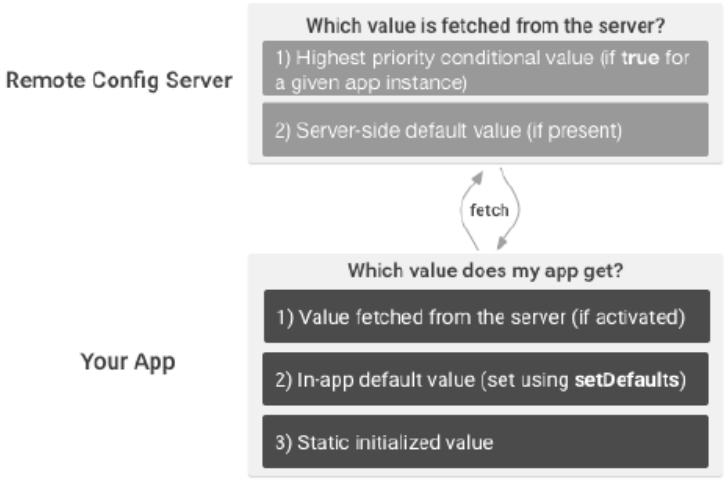
- **Implementación rápida de modificaciones:** Podemos realizar modificaciones en el aspecto y el comportamiento cambiando valores desde el servidor.
- **Personalización para segmentos de usuarios:** Podemos usar Remote Config para proporcionar variaciones en la aplicación para diferentes segmentos de usuarios a través de la versión de la aplicación, de Firebase Analytics, del idioma y más.
- **Ejecuta pruebas A/B:** Podemos usar un porcentaje aleatorio de usuarios con Firebase Analytics para realizar pruebas A/B para mejoras. De este modo podemos validarlas antes de implementarlas en todos los usuarios.

Remote Config realiza las tareas de obtención y almacenamiento en caché de valores y otorga el control sobre cuándo se activan los valores nuevos. Esto permite controlar cualquier modificación.

Los valores predeterminados se pueden obtener desde la aplicación o desde el servidor. Los métodos `get` de la biblioteca Remote Config proporcionan un único punto de acceso a los valores. La aplicación obtiene los valores del servidor usando la misma lógica que usa para obtener valores predeterminados de la aplicación.

Usamos Firebase console para crear parámetros. Utilizamos el mismo nombre en el servidor que en la aplicación. Para cada parámetro, podemos configurar un valor en el servidor para anular el valor en la aplicación. También podemos crear valores condicionales para instancias de la aplicación que cumplen ciertas condiciones.

El siguiente gráfico muestra cómo se priorizan los valores en el servidor y en la aplicación:



Los pasos para la implementación de Remote Config son:

- **Desarrollamos la aplicación usando parámetros:** Definimos qué aspectos del comportamiento y aspecto de la aplicación queremos poder cambiar usando Remote Config y los traducimos en parámetros.
- **Configuramos los valores de los parámetros predeterminados:** Utilizamos `setDefaults()` para configurar los valores predeterminados.
- **Agregamos la lógica para obtener, activar y conseguir los valores de los parámetros:** La aplicación puede obtener los valores de los parámetros de manera segura y eficiente desde el servidor. No debemos preocuparnos por cuál es el mejor momento para obtener los valores o incluso si existe algún valor en el servidor. La aplicación usa métodos `get` para obtener el valor de un parámetro, de forma similar a como se realiza la lectura del valor de una variable local.
- **Configuramos los valores de los parámetros condicionales y predeterminados del servidor:** Se pueden definir valores en la consola de Firebase para cambiar los valores predeterminados en la aplicación. Podemos hacer esto antes o después de lanzar la aplicación, porque los mismos métodos `get` acceden a valores predeterminados en la aplicación y a valores obtenidos desde el servidor.

Debemos tener en cuenta las siguientes políticas:

- No usaremos Remote Config para realizar las actualizaciones que requieren de una autorización del usuario. Esto puede hacer que la aplicación se perciba como poco confiable.
- No almacenaremos datos confidenciales en claves de parámetros de Remote Config o valores de parámetros. Es posible decodificar cualquier clave o valor de parámetro almacenado en la configuración de Remote Config.
- No intentaremos eludir los requisitos de la plataforma de destino de la aplicación usando Remote Config.

Cuando usamos Remote Config, definimos uno o más parámetros y proporcionamos valores predeterminados en la aplicación. Podemos anular los valores predeterminados en la aplicación mediante la definición de los valores en el servidor usando la consola de Firebase.

Al usar la consola de Firebase, podemos dar nuevos valores a los parámetros, así como establecer condiciones.

Se utiliza una condición para que se apliquen a un grupo de instancias de la aplicación. Las condiciones están compuestas de una o más reglas que debe cumplir una instancia. Un valor condicional consiste en una condición y un valor que se otorga a las instancias que cumplen esa condición. Un parámetro puede tener múltiples valores condicionales que usan condiciones diferentes. Los parámetros pueden compartir condiciones dentro de un proyecto.

Los valores de parámetros del servidor se obtienen de acuerdo a la siguiente lista de prioridades:

1. Primero, se aplican los valores condicionales, si alguna de las condiciones se cumple. Si se cumplen múltiples condiciones, la que se muestra más arriba en la consola de Firebase tiene prioridad. Podemos cambiar la prioridad de las condiciones arrastrando y soltando las condiciones en la pestaña **Condiciones**.
2. Si no hay o no se cumple ninguna condición, se proporciona el valor definido en el servidor. Si un parámetro no existe en el servidor o si el valor predeterminado está configurado en "Ningún" valor, entonces no se proporciona ningún valor.

Los valores de parámetros se obtienen a través de métodos `get` de acuerdo a la siguiente lista de prioridades:

1. Si un valor se obtuvo desde el servidor y luego se activó, la aplicación usa este valor. Los valores de parámetros activados son persistentes.
2. Si no se otorgó ningún valor desde el servidor o si no se activaron valores obtenidos desde el servidor, se usa el valor predeterminado en la aplicación.
3. Si no se configuró ningún valor predeterminado en la aplicación, se usa un valor tipo estático (como `0` para `int` y `false` para `boolean`).

Se admiten los siguientes tipos de condiciones:

- **Usuario en un percentil aleatorio:** Usamos este campo para aplicar un cambio en el porcentaje de instancias aleatorias que cumplan una cierta condición. Usamos los operadores <= y > para segmentar usuarios en grupos que no se superponen. A cada instancia de la aplicación se le asigna un número entero o fraccionario aleatorio dentro de un proyecto, de modo que podemos usar una regla de este tipo para abordar a las mismas instancias de la misma manera. Por ejemplo, para crear dos condiciones relacionadas y que cada una se aplique a un 0.05 % no superpuesto de la base de usuarios, podríamos tener una condición que incluya una regla de <= 005 % y otra condición que incluya una regla de >005 % y <= 0.10 %.
- **Tipo de SO:** Podemos aplicar una condición para las instancias que usen un determinado tipo de SO.
- **País/región de dispositivo:** Esta condición se cumple, si la instancia se encuentra en cualquiera de las regiones o países que se encuentran en la lista.
- **ID de la aplicación:** Seleccionamos una ID de la lista de aplicaciones asociadas con el proyecto de Firebase. Para aplicaciones Android, tiene que coincidir con `android:versionName` de la aplicación.
- **Versión de la aplicación:** Introducimos un valor para especificar una versión concreta (o versiones relacionadas). Antes de usar esta condición, debemos usar una regla de ID de la aplicación para seleccionar una

aplicación que esté asociada con nuestro proyecto Firebase. Tiene que coincidir con `android:versionCode`. Podemos proporcionar una lista de valores separados por comas para actuar sobre varias versiones.

- **Idioma del dispositivo:** Esta condición se cumple en las instancias cuyos dispositivos usen uno de los lenguajes que se encuentran en la lista.
- **Propiedad del usuario:** Seleccionamos una lista de usuarios usando Firebase Analytics. Esta regla requiere el ID de la aplicación en Firebase. Debido a que muchos usuarios Analytics se definen por eventos o por propiedades del usuario, que pueden estar basadas en acciones, puede tomar un tiempo que una condición tenga efecto.

Vamos a usar Remote Config en nuestra aplicación "Eventos":

Ejercicio: Configuración remota en Eventos

1. Agrega la dependencia para Remote Config en el archivo build.graddle [Module:app]:

```
implementation 'com.google.firebase:firebase-config:11.8.0'
```

2. Declara el objeto Remote Config y las variables que te permitirán cambiar el aspecto y comportamiento de la aplicación en la clase `Comun`:

```
static FirebaseRemoteConfig mFirebaseRemoteConfig;
static String colorFondo;
static Boolean acercaDe;
```

3. Inicializa el objeto Remote Config y declara las siguientes variables añadiendo el siguiente código al final del método `onCreate` en la clase `ActividadPrincipal`:

```
mFirebaseRemoteConfig = FirebaseRemoteConfig.getInstance();
FirebaseRemoteConfigSettings configSettings =
                              new FirebaseRemoteConfigSettings
                              .Builder()
                              .setDeveloperModeEnabled(BuildConfig.DEBUG)
                              .build();
mFirebaseRemoteConfig.setConfigSettings(configSettings);
```

`FirebaseRemoteConfig.getInstance()` se usa para almacenar los valores en la aplicación, obtener los valores actualizados desde el servidor y controlar cuando estarán disponibles los valores obtenidos en la aplicación. Mediante `FirebaseRemoteConfigSettings` habilitamos el modo para desarrolladores (`setDeveloperModeEnabled(BuildConfig.DEBUG)`) para permitir actualizaciones frecuentes de la caché.

4. Si no existe, crea una carpeta `xml` en recursos y añade un archivo llamado `remote_config_default.xml` con el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<defaultsMap>
   <entry>
      <key>color_fondo</key>
      <value>WHITE</value>
   </entry>
   <entry>
      <key>acerca_de</key>
      <value>true</value>
   </entry>
</defaultsMap>
```

El fichero xml contiene un mapa de entradas de pares clave/valor. Cada entrada determinará el valor por defecto de un parámetro para la aplicación si no lo obtiene del servidor.

5. Obtiene los valores predeterminados de la aplicación desde el fichero xml añadiendo al final del método `onCreate` de la clase `ActividadPrincipal` la siguiente instrucción:

```
mFirebaseRemoteConfig.setDefaults(R.xml.remote_config_default);
```

6. Copia al final del método `onCreate` de la clase `ActividadPrincipal`:

```
long cacheExpiration = 3600;
mFirebaseRemoteConfig.fetch(cacheExpiration)
   .addOnSuccessListener(new OnSuccessListener<Void>() {
       @Override
       public void onSuccess(Void aVoid) {
          mFirebaseRemoteConfig.activateFetched();
          getColorFondo();
          getAcercaDe();
       }
   })
   .addOnFailureListener(new OnFailureListener() {
      @Override
      public void onFailure(@NonNull Exception exception) {
         colorFondo=mFirebaseRemoteConfig.getString("color_fondo");
         acercaDe=mFirebaseRemoteConfig.getBoolean("acerca_de");
      }
   });
```

Podemos acceder a los valores de configuración al llamar a uno de los métodos disponibles `get<type>` (por ejemplo, `getString`) en el objeto `FirebaseRemoteConfig` que se pasa al `callback`.

Remote Config almacena los valores en la caché del dispositivo después de la primera solicitud con éxito. De forma predeterminada, la caché caduca después de 12 horas, podemos cambiar la caducidad predeterminada, indicando la caducidad deseada en `setCacheExpirationSeconds`. Si los

valores están en la caché un mayor tiempo que la caducidad deseada, Remote Config va a obtener los valores desde el servidor. Si la aplicación solicita valores actualizados al usar `setCacheExpirationSeconds` varias veces, las solicitudes se regulan y se le proporciona un valor en caché.

Durante el desarrollo de la aplicación, probablemente deseemos actualizar la caché con mucha frecuencia para que nos permita iterar rápidamente cuando desarrollamos y probamos la aplicación. Podemos agregar temporalmente un objeto `FirebaseRemoteConfigSettings` con `isDeveloperModeEnabled` configurado en `true`, lo cual modifica las configuraciones de almacenamiento en caché del objeto `FirebaseRemoteConfig`.

7. Añade al final de la clase `ActividadPrincipal` el siguiente código:

```
private void getColorFondo() {
   colorFondo = mFirebaseRemoteConfig.getString("color_fondo");
}
private void getAcercaDe() {
   acercaDe = mFirebaseRemoteConfig.getBoolean("acera_de");
}
```

8. Para cambiar el color de fondo del `WebView` en `EventosWeb`, añade al final del fichero `funciones.js` la siguiente función y actualiza el proyecto en Firebase Hosting:

```
function colorFondo(color){
   document.body.style.backgroundColor = color;
}
```

9. Añade delante de `return true;` , en el método `onCreateOptionsMenu` en la clase `EventoDetalles`, el siguiente código para mostrar u ocultar el menú **ACERCA DE**:

```
if (!acercaDe) {
   menu.removeItem(R.id.action_acercaDe);
}
```

10. Sustituye el método `onPageFinished` de la clase `EventosWeb` por el siguiente:

```
@Override
public void onPageFinished(WebView view, String url) {
   dialogo.dismiss();
   navegador.loadUrl("javascript:colorFondo(\""+colorFondo+"\")");
   navegador.loadUrl("javascript:muestraEvento(\""+evento+"\");");
}
```

Nos ha faltado programar el color de fondo. Para ello haremos una llamada a la función `colorFondo` con el color que queremos que se muestre.

11. Ejecuta la aplicación.

Comprueba que la aplicación se ejecuta correctamente. Como todavía no hemos definido ni los parámetros ni sus valores en la consola de Firebase, la aplicación coge por defecto los valores que hemos definido en `remote_config_default.xml`. Vamos a dar de alta los parámetros en la consola de Firebase y a comprobar cómo cambia el comportamiento y aspecto de nuestra aplicación.

12. En la consola de Firebase, seleciona el proyecto “Eventos” y accede al menú **REMOTE CONFIG**.
13. Pulsa sobre el botón **Agrega tu primer parámetro**.
14. Introduce la clave “acerca_de” con el valor “false”.
15. Pulsa en **Publicar** para que el parámetro sea accesible desde la aplicación.
16. Sal completamente de la aplicación y, sin volverla a instalar, vuélvela a ejecutar.

    Comprueba que ya no aparece del menú **ACERCA DE** cuando accedes a los detalles de un evento. Puede que el cambio tarde algún tiempo. Baja el tiempo en `setCacheExpirationSeconds` si tarda demasiado.
17. Vuelve a introducir una nueva clave llamada “acerca_de” con valor “true”.
18. Pulsa en **Publicar** para que los cambios sean visibles para la aplicación.
19. Vuelve a cerrar completamente y ejecutar la aplicación.

    Comprueba que ahora sí que aparece el menú **ACERCA DE** que no aparecía anteriormente.

    Acabamos de crear un parámetro en la consola de Firebase y le hemos asignado un valor. También hemos visto cómo cuando cambiamos el valor del parámetro, ha cambiado el comportamiento de la aplicación, mostrando y ocultando el menú **ACERCA DE**.

    Debes tener en cuenta que los valores de los parámetros son leídos del servidor cuando se inicia la aplicación y puede que haya expirado el tiempo que indicamos en `cacheExpiration`.

    Remote Config no solamente acepta pares parámetro/valor, sino que podemos definir condiciones para que cambie el valor de un parámetro.
20. En la consola de Firebase pulsa el botón **Añadir un parámetro** con el nombre “color_fondo”.
21. Pulsa sobre el texto “Agregar valor de condición” y en el diálogo que se abre pulsa sobre el texto “Definir una nueva condición” para añadir una condición.
22. Introduce “Idioma” en el nombre de la condición.
23. En “Se aplica si …”, selecciona “Device language”, selecciona “afar(aa)” y pulsa sobre el botón **Crear condición**.

    Ya hemos creado nuestra primera condición. Se va a cumplir si la aplicación se encuentra instalada en un dispositivo que utilice el idioma “afar(aa)” como

idioma. Ahora tenemos que definir el valor que tendrá el parámetro si se cumple la condición y qué valor tendrá por defecto.

24. Introduce el valor “YELLOW” en “Valor de Idioma” y “WHITE” como valor por defecto.

25. Pulsa en **Añadir parámetro** para terminar con el alta.

26. Pulsa el botón **Publicar cambios** para que se haga efectivo el uso del parámetro.

27. Si tienes abierta la aplicación, ciérrala y ejecuta la aplicación.

    Comprueba que el fondo del `WebVi ew` de nuestra aplicación es blanco, a menos que el idioma del dispositivo sea “afar(aa)”.

28. Edita el parámetro “color_fondo” y crea una nueva condición llamada “Idioma_Dispositivo” con el idioma del dispositivo en el que estás ejecutando la aplicación. El valor del parámetro para esta condición será “AZURE”.

29. Recuerda pulsar en **Publicar cambios** para que sean visibles para la aplicación.

30. Cierra y ejecuta la aplicación.

    Comprueba que el color del `WebVi ew` ahora es azul.

Hemos visto cómo cambiar el comportamiento y aspecto de una aplicación sin tener que modificar la programación y sin que los usuarios tengan que actualizar la aplicación.

Práctica: Configuración remota según país

Crea una pantalla de bienvenida (splash screen) que se mostrará al iniciar la aplicación. Se le dará la bienvenida al usuario a la aplicación y se le advertirá de que la aplicación mostrará eventos de España. Si el usuario se encuentra en España mostrará el mensaje “Esta aplicación muestra información sobre eventos” y si el usuario se encuentra en otro país, advertirá “Esta aplicación muestra eventos de España”. Prográmalo utilizando la configuración remota de Firebase.

Preguntas de repaso: Configuración remota

5.4. Firebase Stability

Vamos a ver una introducción al apartado Firebase Stability en la consola de Firebase. Está compuesta por los siguientes apartados: Crashlytics, Performance y Test Lab. No veremos Crash Reporting, que está siendo sustituido por Crashlytics. Este conjunto de herramientas está pensado para otorgar estabilidad a la aplicación. Se obtienen informes de errores y rendimiento directamente desde los usuarios que utilizan la aplicación. También para poder realizar un análisis de nuestra aplicación en diferentes dispositivos y configuraciones.

5.4.1. Crashlytics

Firebase Crashlytics es una herramienta de informes de fallos. Proporciona información sobre los problemas de una aplicación. Funciona en iOS y Android.

Es una herramienta ligera que informa de fallos en tiempo real, ayuda a realizar un seguimiento, priorizar y corregir problemas. Agrupa los fallos de forma inteligente y destaca las circunstancias en las que se produjeron, esto permite encontrar soluciones más rápidamente.

Descubre si un fallo afecta a muchos usuarios. Cuando la gravedad de un problema aumenta, podemos recibir alertas. Localiza qué líneas de código provocan los errores.

Las principales funciones de Crashlytics son:

- **Informes de fallos con datos seleccionados:** Crashlytics evalúa y sintetiza una gran cantidad de fallos en una lista de problemas manejable. Además, proporciona información del contexto en el que se produce el error. Destaca la gravedad y prevalencia de los fallos para determinar la causa que lo provoca más rápidamente.

- **Información mejorada del entorno:** Permite filtrar informes por sistema operativo, configuración de hardware, versión... Podemos descubrir si la aplicación falla más en dispositivos de un fabricante o en una orientación de pantalla.

- **Alertas en tiempo real:** Recibe alertas en tiempo real de problemas nuevos, recurrentes y crecientes. Son los que deberían requerir una atención más inmediata.

Para usar Crashlytics en una aplicación Android debemos seguir los siguientes pasos:

1. Conectamos la aplicación: Comenzamos por agregar Firebase a la aplicación.

2. Integramos el SDK: Agregamos las dependencias de Gradle y código para inicializar los informes de Crashlytics.

3. Verificamos los informes en la consola de Firebase: Utilizamos la consola de Firebase para identificar los problemas que afectan a los usuarios y descubrir sus causas.

5.4.1.1. Inicializar Crashlitycs en Android

Veamos como inicializar Crashlitycs en la consola de Firebase.

Ejercicio: Inicializar Firebase Crashlytics en la consola Firebase

1. Accede a la consola de Firebase con la cuenta con la que has creado el proyecto "Eventos".
2. Selecciona el proyecto "Eventos".
3. En el apartado STABILITY selecciona **Crashlytics**.
4. Pulsa en el botón **Configurar Crashlytics**.

   Crashlitycs va a ser el sustituto de Crash Reporting y se basa en Fabric.
5. Tienes que usar un asistente para inicializar Crashlitycs. En el primer apartado ¿ESTA APP ES NUEVA EN CRASHLYTICS? elige Sí, esta app es nueva en Crashlytics y no tiene ninguna versión del SDK.
6. En el segundo paso del asistente, INSTALA EL SDK, nos pide que instalemos el SDK en la aplicación Android. Para ello podrás pulsar en el botón **Ir a los documentos de Crashlytics** para leer la documentación de cómo hacerlo. Lo veremos en el siguiente ejercicio.
7. En el último apartado nos indica que para empezar a utilizar Crashlytics solo tienes que comenzar a utilizar la aplicación y esperar a que se produzca algún error.

   Veamos cómo configurar Crashlytics en una aplicación Android.

Ejercicio: Configurar Firebase Crashlytics en una app Android

1. Tienes que tener configurado Firebase en la aplicación Android. Si realizas este ejercicio en el proyecto "Eventos", ya lo tienes hecho de ejercicios anteriores.
2. En el fichero `build.gradle` del proyecto, agrega dentro de `repositories` en `buildscripts`:

```
maven {
   url 'https://maven.fabric.io/public'
}
```

3. En el mismo fichero gradle añade en `dependencies` en `buildscripts`:

```
classpath 'io.fabric.tools:gradle:1.+'
```

4. Sigue en el mismo archivo gradle. Inserta dentro de `repositories` en `allprojects`:

```
maven {
url 'https://s3.amazonaws.com/fabric-artifacts-private/internal-snapshots'
}
maven {
   url 'https://maven.fabric.io/public'
}
```

5. Inserta debajo de la línea `apply plugin: 'android-apt'` en `build.graddle` el siguiente `plugin`:

```
apply plugin: 'io.fabric'
```

6. Añade las siguientes dependencias en el fichero anterior:

```
compile('com.crashlytics.sdk.android:crashlytics:2.7.0-SNAPSHOT@aar') {
    transitive = true;
}
```

7. Agrega la cadena `useFirebaseAppId` a strings.xml:

```
<string name="com.crashlytics.useFirebaseAppId">true</string>
```

8. Cambia la vista del proyecto de Android a "Project Files" y crea un archivo app/fabric.properties con la línea siguiente:

```
USE_FIREBASE_APP_ID=true
```

9. Importa en la clase `ActividadPrincipal`, las clases `Crashlytics` y `Fabric`:

```
import com.crashlytics.android.Crashlytics;
import io.fabric.sdk.android.Fabric;
```

10. Inicializa Crashlytics añadiendo el siguiente código al final del método `onCreate()` de la clase `ActividadPrincipal`:

```
Fabric.with(this, new Crashlytics());
```

11. Inserta la siguiente opción de menú en el fichero `menu_principal.xml` que encontrarás en la carpeta `menu` de los recursos:

```
<item
    android:id="@+id/action_error"
    android:orderInCategory="101"
    android:title="Error"
    app:showAsAction="never" />
```

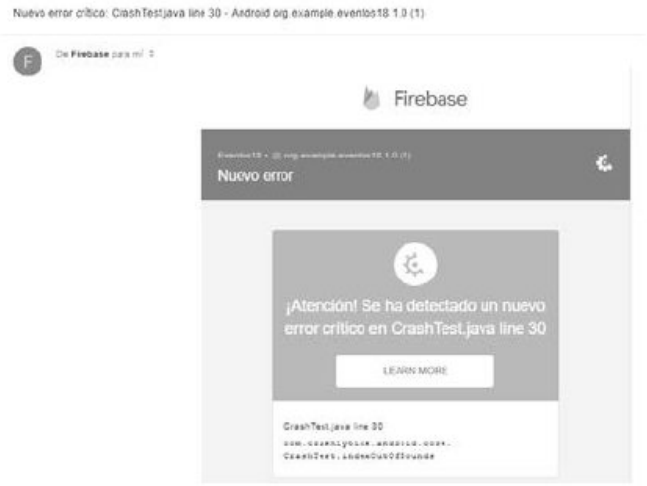
12. Añade debajo de `int id=item.getItemId();` del método `onOptionsItemSelected` en la clase `ActividadPrincipal` el siguiente código:

```
if (id == R.id.action_error) {
    Crashlytics.getInstance().crash();
    return true;
}
```

No es necesario esperar un fallo para saber si Crashlytics está funcionando. Puedes utilizar el método: `Crashlytics.getInstance().crash()` para forzar un fallo. En nuestro caso, lo provocaremos cuando seleccionemos el menú **ERROR** en la pantalla principal.

13. Ejecuta la aplicación y provoca un fallo.

Elige la opción **Error** en la pantalla principal para forzar un error. Comprueba que recibes casi inmediatamente un e-mail informando del error en la cuenta de desarrollo. El e-mail recibido será similar al de la imagen.



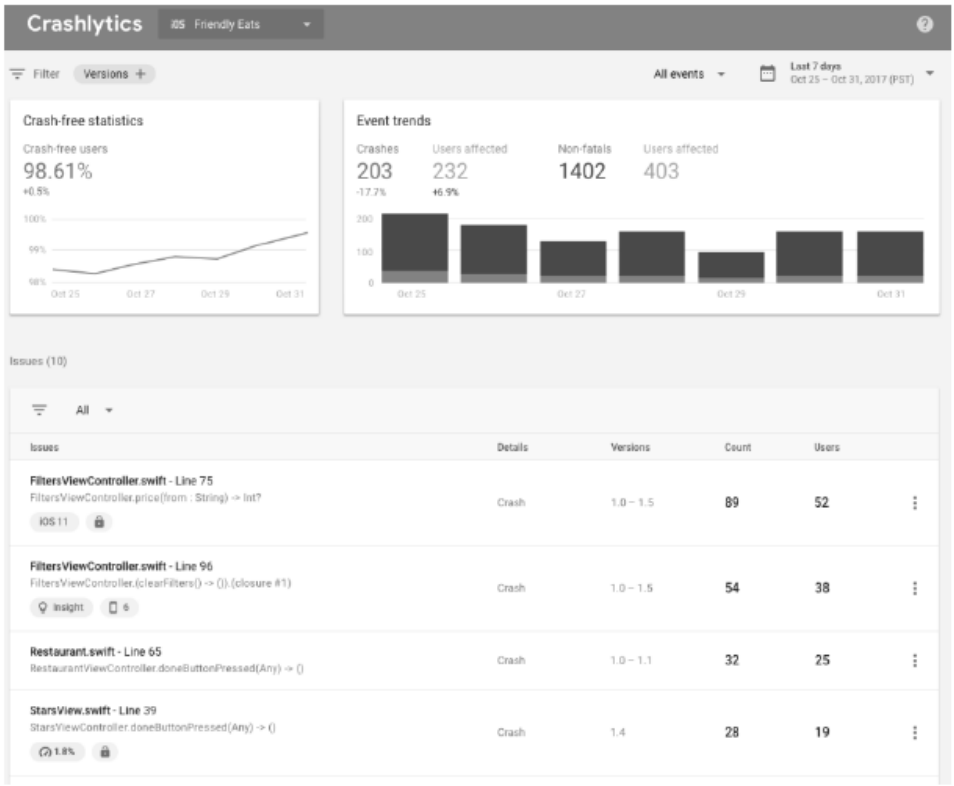
Si pulsamos en **Learn more**, accederemos a la consola de Firebase, directamente a la información del error. Allí podremos ver exactamente, además de otra mucha información, qué línea de que actividad es la que ha producido el error.

Desde la consola de Firebase podemos mantener un control de fallos solucionados. Cuando accedemos a los detalles de un error en la consola, tenemos el botón **Cerrar**. Si lo pulsamos, se cerrará la incidencia. Se dará como solucionada.

Si realmente no hemos solucionado el error, la podremos reabrir pulsando en el botón **Reabrir** o **Volver a abrir**.

Con Firebase Crashlytics podemos supervisar los informes de fallos desde la consola de Firebase c.

Consulta de los informes de errores desde la consola de Firebase. Abrimos la consola de Firebase y seleccionamos el proyecto de la aplicación “Eventos”. Seleccionamos “**Crashlytics**” en la barra de navegación del lado izquierdo.

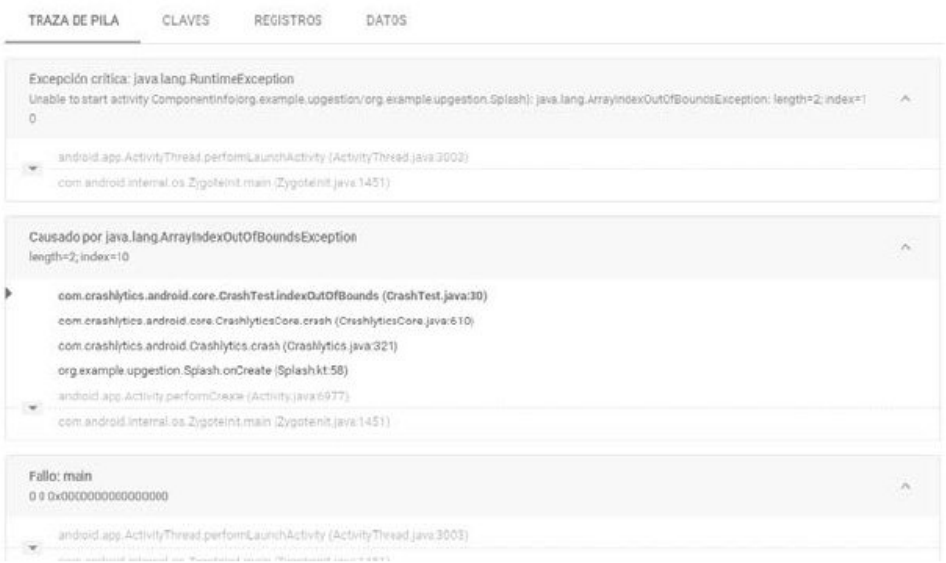


Tenemos tres paneles con información. Arriba a la izquierda, se encuentra el gráfico diario de “Estadísticas de la ausencia de fallos”, que informa del porcentaje de usuarios que no han tenido ningún fallo. Arriba a la derecha, está el gráfico diario de “Tendencias del evento”, que informa de los errores. Distingue entre errores de bloqueo y de no bloqueo, además de a cuántos usuarios ha afectado cada tipo de error. En la parte de abajo, tenemos hay una tabla con los distintos errores.

Podemos ver diferentes tipos de gráficos, como los que nos muestran el número de errores por versión de la aplicación, dispositivo, versión del sistema operativo y estado de los dispositivos (si se produce en primer o segundo plano).

Finalmente, podemos consultar las sesiones de cada error. Será donde obtendremos información de un mismo error, que se ha producido en diferentes usuarios, de forma unitaria. Nos informará desde las características del dispositivo, como puedes ver en la siguiente imagen.

Además de la traza del error, así obtendremos la misma información que se obtiene del Logcat de Android Studio, pero de cada usuario:



Recordemos quitar todas las líneas `Crashlytics.getInstance().crash();` antes de publicar la aplicación. Esta instrucción solo sirve para provocar errores y realizar pruebas.

Crashlytics comienza a recopilar informes de fallos automáticamente, solo tenemos que agregar el SDK. Podemos personalizar la configuración con informes opcionales, registros y claves, también podemos hacer seguimiento de errores recuperables.

5.4.1.2. Habilitar los informes de inclusión voluntaria

Por defecto, Firebase Crashlytics recopila automáticamente informes de fallos de todos los usuarios de la aplicación. Para dar a los usuarios más control sobre los datos que envían, podemos habilitar los informes de inclusión voluntaria. De esta forma solo se recopilarán datos de los usuarios que acepten participar.

Para activar los informes de inclusión automática en una aplicación Android, tenemos que deshabilitar la recopilación automática de fallos e inicializar Crashlytics solo para los usuarios que acepten participar.

Para desactivar la recolección automática, incluimos dentro de `application` en el manifiesto la siguiente etiqueta:

```
<meta-data android:name="firebase_crashlytics_collection_enabled"
                                                android:value="false" />
```

Habilitamos la recopilación de datos mediante la siguiente instrucción en alguna actividad de la aplicación:

```
Fabric.with(this, new Crashlytics());
```

En el ejercicio anterior ya hemos inicializado la recopilación de datos incluyendo esta instrucción en el método `onCreate` de la clase `ActividadPrincipal`. Así que, si quisiéramos activar la inclusión voluntaria, solamente deberían ejecutar la instrucción los usuarios que deseen participar.

Durante la ejecución de la aplicación no se puede detener la recopilación de datos para los informes de Crashlytics una vez que se haya aceptado participar. Para inhabilitar los informes después de inicializar Crashlytics, los usuarios deben reiniciar la aplicación y revocar la autorización a participar.

Práctica: Activar informes de inclusión voluntaria

Permite a los usuarios elegir si quieren participar en el envío de informe de fallos. Para ello, pregúntales mediante un cuadro de diálogo al iniciar por primera vez la aplicación lo siguiente:

“Con el fin de mejorar la aplicación, te pedimos que participes en el envío automático de errores a nuestros servidores. ¿Estás de acuerdo?”.

Si el usuario acepta, guardarás su respuesta en las preferencias y activarás la captura de fallos. Si no acepta, también guardarás su respuesta y no le volverás a preguntar.

5.4.1.3. Agregar registros personalizados

Para tener más información del contexto previo en el que se produce un error, podemos agregar registros de Crashlytics personalizados a la aplicación. Crashlytics asocia los registros con sus datos de bloqueo y los hace visibles en la consola de Firebase.

En Android, utilizamos el método `Crashlytics.log` para ayudar a identificar problemas. `Crashlytics.log` escribe registros en un informe de fallos, además, opcionalmente, puede mostrar la información en el `Log`:

- Para mostar información en Firebase Crashlytics y el `Log` utilizamos:

```
Crashlytics.log(int priority, String tag, String msg);
```

- Para mostrar información solamente en Firebase Crashlytics:

```
Crashlytics.log(String msg);
```

Para evitar ralentizar de la aplicación, Crashlytics limita los registros a 64 kB. Eliminaremos entradas de registro antiguas si los registros de una sesión sobrepasan ese límite.

5.4.1.4. Añadir claves personalizadas

Las claves personalizadas ayudan a obtener el estado específico de la aplicación hasta que ocurre un error. Podemos asociar pares de clave/valor arbitrarios a los informes y verlos en la consola de Firebase.

Hay cinco métodos para establecer claves. Cada uno maneja un tipo de datos diferente:

```
Crashlytics.setString ( key , value );
Crashlytics.setBool ( String key , boolean value );
Crashlytics.setDouble ( String key , double value );
Crashlytics.setFloat ( String key , float value );
Crashlytics.setInt ( String key , int value );
```

Veamos algún ejemplo para actualizar el valor de una clave:

```
Crashlytics.setInt("nivel_actual", 3);
Crashlytics.setString("ultima_accion_UI","menu_suscripciones");
```

Crashlytics admite un máximo de 64 pares clave/valor. Una vez que alcanza este umbral, los valores adicionales no se guardan. Cada par clave/valor puede tener un tamaño de hasta 1 kB.

5.4.1.5. Establecer ID de usuario

Para diagnosticar un problema, a menudo es útil saber cuál de los usuarios experimentó un bloqueo determinado. Crashlytics incluye una forma de identificar anónimamente a los usuarios en los informes de fallos.

Para agregar un ID de usuario, asignamos a cada usuario un identificador único. Puede ser un número, token o valor hash:

```
    void Crashlytics.setUserIdentifier(String identifier);
```

Si necesitamos borrar un identificador de usuario después de establecerlo, restablecemos el valor a una cadena en blanco.

5.4.1.6. Registrar excepciones no fatales

Además de informar automáticamente de errores, Crashlytics permite registrar excepciones no fatales. En Android, significa que puede registrar excepciones detectadas en los bloques `catch` de la aplicación.

Todas las excepciones registradas aparecen como problemas no fatales en la consola de Firebase. El resumen del problema contiene toda la información de estado que normalmente recibe de los bloqueos, como la versión de Android y el hardware del dispositivo.

Crashlytics procesa las excepciones en un hilo dedicado, por lo que el impacto en el rendimiento es mínimo. Para reducir el tráfico de red, Crashlytics combina las excepciones registradas y las envía la próxima vez que se inicia la aplicación.

Crashlytics solo almacena las 8 excepciones más recientes de una sesión. Si la aplicación arroja más de 8 excepciones en una sesión, se pierden las excepciones anteriores.

5.4.1.7. Administrar datos de Crash Insights

Crash Insights nos ayuda a resolver problemas al comparar seguimientos con los de otras aplicaciones de Firebase y nos permite saber si nuestro problema es parte de una tendencia más amplia. Para muchos problemas, Crash Insights incluso proporciona recursos para ayudar a depurar el bloqueo.

Crash Insights utiliza datos acumulados de bloqueos para identificar tendencias comunes de estabilidad. Si preferimos no compartir los datos de la aplicación, podemos cancelar la suscripción a Crash Insights en el menú de Crash Insights en la parte superior de la lista de problemas de Crashlytics en la consola de Firebase.

5.4.2. Performance

Obtengamos estadísticas útiles sobre el rendimiento y las latencias que experimentan los usuarios de una aplicación. Firebase Performance Monitoring es un servicio que ayuda a obtener estadísticas sobre las características de rendimiento de aplicaciones en iOS y Android. Usamos Performance Monitoring para recopilar datos de rendimiento y, luego, revisamos y analizamos los datos en la consola de Firebase. Nos ayuda a comprender dónde y cuándo se puede mejorar el rendimiento de una aplicación.

Medimos de forma automática el tiempo de inicio de la aplicación, las solicitudes de red HTTP/S y mucho más. Cuando integramos Performance Monitoring en una aplicación, no necesitamos escribir ningún código para que comience a supervisar varios aspectos críticos del rendimiento, como el tiempo de inicio, la actividad en primer plano, la actividad en segundo plano y las solicitudes de red HTTP/S.

Obtengamos estadísticas sobre las situaciones en las que se puede mejorar el rendimiento. La optimización del rendimiento puede ser un desafío cuando no sabemos exactamente por qué no se cumplen las expectativas del usuario. Por esto, Performance Monitoring permite ver métricas de rendimiento desglosadas por país, dispositivo, versión de la aplicación y sistema operativo.

Podemos personalizar Performance Monitoring para una aplicación. Se pueden crear seguimientos para registrar el rendimiento en situaciones específicas, como cuando cargamos una pantalla nueva. Además, podemos crear contadores para llevar un recuento de los eventos que definimos.

Un seguimiento es un informe de rendimiento capturado en un periodo de tiempo. Cuando se instala, Performance Monitoring proporciona seguimientos del inicio de forma automática, que miden el tiempo desde que el usuario abre la aplicación hasta que está lista para responder. Además, proporciona seguimientos en primer y segundo plano para mostrar estadísticas de rendimiento cuando está activa o inactiva.

Podemos configurar seguimientos personalizados. Un seguimiento personalizado es un informe de datos de rendimiento asociados con una parte del código de la aplicación. Define el inicio y el final de un seguimiento personalizado. Se puede configurar un seguimiento personalizado que contabilize el rendimiento de un determinado evento. Por ejemplo, podemos crear un contador de la cantidad de veces que la interfaz de usuario deja de responder.

Una solicitud de red HTTP/S es un informe que captura el tiempo desde que la aplicación envía una solicitud a un extremo del servicio hasta que se completa la respuesta de ese extremo. Captura varias métricas sobre cualquier extremo al que se envíe una solicitud:

- **Tiempo de respuesta:** tiempo transcurrido desde que se envía la solicitud hasta que se recibe la respuesta por completo.
- **Tamaño de la carga útil:** tamaño en bytes de la carga útil de red que descargó o subió la aplicación.
- **Tasa de éxito:** porcentaje de respuestas correctas en comparación con el total de respuestas (para medir errores de la red o del sistema).

Tanto para los seguimientos como para las solicitudes de red HTTP/S, podemos ver los datos de supervisión del rendimiento ordenados según las siguientes categorías:

- **Seguimientos:** versión de la aplicación, país, dispositivo, SO, radioteléfono y proveedor.
- **Solicitudes de red HTTP/S:** versión de la aplicación, país, dispositivo, SO, radioteléfono, proveedor y tipo MIME.

Performance Monitoring nunca almacena información de identificación personal de forma permanente (como nombres, direcciones de correo electrónico o números de teléfono). Para la supervisión de solicitudes de red HTTP/S, Performance Monitoring usa URL (sin incluir parámetros) para crear patrones de URL generales y anónimos, son los que se conservan y se muestran en la consola de Firebase.

5.4.2.1. Performance Monitoring en Android

No tenemos que activar nada en la consola de Firebase, simplemente instalamos del SDK de Performance Monitoring en la aplicación Android. Los datos aparecerán en la consola hasta 12 horas después de haberse capturado en el dispositivo Android.

Ejercicio: Performance Monitoring en Android

1. Abre el archivo `build.gradle` de la aplicación y agrega debajo de `apply plugin: 'com.android.application'`:

   En la sección `buildscript -> repositories`:

```
jcenter()
```

   En la sección `buildscript -> dependencies`:

```
classpath 'com.google.firebase:firebase-plugins:1.1.1'
```

   Seguramente `jcenter()` ya lo tengas incluido.

2. Abre el archivo `build.gradle` de la aplicación y agrega lo siguiente debajo de `apply plugin: 'com.android.application'`:

```
apply plugin: 'com.google.firebase.firebase-perf'
```

   Agrega en la sección `dependencies`:

```
compile 'com.google.firebase:firebase-perf:11.8.0'
```

3. Vuelve a compilar la aplicación y ejecútala.

   Ahora, se supervisarán los seguimientos automáticos y las solicitudes de red HTTP/S. Los datos empezaran a mostrarse hasta 12 horas después de haberse recopliado en la consola de Firebase.

   Accede a la consola de Firebase y pulsa sobre el menú **PERFORMANCE** del apartado STABILITY. Podrás consultar los seguimientos automáticos. Un seguimiento es un informe de datos de rendimiento que se capturan en un periodo de tiempo. Performance Monitoring proporciona automáticamente los siguientes tipos de seguimientos:

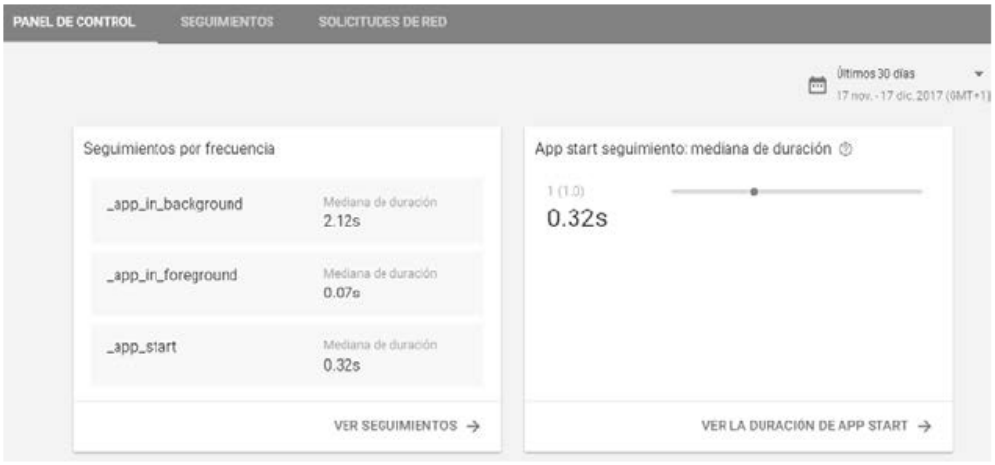
   - **Seguimientos de inicio de aplicación (_app_start):** miden el tiempo desde que el usuario abre la aplicación hasta que está lista para responder. Se inicia cuando el `ContentProvider` de `FirebasePerfProvider` completa su método `onCreate` y se detiene cuando se llama al método `onResume()` de la actividad.

   - **Seguimientos de la aplicación en segundo plano (_app_in_background):** miden el tiempo que la aplicación se ejecuta en

segundo plano. Se inicia cuando se llama al método `onStop()` de la última actividad en abandonar el primer plano y se detiene cuando se llama al método `onResume()` de la primera actividad en llegar al primer plano.

- **Seguimientos de la aplicación en primer plano (_app_in_foregorund):** miden el tiempo que la aplicación se ejecuta en primer plano y está disponible para el usuario. Se inicia cuando se llama al método `onResume()` de la primera actividad en llegar al primer plano y se detiene cuando se llama al método `onStop()` de la última actividad en abandonar el primer plano.

Puedes consultar las estadísticas de los seguimientos automáticos en la consola de Firebase.



5.4.2.2. Inhabilitar Firebase Performance Monitoring

Es recomendable que los usuarios acepten o rechacen la opción de usar Firebase Performance Monitoring, por lo que se aconseja configurar la aplicación, de manera que se pueda habilitar o deshabilitar. Esta opción también puede ser útil durante la programación y las pruebas de la aplicación.

Podemos inhabilitar Performance Monitoring cuando compilamos la aplicación con la opción de volver a habilitarlo en tiempo de ejecución, o compilar con Performance Monitoring habilitado y, luego, tener la opción de inhabilitarlo en tiempo de ejecución con Firebase Remote Config. Además, se puede desactivar Performance Monitoring por completo, sin la opción de habilitarlo en tiempo de ejecución.

Inhabilitar Performance Monitoring durante la compilación

Una situación en la que inhabilitar Performance Monitoring durante el proceso de compilación podría ser útil es para evitar recoger datos de rendimiento de una versión previa al lanzamiento de la aplicación durante la programación y las pruebas.

Para inhabilitar los seguimientos automáticos (pero no los personalizados) y la supervisión de solicitudes de red HTTP/S durante la compilación, agregamos la siguiente propiedad al archivo gradle de la aplicación dentro del apartado `android`:

```
android.applicationVariants.all {
   FirebasePerformance {
      instrumentationEnabled false
   }
}
```

Cambiar esta propiedad a `true` vuelve a habilitar los seguimientos automáticos y la supervisión de solicitudes de red HTTP/S.

Además, podemos inhabilitar Performance Monitoring durante la compilación para posteriormente habilitarlo en tiempo de ejecución. Agregaremos el siguiente elemento `<meta-data>` al manifiesto:

```
<application>
   <meta-data android:name="firebase_performance_collection_enabled"
                                                android:value="false" />
</application>
```

Para desactivar Performance Monitoring por completo sin la opción de habilitarla en tiempo de ejecución, agregamos el siguiente elemento `<meta-data>`:

```
<application>
<meta-data android:name="firebase_performance_collection_deactivated"
                                                android:value="true" />
</application>
```

Inhabilitar Performance Monitoring durante tiempo de ejecución

Podemos deshabilitar Performance Monitoring en tiempo de ejecución ejecutando el método:

```
FirebasePerformance.getInstance().setPerformanceCollectionEnabled(false);
```

La volveremos a activar llamando al mismo método, pero con valor `true`.

Remote Config nos permite hacer cambios en el comportamiento y el aspecto de la aplicación, lo que proporciona una manera ideal de inhabilitar Performance Monitoring en instancias de forma remota.

Práctica: Configurar Performance Monitoring con Remote Config

Queremos controlar Performance Monitoring de forma remota. Así activaremos o desactivaremos Performance Monitoring para todos los usuarios de nuestra aplicación en el momento que queramos. No dependeremos de que tenga una determinada versión de la aplicación para que capte datos o no. Para ello crea un

parámetro en Remote Config en, la consola de Firebase, llamado `PerformanceMonitoring`. Si tiene el valor `0`, no se capturarán datos y si tiene el valor `1`, sí que se capturarán.

Realiza los cambios necesarios en la aplicación Android. Recuerda utilizar el método:

```
FirebasePerformance.getInstance().setPerformanceCollectionEnabled(valor);
```

5.4.2.3. Seguimientos personalizados

Un seguimiento personalizado es un informe de rendimiento que asociamos a una parte del código de la aplicación. Es posible tener varios seguimientos personalizados en una aplicación y pueden estar ejecutándose al mismo tiempo. Cada seguimiento puede tener uno o más contadores asociados a eventos. Esos contadores están asociados con los seguimientos que los crean.

Vamos a realizar un seguimiento personalizado a la actividad `EventoDetalles`.

Ejercicio: Crear seguimientod personalizados

1. Importa las siguientes clases en la clase `EventoDetalles`:

```
import com.google.firebase.perf.FirebasePerformance;
import com.google.firebase.perf.metrics.Trace;
```

2. Declara la siguiente variable en la clase `EventoDetalles`:

```
Trace mTrace;
```

3. Inserta el siguiente código al final del método `onCreate` de la clase `EventoDetalles`:

```
mTrace =
      FirebasePerformance.getInstance().newTrace("trace_EventoDetalles");
mTrace.start();
```

   Utilizaremos un objeto `Trace` para realizar el seguimiento. Lo inicializamos con `FirebasePerformance.getInstance().newTrace("nombre_traza")`, donde especificaremos el nombre que deseamos dar al seguimiento (en el ejemplo `nombre_traza`). Iniciaremos el seguimiento con el método `start()`.

4. Añade el siguiente código al final de la clase `EventoDetalles`:

```
@Override
protected void onResume(){
   super.onResume();
   mTrace.start();
}
```

```
@Override
protected void onStop(){
   super.onStop();
   mTrace.stop();
}
```

Iniciamos y paramos el seguimiento con los métodos `start()` y `stop()` del objeto `Trace`.

5. Ejecuta la aplicación.

   Entra varias veces en la actividad `EventoDetalles` manteniéndola abierta en primer plano con diferentes duraciones. Observa las estadísticas capturadas en la consola de Firebase.

   Podemos asociar seguimientos a métodos específicos. Utilizamos la ación `@AddTrace` a los métodos que deseemos monitorizar y proporciona una cadena que identifique la traza. La traza comienza al inicio del método y se para cuando se completa su ejecución. Las trazas creadas con este método no tienen contadores asociados.

```
@Override
@AddTrace(name = "onCreateTrace", enabled = true)
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   ...
   ...
}
```

Por defecto, obtendremos la duración del seguimiento, es decir, el tiempo que transcurre entre los métodos `start()` y `stop()`. Pero podemos asociar métricas personalizadas a un seguimiento. Puede ser interesante contar eventos que son relevantes para evaluar el rendimiento de la aplicación. Por ejemplo, contar el número de veces que usamos el disco local, que llamamos al GPS, cuántas veces realizamos una llamada de red…

Debemos proporcionar un nombre para cada uno de estos eventos. Se trata de un contador incremental. Debemos colocar los contadores entre los métodos `start()` y `stop()` de la traza. Estos contadores se asociarán a la traza y podrán ser consultados en la consola de Firebase.

Estos son algunos ejemplos de contadores:

```
ClipData.Item item = cache.fetch("item");
if (item != null) {
    mTrace.incrementCounter("item_cache_exito");
} else {
   mTrace.incrementCounter("item_cache_error");
}

mTrace.incrementCounter("llamada_disco");
mTrace.incrementCounter("frame_perdido");
```

5.4.3. Test Lab

Probemos una aplicación en dispositivos alojados en los servidores de Google. Firebase Test Lab ofrece una infraestructura basada en la nube para probar aplicaciones Android. Con una sola operación, podemos comenzar a probar la aplicación en una amplia variedad de dispositivos y configuraciones. Los resultados de las pruebas (que incluyen registros, vídeos y capturas de pantalla) aparecen en la consola de Firebase. Incluso sin escribir código de prueba, Test Lab puede evaluarla automáticamente en busca de bloqueos.

Usaremos Test Lab para evaluar una aplicación en dispositivos reales instalados en un centro de datos de Google. Nos ayuda a encontrar problemas que solo ocurren en configuraciones de dispositivos específicas (por ejemplo, un Nexus 5, con un nivel de API de Android en particular, con una configuración regional específica).

Evaluemos una aplicación sin tener que escribir pruebas antes. Gracias a la prueba Robo, podemos identificar problemas en la aplicación sin tener que escribir pruebas. La prueba Robo analiza la estructura de la interfaz de usuario y la explora mediante una simulación automática del uso de las actividades por un usuario. Si escribimos pruebas de instrumentación, Test Lab también puede ejecutar esas pruebas.

Los dispositivos que se usan para las pruebas son dispositivos reales en los que se cargan niveles de API de Android actualizados o configuraciones regionales, según nuestras especificaciones. Esto permite que hagamos pruebas de uso en un conjunto de dispositivos y configuraciones reales.

Ejercicio: Prueba Robo en Firebase Test Lab

1. Selecciona la opción **Test Lab** en el apartado STABILITY de la consola de Firebase del proyecto “Eventos”.
2. Genera el APK de la aplicación “Eventos” y súbelo a Test Lab.

El APK a evaluar debe estar firmado. Utiliza la opción **Generate Signed APK**… en el menú **BUILD** en Android Studio para firmar.

3. Se va a generar una primera prueba Robo automáticamente. Puede tardar varios minutos en terminar, puedes cerrar la ventana y consultar los resultados más tarde.

Cuando se termine de procesar la prueba, recibirás un e-mail que te indica que ya están disponibles los resultados.

4. Vuelve a acceder a la consola de Firebase, si la has abandonado. Ves al apartado TEST LAB y selecciona la prueba que has realizado.

Test Lab te permite ejecutar pruebas de instrumentación Espresso y UI Automator 2.0 escritas para evaluar una aplicación en la consola de Firebase, Android Studio o la interfaz de línea de comandos de gcloud. También puedes usar la prueba Robo para evaluar una aplicación automáticamente en Firebase o en la línea de comando de gcloud.

La prueba Robo guarda registros, crea un "mapa de actividad" que muestra un conjunto de capturas de pantalla relacionadas con comentarios y genera un vídeo a partir de una secuencia de capturas de pantalla para mostrarte las operaciones que realizó el usuario simulado.

Si vas a ejecutar pruebas de instrumentación, escribe tu propia prueba específica para la aplicación. Cuando desarrolles pruebas de instrumentación, no olvides agregar la biblioteca de capturas de pantalla de Test Lab a tu proyecto de prueba, de esta manera podrás interpretar los resultados más fácilmente.

Elige un entorno y una matriz de pruebas. Elige un ambiente de prueba (Firebase console, Android Studio o la interfaz de línea de comando gcloud) y define una matriz de pruebas. Esto implica seleccionar un conjunto de dispositivos, niveles de API, configuraciones regionales y orientaciones de pantalla.

Ejecuta tus pruebas y revisa los resultados. Según las dimensiones de la matriz de pruebas, es posible que Test Lab tarde varios minutos en completar su ejecución. Cuando termine el proceso, podrás revisar los resultados de las pruebas en la consola de Firebase.

Prueba Robo

La prueba Robo captura los archivos de registro, guarda una serie de capturas de pantalla con anotaciones y crea un vídeo a partir de las capturas de pantalla para mostrarnos las operaciones que realizó el usuario simulado. Estos registros, capturas de pantalla y vídeos nos pueden ayudar a determinar la causa de bloqueos y a encontrar errores en la IU.

Podemos configurar la prueba Robo de varias formas:

- **Profundidad máxima.** Para configurar la profundidad máxima con la que explora la prueba Robo, indicamos a la prueba con cuánta rigurosidad debe explorar una rama en particular de la IU antes de volver a la raíz de la IU (la pantalla principal) para explorar otra rama. El valor predeterminado para la profundidad máxima es 50, y cualquier valor inferior a 2 impide que la prueba explore la aplicación más allá de la pantalla principal.

- **Tiempo de espera.** Según la complejidad de la IU, la prueba Robo podría tardar 5 minutos o más en completar un conjunto meticuloso de interacciones de IU. Recomendamos asignar al tiempo de espera de la prueba al menos 120 segundos (2 minutos) para la mayoría de las aplicaciones, y 300 segundos (5 minutos) para las aplicaciones de complejidad moderada. El valor predeterminado del tiempo de espera es de 300 segundos (5 minutos) para las pruebas que se ejecutan desde Android Studio y Google Developer Console, y de 1500 segundos (25 minutos) para las pruebas que se ejecutan desde la línea de comandos `gcloud`.

Si nuestra aplicación tiene una IU con una profundidad considerable y varias ramas de IU que pueden explorarse, es recomendable que usemos algunas de las siguientes opciones para garantizar que la prueba Robo explore la aplicación de forma completa:

- Establecemos un valor alto para el tiempo de espera, de manera que la prueba Robo pueda explorar varias ramas de la IU.

- Establecemos un valor bajo para la profundidad máxima, de manera que la prueba Robo explore hasta cierto punto cada rama de la IU.

Podemos usar la prueba Robo en Google Play Console cuando subimos y publicamos el archivo APK con el canal Alpha o el Beta. La prueba Robo se ejecuta en un conjunto común de dispositivos físicos desde diferentes ubicaciones geográficas, por lo que la prueba se realiza en varios factores de forma y configuraciones de hardware.

La prueba Robo permite acceder a cuentas de prueba y también permite introducir un texto predefinido en campos de la aplicación. Para el acceso personalizado y otras entradas de texto predefinido, la prueba Robo puede introducir texto en campos `EditText`. Para cada `string`, debemos identificar el campo `EditText` con el nombre de un recurso de Android.

La prueba Robo tiene dos métodos mutuamente exclusivos para admitir el acceso:

- **Acceso personalizado:** Si proporcionamos las credenciales de una cuenta de prueba, debemos indicarle a la prueba Robo dónde introducirlas.
- **Acceso automático:** Si la aplicación tiene una pantalla de acceso que usa una cuenta de Google para la autentificación, la prueba Robo usa una cuenta de prueba de Google, a menos que proporcionemos las credenciales de una cuenta de prueba para el acceso personalizado.

5.5. Servicio de Backup de Google

El servicio de Backup de Google nos permite realizar una copia de seguridad de los datos de nuestras aplicaciones en la nube. El fin es proporcionar un punto de restauración de datos y configuraciones de aplicaciones. Si un usuario realiza una restauración de fábrica o se cambia a un nuevo dispositivo con Android, el sistema restaura automáticamente los datos de la copia de seguridad cuando la aplicación se vuelve a instalar. De esta manera, los usuarios no tienen que reintroducir datos de configuración de la aplicación. Este proceso es completamente transparente para el usuario. No afecta a la funcionalidad o la experiencia del usuario en la aplicación. La aplicación necesita una API 8 o superior, además de una cuenta de Google. A partir de la API 23 (Android 6.0) el usuario tiene la posibilidad de utilizar un nuevo servicio de Google Play Services, Auto Backup, que realiza una copia de seguridad completa de los datos de la aplicación sin añadir código extra.

Durante una operación de copia de seguridad, el administrador de copia de Android (`BackupManager`) busca en la aplicación los datos de copia de seguridad, y luego se los pasa a un transporte de copia de seguridad, que a su vez los almacena en la nube. Durante una operación de restauración, `BackupManager` recupera los datos de la copia de seguridad y los devuelve a la aplicación. De forma que la aplicación pueda restaurar los datos en el dispositivo. Es posible realizar una solicitud para pedir una restauración, pero eso no debería ser necesario. Android realiza automáticamente la operación de restauración cuando la aplicación es instalada y existen datos de copia de seguridad asociadas con el usuario. El escenario principal en el que los datos de copia de seguridad se restauran es cuando un usuario restablece o actualiza a un nuevo dispositivo y sus aplicaciones previamente instaladas se reinstalan.

El servicio de copia de seguridad no está diseñado para la sincronización de datos de la aplicación con otros clientes o guardar datos a los que deseemos acceder durante el ciclo de vida normal de la aplicación. No podemos leer o escribir datos de copia de seguridad a demanda y no se puede acceder a ella de ninguna otra manera que a través de las API proporcionadas por el administrador de copia de seguridad.

El transporte de copia de seguridad es el componente del lado del cliente enmarcado en Android, que es personalizable por el fabricante del dispositivo y del proveedor de servicios. No se garantiza que la copia de seguridad esté disponible

en todos los dispositivos con Android. Sin embargo, nuestra aplicación no se verá afectada en caso de que un dispositivo no proporcione un transporte de copia de seguridad. Si creemos que los usuarios se beneficiarán de la copia de seguridad de datos en la aplicación, entonces podemos desarrollarla, probarla, y luego publicar la aplicación sin preocuparnos por si los dispositivos no pueden realizar copias de seguridad y causar un error en la aplicación.

Podemos estar seguro de que nuestros datos de copia de seguridad no pueden ser leídos por otras aplicaciones en el dispositivo.

5.5.1. Fundamentos

En versiones de Android menores a 6.0, para realizar las copias de seguridad de los datos de una aplicación, es necesario implementar un agente de copia de seguridad. Este agente es llamado por el administrador de copia de seguridad para que le proporcione los datos que desea copiar. También es llamado para restaurar la copia de seguridad cuando la aplicación se vuelve a instalar. El administrador de copia de seguridad se ocupa de todas sus transacciones de datos con el almacenamiento en la nube y el agente de copia de seguridad se ocupa de todas las transacciones de datos con la aplicación.

Para implementar un agente de copia de seguridad, debemos:

- Declarar el agente de copia de seguridad en el archivo de manifest.
- Registrar la aplicación con el servicio de copia de seguridad activado.
- Definir un agente de copia de seguridad mediante `BackupAgent` o `BackupAgentHelper`

La clase `BackupAgent` proporciona la interfaz con la cual la aplicación se comunica con el administrador de copia de seguridad. Si se extiende esta clase directamente, es necesario sobrescribir `onBackup()` y `onRestore()` para realizar y restaurar la copia de seguridad.

La clase `BackupAgentHelper` proporciona una envoltura conveniente para la clase `BackupAgent`, lo que minimiza la cantidad de código que se necesita escribir. En `BackupAgentHelper`, debemos utilizar uno o más objetos “ayudantes”, que automáticamente realizan y restauran la copia de seguridad de ciertos datos. Por lo tanto, no es necesario implementar `onBackup()` y `onRestore()`.

Android proporciona actualmente ayudantes que realizan copias de seguridad y restauran archivos completos de `SharedPreferences` y archivos de almacenamiento interno.

5.5.2. Auto Backup for Apps

Para aplicaciones cuya versión del target SDK sea Android 6.0 (API 23) o superior y que se ejecuten en dispositivos con Android 6.0 o superior, automáticamente se realizará una copia de seguridad de los datos en la nube. El sistema realiza automáticamente la copia de prácticamente todos los datos de la aplicación por defecto, y lo realiza sin tener que escribir código adicional.

El usuario debe activar Auto Backup for Apps. El diálogo para realizar la activación aparece en el asistente de configuración o cuando se configura la primera cuenta de Google en el dispositivo.

Cuando un usuario instala la aplicación en un nuevo dispositivo o la reinstala (por ejemplo, después de una restauración de fábrica), automáticamente el sistema restaura sus datos desde la copia en la nube. Vamos a ver cómo configurar la característica Auto Backup for Apps, explicando su comportamiento por defecto, y cómo excluir datos que no se quiera que el sistema realice copia de seguridad.

La copia se guarda encriptada en la cuenta Google Drive del usuario. No computa en la cuota de Google Drive del usuario. Cada aplicación puede almacenar hasta 25 MB. Una vez superado este límite, la aplicación deja de enviar datos a la nube. Si la aplicación solicita una restauración, obtiene los últimos datos enviados a la nube.

La copia se realiza cuando el dispositivo se encuentra en las siguientes condiciones:

- Está en reposo.
- Se está cargando.
- Está conectado a una red wifi.
- Han transcurrido al menos 24 horas desde la última copia.

Activamos la Autobackup en una aplicación añadiendo el artributo `android:allowBackup` con el valor `true`, en la etiqueta `Aplicacion` en el manifiesto.

En función de qué datos necesite la aplicación o cómo los guarde, puede ser necesario utilizar reglas específicas para incluir o excluir ciertos ficheros o directorios. Especificaremos estas reglas a través del manifiesto de la aplicación, donde indicaremos qué fichero XML contendrá el esquema de configuración de la copia.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   package="org.example.eventos">
   <uses-sdk android:minSdkVersion="23"/>
   <uses-sdk android:targetSdkVersion="23"/>
   <application ...
      android:fullBackupContent="@xml/esquema_backup">
   </application>
   ...
</manifest>
```

El atributo `Android:fullBackupContent` indica que el fichero XML, que se encuentra en el directorio res/xml, contiene las reglas para controlar qué ficheros son copiados. Por ejemplo, si queremos excluir un archivo de la copia, el fichero de configuración sería el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<full-backup-content>
   <exclude domain="file" path="device_info.txt"/>
</full-backup-content>
```

Por defecto, la copia automática excluye de la copia ficheros temporales y de caché. Los ficheros afectados son:

- Ficheros de los directorios obtenidos por los métodos `getCacheDir()` y `getCodeCacheDir()`.
- Ficheros localizados en un almacenamiento externo, a menos que residan en el directorio obtenido mediante el método `getExternalFilesDir()`.
- Ficheros del directorio obtenido por el método `getNoBackupFilesDir()`.

Podemos especificar qué ficheros incluir o excluir de la copia. La sintaxis del fichero XML de configuración es la siguiente:

```
<full-backup-content>
   <include domain=["file" | "database" | "sharedpref" | "external"
          | "root"]
   path="string" />
   <exclude domain=["file" | "database" | "sharedpref" | "external"
          | "root"]
   path="string" />
</full-backup-content>
```

Los siguientes elementos y atributos nos permiten especificar qué ficheros incluir o excluir de la copia:

- `<include>`: Especifica el conjunto de recursos a copiar. Si se especifica un elemento `<include>`, el sistema solo hará la copia de este elemento. Podemos especificar múltiples elementos usando varios `<include>`.
- `<exclude>`: Especifica qué datos queremos que el sistema excluya cuando se realice la copia. Si se incluyen elementos `<include>` y `<exclude>`, los elementos `<exclude>` tienen preferencia.
- `domain`: Indica qué tipo de recursos van a ser incluidos o excluidos de la copia. Podemos asignar los siguientes valores:
  - `root`: Si los recursos se encuentran en el directorio raíz de la aplicación.
  - `file`: Indica un recurso en el directorio retornado por el método `getFilesDir()`.
  - `database`: Especifica la base de datos que retorna `getDatabasePath()` o que interactúa con la aplicación mediante la clase `SQLiteOpenHelper`.
  - `sharedpref`: Especifica el objeto `SharedPreferences` que retorna el método `getSharedPreferences()`.

  - `external`: Indica qué recurso está en un almacenamiento externo, y corresponde a un fichero que está en el directorio que retorna el método `getExtenalFilesDir()`.
- `path`: Indica la ruta al fichero que queramos incluir o excluir en la copia.

Podemos elegir no realizar la copia automática configurando el atributo `android:allowBackup` a `false`, en el elemento `application` en el manifiesto. Veamos un ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      package="org.example.evento">
   <uses-sdk android:minSdkVersion="23"/>
   <uses-sdk android:targetSdkVersion="23"/>
   <application ...
      android:allowBackup="false">
   </application>
   ...
</manifest>
```

Hay dos escenarios en los cuales podemos necesitar soportar versiones de Android inferiores a Android 6.0 (API 23): actualizando una aplicación para aprovechar la nueva funcionalidad Auto Backup de Android 6.0, mientras continuamos soportando versiones anteriores, o podemos realizar una nueva aplicación, pero queremos asegurarnos de que funcionará en dispositivos con versiones anteriores a Android 6.0 además de tener la funcionalidad de Auto Backup.

Si la aplicación utiliza Android Backup, añadimos el atributo `android:fullBackupOnly="true"` en el elemento `<application/>` en el manifiesto. Cuando se ejecuta en un dispositivo con Android 5.1 (API 22) o inferior, la aplicación realiza la copia con Android Backup. Si el dispositivo es Android 6.0 o superior utiliza Auto Backup for Apps.

Podemos personalizar el proceso en `onCreate()` o `onFullBackup()`. Por ejemplo, para recibir una notificación cuando ocurra una restauración en `onRestoreFinished()`.

Una vez configurada la copia, deberíamos probarla para asegurarnos de que guarda y restaura correctamente. Para ayudar a analizar el fichero XML, activamos el logging antes de realizar el test:

```
$ adb shell setprop log.tag.BackupXmlParserLogging VERBOSE
```

Primero inicializamos `BackupManager` ejecutando:

```
$ adb shell bmgr run
```

Para ejecutar manualmente la copia, ejecutamos el siguiente comando. Reemplazamos `<PACKAGE>` con el nombre del paquete de la aplicación:

```
$ adb shell bmgr fullbackup <PACKAGE>
```

Para iniciar la restauración, ejecutamos. Reemplazamos `<PACKAGE>` por el nombre del paquete:

```
$ adb shell bmgr restore <PACKAGE>
```

Podemos restaurar una aplicación desinstalándola y reinstalándola. Los datos son automáticamente restaurados desde la nube cuando la instalación se ha completado.

Si la copia falla, podemos limpiar los datos de la copia y los metadatos asociados desactivando y activando Backup en Ajustes, reseteando a valores de fábrica el dispositivo o ejecutando el siguiente comando:

```
$ adb shell bmgr wipe <TRANSPORT> <PACKAGE>
```

El valor de `<TRANSPORT>` nos lo indica el paquete `com.google.android.gms`. Para obtener la lista de transportes ejecutamos:

```
$ adb shell bmgr list transports
```

Para aplicaciones Firebase Cloud Messaging (FCM), copiar el identificador puede causar comportamientos inesperados al restaurarlo. Cuando un usuario instala la aplicación en un dispositivo nuevo, el identificador FCM no es el mismo para el nuevo dispositivo. Excluimos el identificador en la copia.

Preguntas de repaso: Android Backup Service

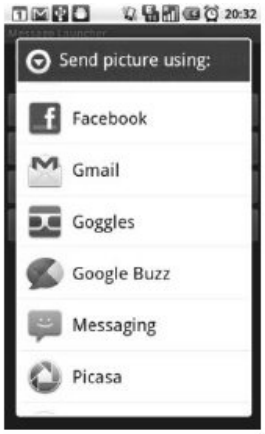
CAPÍTULO 6.

Redes sociales: Facebook y Twitter

JORDI BATALLER MASCARELL

Es difícil hoy en día no estar vinculado a una red social en la que mantener el contacto con amigos y conocidos, compartir inquietudes, aficiones, deportes, juegos y opiniones. Con sus matices, Facebook y Twitter nos ofrecen darnos a conocer y conocer a otros. Por ello, es casi obligatorio que también los programas que desarrollemos puedan usar estas redes sociales para obtener o mostrar información de quién los está utilizando.

Si entramos en el terreno del desarrollo en Android, antes de programar hay que pensar cuál es la forma más sencilla de resolver el problema que tenemos entre manos, evaluando varias alternativas. En el caso de querer publicar contenido en una red social, la primera pregunta que debemos hacernos es si lanzar una intención implícita “SEND” es suficiente para nuestros propósitos, de forma que otra aplicación distinta de la nuestra se encargue de ello:



Si, en cambio, lo que queremos es:

- integrar el acceso y uso de una red social directamente en nuestra aplicación
- aprovechar el login de la red social para validar los usuarios de nuestra aplicación y ahorrarnos el trabajo de mantener usuarios y contraseñas (con las correspondientes altas, bajas y modificaciones) y evitando que programemos nuestro proceso propio de login/logut

sigamos leyendo.

Objetivos

- Conseguir una cuenta de desarrollador en Facebook y Twitter.
- Aprender a utilizar la “consola” de gestión de aplicaciones en estas dos redes sociales.
- Dar de alta la aplicación que queremos desarrollar.
- Descargar y configurar las bibliotecas que nos servirán para interactuar con las redes sociales.
- Configurar y programar una aplicación integrada en Facebook y Twitter.

6.1. Android y Facebook

6.1.1. Preliminares

En primer lugar, vamos a ver cómo dar de alta nuestra aplicación, configurarla y descargar el kit de desarrollo propio de Facebook

Darse de alta en Facebook como desarrollador

Para escribir aplicaciones para Facebook hay que tener un usuario en la red social, pero además hay que darse de alta como desarrollador. Accedemos a:

http://developers.facebook.com/

y vamos a Entrar (o Login si está en inglés). Aquí nos pedirá hacer login con Facebook o darnos de alta si no tenemos cuenta aún. Debemos darnos de alta y entrar en la sección de desarrolladores. Por tanto:

Ejercicio: Date de alta en Facebook developers y explora el sitio

Aplicación oficial de Facebook para Android

Es necesario instalar la aplicación oficial de Facebook en el teléfono o en el emulador que vayamos a utilizar. En el caso de los teléfonos, muchos la tienen preinstalada. Si el teléfono o emulador no la tienen, se puede instalar fácilmente desde Google Play. (El emulador debe haber sido creado con una imagen que soporte, obviamente, Google Play).

SDK de Facebook para Android

Facebook proporciona un SDK (kit para desarrollo de software) para escribir aplicaciones que interactúen con su plataforma. Anteriormente debíamos descargar este SDK manualmente para utilizarlo en nuestros programas. Sin embargo, actualmente no es necesario hacerlo porque cambiando la configuración del fichero build.gradle de nuestro proyecto, este será capaz de realizar la descarga automáticamente. Si necesitáramos descargarlo manualmente, podemos hacerlo en el siguiente enlace: https://developers.facebook.com/docs/android/.

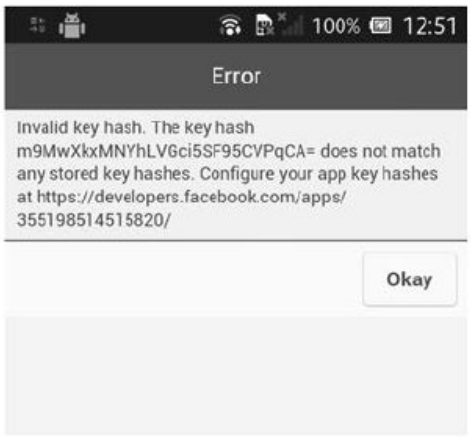
Ejemplos básicos de Facebook SDK: HelloFacebookSample

Podemos encontrar el código fuente de una serie de ejemplos en el siguiente repositorio de GitHub: https://github.com/facebook/facebook-android-sdk

Podemos configurar Android Studio para que acceda a los ejemplos, o descargarlos en un zip mediante el botón **Clone or download**.

Ejercicio: Descarga de GitHub el código de las aplicaciones de ejemplo de Facebook

Conviene probar el ejemplo “HelloFacebookSample” para ver que todo va correctamente antes de realizar nuestros propios programas. Solo hay que abrir ese ejemplo donde lo hayamos descargado y ejecutarlo. Pero antes en teoría hay que registrar en Facebook Developers la clave (keyhash) con la que AndroidStudio firma nuestras aplicaciones (ficheros .apk). Si no realizamos este paso, al utilizar, por ejemplo, la aplicación HelloFacebookSample, veríamos en algún momento el siguiente error:



O en forma de toast:

Por tanto, para poder ejecutar las aplicaciones de ejemplo consultaremos la clave pública con que se firman nuestras aplicaciones (.apk) y a la que justamente

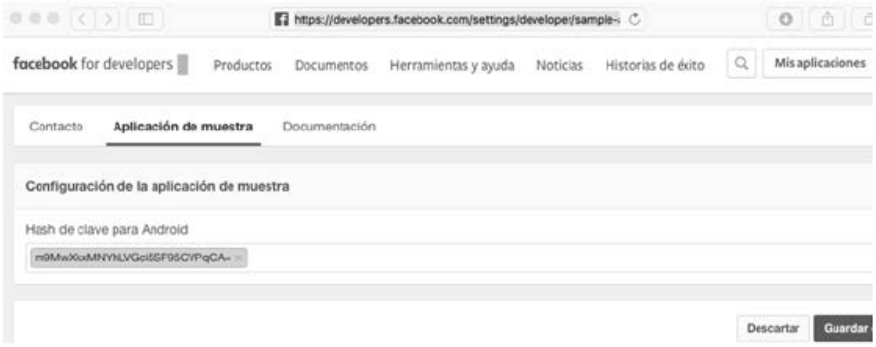
hace referencia el mensaje de error. La clave está en "~/android/debug.keystore" y se averigua utilizando los comandos keytool y openssl:

```
keytool -exportcert -alias androiddebugkey -keystore
%HOMEPATH%\.android\debug.keystore | openssl sha1 -binary | openssl base64
Escriba la contraseña del almacén de claves: android
```

(Ver más detalles en la siguiente sección). Cuando sepamos esa clave, vamos a:

https://developers.facebook.com/settings/developer/sample-app

elegimos la pestaña **Aplicación de muestra** (Sample App) y en "Hash de clave para Android (Android Key Hash)" copiamos nuestra clave pública, dando a **Guardar** (Save Changes).



Aquí hemos de realizar dos avisos:

- En alguna ocasión la aplicación HelloFacebookSample ha funcionado sin consignar la clave (normalmente con la aplicación oficial de Facebook abierta en el teléfono con nuestro usuario).
- En más de una ocasión hemos tenido problemas para guardar la clave en la anterior página web. Resulta que al pulsar el botón **Guardar**, la clave desaparece. Para solucionar este problema se puede:
    - Probar navegadores y/o sistemas operativos diferentes.
    - Dar de alta una aplicación en Facebook Developers, como se describe en el siguiente apartado ("Configurar nuestra nueva aplicación").

Por último, recordar que necesitaremos recompilar las aplicaciones de ejemplo de Facebook para que integren la clave a partir de debug.keystore si la hemos cambiado; y que tendremos que consignar las claves de todos los debug.keystore si trabajamos con ordenadores diferentes.

Por fin, llega el momento de probar el ejemplo "HelloFacebookSample":

Con esta aplicación podremos hacer login en Facebook y enviar mensajes y fotos a nuestro muro. Cuando la aplicación pida permisos aparecerá una pantalla para que nos identifiquemos, solicitando el nombre de usuario y contraseña de Facebook. Esto no es necesario si tenemos la aplicación oficial de Facebook abierta con nuestro usuario o ya hemos dado los permisos con anterioridad.

Ejercicio: Prueba la aplicación HelloFacebookSample

Configurar nuestra nueva aplicación (en Facebook Developers)

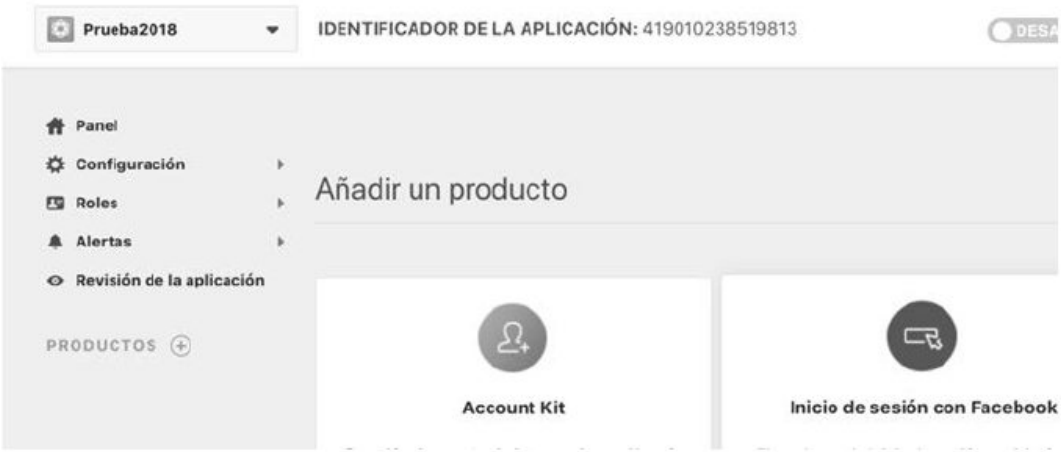
Después de comprobar que la aplicación de ejemplo funciona, vamos a tratar de escribir una aplicación nuestra que haga algo similar, tomando código de los ejemplos.

El primer paso, antes de utilizar Android-Studio, es dar de alta en nuestro gestor de Apps de Facebook Developers los datos de la aplicación que queremos desarrollar, aunque esta no esté aún escrita.

Para ello, vamos a developers.facebook.com/apps, donde pulsaremos en el Botón **Añadir una nueva aplicación** (o también en “My Apps / Add a New App” o Mis Aplicaciones / Añadir una nueva aplicación).

Se nos preguntarán datos básicos como el nombre de la aplicación, un correo electrónico, el tipo de aplicación, etc.

Terminado este paso, estaremos en el PANEL DE CONTROL de nuestra nueva aplicación.

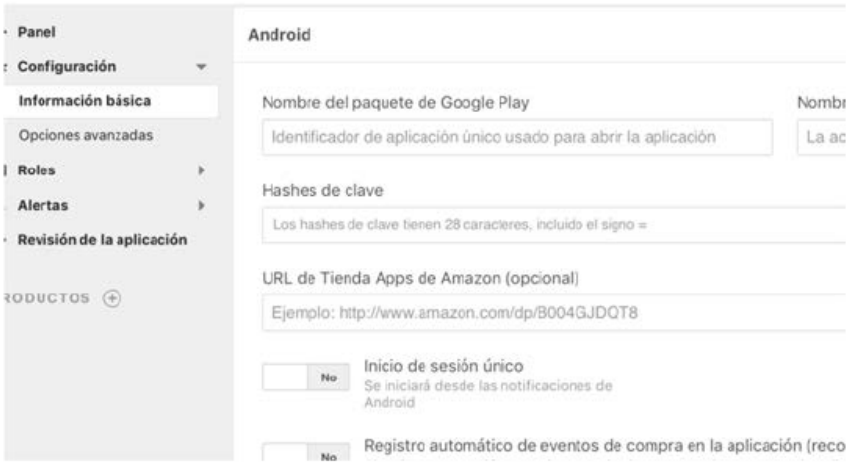


Si pulsamos en **Información básica**, en la pestaña **Configuración** de la izquierda, veremos:

Fijémonos en el Identificador de la aplicación (App ID) que nos ha asignado Facebook y en la Clave secreta, porque luego los vamos a necesitar. Aquí podemos poner o cambiar el correo electrónico de contacto, así como la imagen o icono de nuestra aplicación.

Ahora, pulsamos el botón **Añadir plataforma** y elegimos Android en la siguiente pantalla:

Tras esto, aparece una nueva zona "Android" en la pantalla:



En esta zona de datos en la pantalla es donde deberemos escribir el nombre de la clase principal de nuestra aplicación y especialmente la clave (Hashes de clave) con que estará firmada nuestra .apk.

Para poder consignar en la anterior pantalla la parte pública de la clave con que se firman nuestros ficheros .apk, deberemos averiguarla. Las claves están guardadas en %HOMEPATH%/.android/debug.keystore y se obtiene con los comandos keytool y openssl:

```
keytool -exportcert -alias androiddebugkey
 -keystore %HOMEPATH%\.android\debug.keystore
 | openssl sha1 -binary | openssl base64

Escriba la contraseña del almacén de claves: android
```

Véase que la contraseña que hemos de dar es **android**. El anterior comando (todo en una misma línea) nos devolverá una clave pública como la siguiente (acabada en con el símbolo =):

```
IpgCWdGg47PEaDQy/aO+54rt/TU=
```

Nota: En Unix, el comando es el mismo cambiando %HOMEPATH% por $HOME y las barras \ por /. La utilidad keytool está instalada con el JDK de Java. La utilidad openssl puede descargarse gratuitamente desde Internet (buscar en Google). Ambas hay que añadirlas a la variable de entorno PATH.

El keyhash se puede averiguar también a partir de un .apk que tengamos. En Mac-OSX el comando es

```
keytool -list -printcert -jarfile miaplicacion.apk | grep "SHA1: "
  | cut -d " " -f 3 | xxd -r -p | openssl base64
```

En linux/unix/MacOS es:

```
keytool -list -printcert -jarfile [path_to_your_apk] |
grep -Po "(?<=SHA1:) .*" |  xxd -r -p | openssl base64
```

En windows se pueden utilizar estos comandos, habiendo instalado antes las herramientas grep, cut y xxd. Finalmente, también es posible extraer el keyhash por programa mediante el siguiente código:

```
PackageInfo info;
try {
   info = getPackageManager().getPackageInfo("com.you.name", // nombre
paquete
          PackageManager.GET_SIGNATURES);
    for (Signature signature : info.signatures) {
      MessageDigest md;
      md = MessageDigest.getInstance("SHA");
      md.update(signature.toByteArray());
      String something = new String(Base64.encode(md.digest(), 0));
// Log.d("KeyHash:", Base64.encodeToString(md.digest(), Base64.DEFAULT));

        Log.e("hash key", something);
    }
} catch (NameNotFoundException e1) {
    Log.e("name not found", e1.toString());
} catch (NoSuchAlgorithmException e) {
    Log.e("no such an algorithm", e.toString());
} catch (Exception e) {
    Log.e("exception", e.toString());
}
```

Este paso de averiguar la clave hash y darla de alta en Facebook Developers suele dar problemas y por ello, más tarde, la aplicación no puede publicar en Facebook o realizar cualquier otra actividad relacionada. Recomendamos fijarse en que no aparece ningún mensaje de aviso (“warning”) o error en la consola al extraer la clave. También es conveniente que extraigamos más de una vez la clave y comprobemos que es la misma siempre. Si uno de los comandos falla, al final lo que obtendremos será un mensaje de aviso o error codificado como clave hash, en lugar de la auténtica clave guardada.

Cuando ya estemos seguros de tener la clave correcta, en Facebook Developers, vamos a la sección de Android de la configuración y copiamos nuestra clave en Hashes de clave (Key hashes) y guardamos los cambios.

Los campos Nombre del paquete de Google Play (Google Play Package Name) y Nombre de la clase (Class Name) los rellenaremos cuando generemos el

proyecto de Android Studio. Es importante recordar que la clave la tenemos guardada localmente en nuestro ordenador (en debug.keystore), por lo que si cambiamos de ordenador o generamos una .apk para distribuir ("release": que utiliza las claves de release.keystore), deberemos repetir este paso, averiguando la clave correspondiente y añadiéndola como acabamos de describir.

Ejercicio: Aprende a extraer el keyhash de tu debug.keystore de alguna de las formas antes explicadas

Ejercicio: Da de alta una nueva aplicación tuya en Facebook Developers siguiendo las anteriores instrucciones

Algunas consideraciones "Sandbox mode", "aproved items"

Si se quiere que la aplicación sea utilizada por cualquier usuario además del desarrollador, hay que desactivar "En desarrollo" (sandbox mode) haciendo que sea una aplicación pública. Esto puede hacerse en Revisión de la aplicación / Status & Review marcando "**Sí**" en ¿Quieres que …. sea una aplicación pública?

Las aplicaciones, por defecto, tienen derecho a obtener de Facebook los datos básicos del usuario acreditado: "e-mail, public_profile, y user_friends. El usuario es informado y nos concede estos permisos cuando hace login en Facebook desde nuestra aplicación.

Desde 2015, si nuestra aplicación desea realizar acciones u obtener información diferente de las anteriores (p.ej. pedir permiso para publicar: publish_actions), deberemos solicitar a Facebook que revise nuestra aplicación y la valide.

Es decir: **deberemos informar y solicitar a Facebook autorización para poder pedir permiso al usuario sobre determinadas capacidades.** En este mismo apartado, podremos pulsar **Iniciar una solicitud** (Start a submission) para conseguirlo. No es un proceso inmediato: hay que proporcionar información a Facebook y esperar entre 4 y 7 días laborables a que respondan.

En resumen, por defecto (sin que Facebook valide nuestra aplicación), solo podremos publicar desde nuestra aplicación si el usuario Facebook que la utiliza es el mismo usuario que como desarrollador ha dado de alta la aplicación. Insistimos: para que un usuario distinto del desarrollador pueda publicar mensajes, el desarrollador tendrá que haber desactivado el sand box mode y que su aplicación haya sido revisada y aprobada para el público general. Mientras estemos en esta situación, podremos ver avisos como el siguiente: "Some of the permissions below have not been approved for use by Facebook".

Entendamos que toda esta protección es para evitar comportamientos incorrectos de una aplicación no validada (p. ej. que publique en nombre de un usuario, mensajes que este no ha escrito).

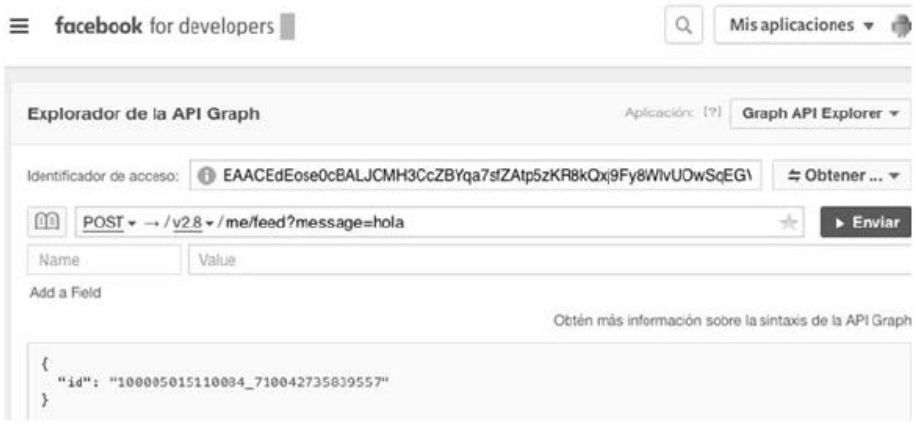
API Graph

La biblioteca del SDK de Facebook tiene un grupo de funciones denominadas API Graph que permiten interactuar directamente con Facebook. Se denomina así porque usa la metáfora de que una red social es un grafo: nodos que son personas o recursos y aristas que conectan las personas y los recursos entre ellas (se puede encontrar información sobre esta API en https://developers.facebook.com/docs/graph-api/overview/). Esta API es de tipo REST: utiliza verbos de HTTP como GET, POST, DELETE para ejecutar acciones sobre los recursos e información almacenada en Facebook. Por ejemplo, para obtener información sobre el usuario actual (“me”, en inglés) la petición sería:

```
GET /v2.8/me HTTP/1.1
```

Sin embargo, trabajar a este nivel de detalle es extremadamente complicado porque la petición HTTP hay que completarla con muchos más campos que inclu-

yen información (tokens) de validación. El SDK de Facebook para Android envuelve dichas peticiones dentro de métodos mucho más sencillos de usar. En cualquier caso, hay una herramienta de exploración de esta API en este enlace: https://developers.facebook.com/tools/explorer/. En la siguiente pantalla podemos ver qué orden REST es la que permite enviar un nuevo mensaje ("status") a nuestro muro.



Al final de la anterior página, hay un botón para obtener el código que utilizaríamos para dicha acción según la plataforma de desarrollo. Por ejemplo, para Android veríamos:

Share Dialog

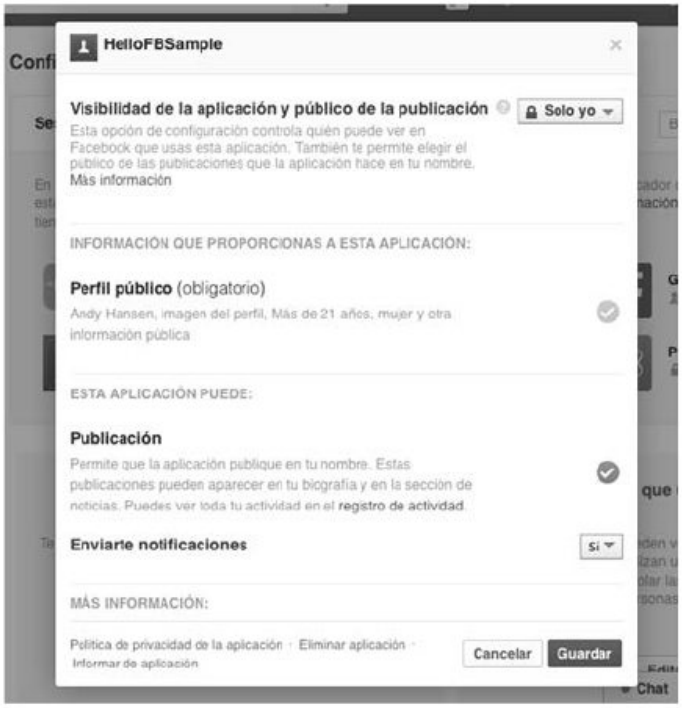
A parte del Graph API que permite interactuar directamente con Facebook, la biblioteca de Facebook tiene otra funcionalidad para interactuar indirectamente con Facebook llamada Share Dialog. Esta forma no requiere login ni validación de nuestra aplicación porque en realidad utilizan la app oficial de Facebook para Android (usa algunas activities suyas). En esta alternativa, el usuario ya ha hecho login en la propia app de Facebook (no en nuestra aplicación), y cada vez que nuestra aplicación requiera algo, aparecerá la ventana correspondiente de la app de Facebook.

Gestión de las aplicaciones “instaladas” en Facebook

Un usuario corriente de Facebook (no un desarrollador) puede ver qué aplicaciones tienen permiso para interactuar en su nombre ante Facebook. Para ver cuáles son, vamos a **Configuración**

y luego pinchamos en la pestaña **Aplicaciones**, donde veremos la lista de aplicaciones autorizadas en nuestro nombre. Por ejemplo:

Desde aquí podemos ver qué permisos utiliza cada aplicación relacionada con Facebook que estamos utilizando (en cualquier dispositivo) y revocar o conceder permisos. Por ejemplo:

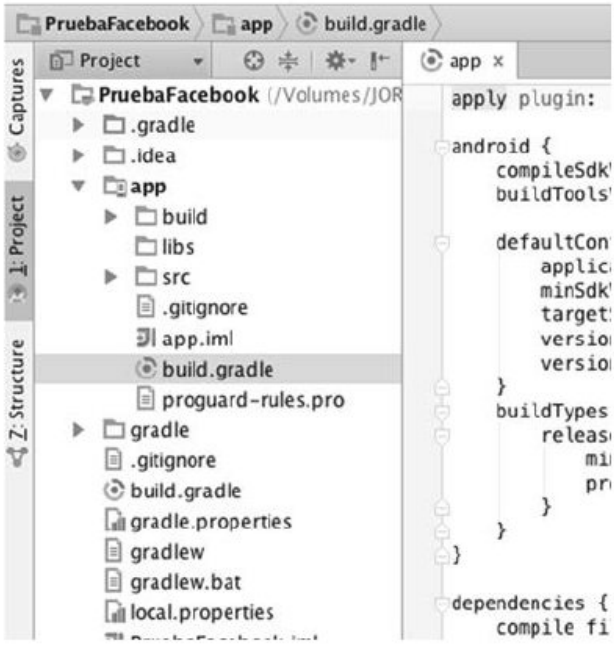


6.1.2. Nuestro proyecto Android

Tras haber realizado la descripción de nuestro programa ante Facebook, a falta de un pequeño detalle, es el momento de crear un proyecto para desarrollar en Android nuestra aplicación.

Importar la biblioteca Facebook-Android-SDK en Android Studio

Para utilizar el SDK de Android en un proyecto **nuevo,** tendremos que abrir el fichero build.gradle (Module:app):



y añadir dentro de la sección android { }:

```
repositories  {
   mavenCentral()
}
```

y dentro de la sección dependencies { }:

```
compile 'com.facebook.android:facebook-android-sdk:[4,5)'
```

Esta modificación hará que automáticamente se conecte al repositorio “maven central” de Internet para descargar la biblioteca de Facebook.

Configurar nuestra aplicación (en Facebook)

Antes ya habíamos declarado en Facebook cuál es la clave con que firmamos nuestra aplicación. Ahora que ya hemos empezado nuestra aplicación con Android Studio, podemos copiar el nombre de la clase Java donde la estamos escribiendo. Si por ejemplo esta se llama:

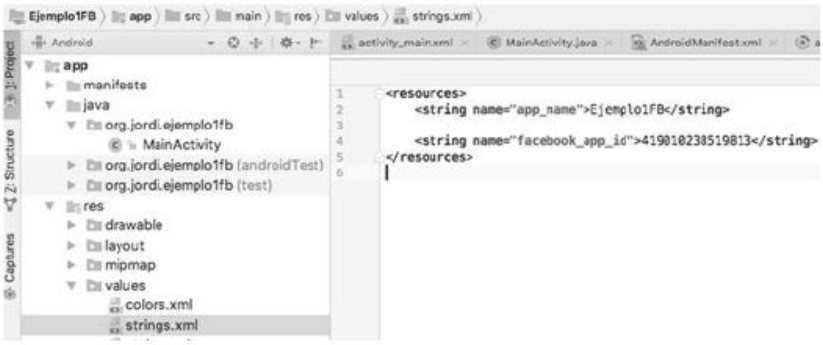
```
org.jordi.ejemplo1fb
```

entonces, trasladaremos ese nombre a la sección CONFIGURACIÓN (Setting) de nuestra aplicación:

Configurar el código de nuestra aplicación

El siguiente paso, consiste en guardar el número identificador con que nuestra aplicación está registrada en Facebook. Podemos ver dónde encontrarlo en la anterior pantalla.

Así pues, en res/values/strings.xml de nuestro proyecto añadimos la propiedad “facebook_app_id” con el valor correspondiente:



En el fichero AndroidManifest.xml hay que realizar cambios, relacionados con el acceso a Internet que necesita nuestra aplicación, con la identificación de nuestra aplicación Facebook, y con el proceso de login. Reproducimos a continuación el contenido del AndroidManifest.xml e indicaremos en los comentarios los 3 cambios necesarios:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jordi.ejemplo1fb">

<!-- 1. añadir estos dos permisos -->
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
          android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

<!-- 2. añadir por exigencia de Facebook -->
        <meta-data android:name="com.facebook.sdk.ApplicationId"
                            android:value="@string/facebook_app_id" />
        <!-- nuestra actividad -->
        <activity android:name=".MainActivity">
           <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name="android.intent.category.LAUNCHER"/>
           </intent-filter>
        </activity>

<!-- 3. añadir actividad para hacer login en Facebook (por exigencia)-->
        <activity android:name="com.facebook.FacebookActivity" />
    </application>
</manifest>
```

6.1.3. Aplicación de ejemplo (usando API Graph)

La aplicación que vamos a estudiar como ejemplo simplemente sirve para hacer login en Facebook y publicar mensajes o imágenes en él (utilizando lal Graph API), en nombre de un usuario acreditado (que solo podrá ser, si no pedimos a Facebook que valide nuestra aplicación, el programador):

Para hacer login, podemos utilizar el botón oficial de Facebook y, como ejemplo adicional, unos botones ordinarios. El proceso de login lleva a la aplicación oficial de Facebook de nuestro teléfono informando al usuario sobre los permisos requeridos. Mientras desde allí (aplicación oficial) no hagamos logout, no se pide nuevamente ninguna contraseña en nuestra aplicación para hacer login. Si ya estábamos identificados con la aplicación oficial, hace el login directamente.

Detalles del código:

- En primer lugar, para incluir el botón de Facebook hay que incluir en layout de la actividad lo siguiente.

```
<com.facebook.login.widget.LoginButton
    android:id="@+id/login_button"
    android:layout_width="wrap_content"
    android:layout_height="121dp"
    facebook:com_facebook_confirm_logout="false"
    facebook:com_facebook_tooltip_mode="never_display"
    />
```

- Hay que configurar el botón de Facebook según los permisos que necesitemos:

```
loginButtonOficial.setPublishPermissions("publish_actions");
```

- Conviene registrar un callback para saber si el login tiene éxito:

```
LoginManag-
er.getInstance().registerCallback(this.elCallbackManagerDeFacebook,
        new FacebookCallback<LoginResult>() {
            @Override
            public void onSuccess(LoginResult loginResult) { …
```

- Debemos sobrescribir onActivityResult() y allí informar a Facebook:

```
this.elCallbackManagerDeFacebook.onActivityResult(requestCode,    re-
sultCode, data);
```

- Una forma sencilla de obtener el profile (datos básicos) del usuario acreditado es esta:

```
Profile profile = Profile.getCurrentProfile();
```

- Cuando un usuario está acreditado, nuestra aplicación dispone de un access token que nos representa ante Facebook, y que es necesario para interactuar con él. Se puede obtener así:

```
AccessToken acessToken = AccessToken.getCurrentAccessToken();
```

- Si no utilizamos el botón de login de Facebook, podemos también hacer login o logout de esta forma:

```
LoginManager.getInstance().logInWithPublishPermissions(this,
       Arrays.asList("publish_actions"));
LoginManager.getInstance().logout();
```

- Enviar mensaje al muro de Facebook, mediante API Graph:

```
Bundle params = new Bundle();
params.putString("message", textoQueEnviar);

GraphRequest request = new GraphRequest(
        AccessToken.getCurrentAccessToken(),
        "/me/feed",
        params,
        HttpMethod.POST,
        new GraphRequest.Callback() {
            public void onCompleted(GraphResponse response) {
                Toast.makeText(THIS, "Publicación realizada: " +
                      textoQueEnviar, Toast.LENGTH_LONG).show();
            }
        }
);
request.executeAsync();
```

- Enviar imagen al muro de Facebook, mediante API Graph:

```
Bundle params = new Bundle();
params.putByteArray("source", byteArray); // bytes de la imagen
params.putString("caption", comentario); // comentario
// si se quisiera publicar una imagen de internet: params.putString("url",
// "{image-url}");
GraphRequest request = new GraphRequest(
        AccessToken.getCurrentAccessToken(), "/me/photos", params,
        HttpMethod.POST,
        new GraphRequest.Callback() {
            public void onCompleted(GraphResponse response) {
                Toast.makeText(THIS, "" + byteArray.length +
                           " Foto enviada: " + response.toString(),
                                       Toast.LENGTH_LONG).show();
            }
        }
);
request.executeAsync();
```

Hay que hacer notar que la Graph API de Facebook es una interfaz tipo REST que por debajo envía peticiones como:

```
POST graph.facebook.com/{ user-id}/feed?message =
                {message}&access_token={access-token}
```

Por completitud incluimos a continuación el código entero de la actividad:

```
package org.jordi.ejemplo1fb;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import com.facebook.AccessToken;
import com.facebook.CallbackManager;
import com.facebook.FacebookCallback;
import com.facebook.FacebookException;
import com.facebook.FacebookSdk;
import com.facebook.GraphRequest;
import com.facebook.GraphResponse;
import com.facebook.HttpMethod;
import com.facebook.Profile;
import com.facebook.login.LoginManager;
import com.facebook.login.LoginResult;
import com.facebook.login.widget.LoginButton;
import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.content.Context;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.util.Log;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import org.json.JSONObject;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.Arrays;
import com.facebook.FacebookSdk;

public class MainActivity extends AppCompatActivity {
    private TextView elTextoDeBienvenida;
    private Button botonHacerLogin;
    private Button botonLogOut;
    private Button botonEnviarFoto;
    private TextView textoConElMensaje;
    private Button botonCompartir;
    // boton oficial de Facebook para login/logout
    LoginButton loginButtonOficial;

    // gestiona los callbacks al FacebookSdk desde el método
    // onActivityResult() de una actividad
    private CallbackManager elCallbackManagerDeFacebook;
```

```
    // puntero a this para los callback
    private final Activity THIS = this;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("cuandrav.onCreate()", " .onCreate() llamado");

        // cosas de Facebook
        // inicializar FacebookSDK
        // 2018: no hace falta, se puede borrar:
http://stackoverflow.com/questions/41904350/facebooksdk-sdkinitializegetapplicationcontext-
deprecated
       // FacebookSdk.sdkInitialize(this.getApplicationContext());

        // pongo el contenido visual de la actividad (hacer antes que
findViewById ()
        // y después de inicializar FacebookSDK)
        //
        this.setContentView(R.layout.activity_main);

        // botón oficial de "login en Facebook"

        // obtengo referencia
        loginButtonOficial = (LoginButton)
findViewById(R.id.login_button);

        // declaro los permisos que debe pedir al ser pulsado
        // ver lista en: https://developers.facebook.com/docs/facebook-login/permissions
        loginButtonOficial.setPublishPermissions("publish_actions");

        //loginButtonOficial.setReadPermissions("public_profile"); // si
pones uno, no puedes poner el otro

        // crear callback manager de Facebook
        this.elCallbackManagerDeFacebook = CallbackManag-
er.Factory.create();

        // registro un callback para saber cómo ha ido el login     Login-
Manager.getInstance().registerCallback(this.elCallbackManagerDeFacebook,
                new FacebookCallback<LoginResult>() {
                    @Override
                    public void onSuccess(LoginResult loginResult) {
                        // App code
                        Toast.makeText(THIS, "Login onSuccess()",
Toast.LENGTH_LONG).show();
                        actualizarVentanita();
                    }
                    @Override
                    public void onCancel() {
                        Toast.makeText(THIS, "Login onCancel()",
Toast.LENGTH_LONG).show();
                        actualizarVentanita();
                    }
```

```
                    @Override
                    public void onError(FacebookException exception) {
                        // App code
                        Toast.makeText(THIS, "Login onError(): " + excep-
tion.getMessage(),
                                Toast.LENGTH_LONG).show();
                        actualizarVentanita();
                    }
                });
        // otras cosas
        // obtengo referencias a mis otros widgets en el layout
        elTextoDeBienvenida = (TextView) findViewById(
                                            R.id.elTextoDeBienvenida);
        botonHacerLogin = (Button) findViewById(R.id.boton_hacerLogin);
        botonLogOut = (Button) findViewById(R.id.boton_LogOut);
        botonEnviarFoto = (Button) findViewById(R.id.boton_EnviarFoto);
        textoConElMensaje = (TextView) findViewById(R.id.txt_mensajeFB);
        botonCompartir = (Button) findViewById(R.id.boton_EnviarAFB);
        this.actualizarVentanita();
        Log.d("cuandrav.onCreate", "final .onCreate() ");
    }

    @Override
    protected void onActivityResult(final int requestCode, final int re-
sultCode, final Intent data) {
        // se llama cuando otra actividad que hemos arrancado termina y
nos devuelve el control
        // tal vez, devolviéndonos algun resultado (resultCode, data)
        Log.d("cuandrav.onActivityResu", "llamado");
        super.onActivityResult(requestCode, resultCode, data);
        // avisar a Facebook (a su callback manager) por si le afecta
        this.elCallbackManagerDeFacebook.onActivityResult(requestCode,
             resultCode, data);
    }

    private void actualizarVentanita() {
        Log.d("cuandrav.actualizarVent", "empiezo");
        // obtengo el access token para ver si hay sesión
        AccessToken accessToken = this.obtenerAccessToken();
        if (accessToken == null) {
            Log.d("cuandrav.actualizarVent","no hay sesion, deshabilito");
            // sesion con facebook cerrada
            this.botonHacerLogin.setEnabled(true);
            this.botonLogOut.setEnabled(false);
            this.textoConElMensaje.setEnabled(false);
            this.botonCompartir.setEnabled(false);
            this.botonEnviarFoto.setEnabled(false);
            this.elTextoDeBienvenida.setText("haz login");
            return;
        }
        // sí hay sesión
        Log.d("cuandrav.actualizarVent", "hay sesion habilito");
        this.botonHacerLogin.setEnabled(false);
```

```
        this.botonLogOut.setEnabled(true);
        this.textoConElMensaje.setEnabled(true);
        this.botonCompartir.setEnabled(true);
        this.botonEnviarFoto.setEnabled(true);
        // averiguo los datos básicos del usuario acreditado
        Profile profile = Profile.getCurrentProfile();
        if (profile != null) {
            this.textoConElMensaje.setText(profile.getName());
        }
        // otra forma de averiguar los datos básicos:
        // hago una petición con "graph api" para obtener datos del
        // usuario acreditado
        this.obtenerPublicProfileConRequest_async(
                // como es asíncrono he de dar un callback
                new GraphRequest.GraphJSONObjectCallback() {
                    @Override
                    public void onCompleted(JSONObject datosJSON, GraphRe-
sponse response) {
                        // muestro los datos
                        String nombre= "nombre desconocido";
                        try {
                            nombre = datosJSON.getString("name");
                        } catch (org.json.JSONException ex) {
                            Log.d("cuandrav.actualizarVent", "callback de
obtenerPublicProfileConRequest_async: excepcion: "
                                    + ex.getMessage());

                        } catch (NullPointerException ex) {
                            Log.d("cuandrav.actualizarVent", "callback de
obtenerPublicProfileConRequest_async: excepcion: "
                                    + ex.getMessage());

                        }
                        elTextoDeBienvenida.setText("bienvenido 2018: " +
nombre);
                    }
                });
    }

    private boolean sePuedePublicar() {
        // compruebo la red
        if (!this.hayRed()) {
            Toast.makeText(this, "¿no hay red? No puedo publicar",
Toast.LENGTH_LONG).show();
            return false;
        }

        // compruebo permisos
        if (! this.tengoPermisoParaPublicar()) {
            Toast.makeText(this, "¿no tengo permisos para publicar? Los
pido.", Toast.LENGTH_LONG).show();
            // pedirPermisoParaPublicar();
            LoginManager.getInstance().logInWithPublishPermissions(this,
Arrays.asList("publish_actions"));
```

```
            return false;
        }

        return true;
    }

    private AccessToken obtenerAccessToken() {
        return AccessToken.getCurrentAccessToken();
    }

    private boolean tengoPermisoParaPublicar() {
        AccessToken accessToken = this.obtenerAccessToken();
        return accessToken != null && ac-
cessToken.getPermissions().contains("publish_actions");
    }

    private boolean hayRed() {
```

```
// comprobar que estamos conetactos a internet,
```

```
// antes de hacer el login
// con facebook. Si no: da problemas.
```

```
        ConnectivityManager connectivityManager = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo activeNetworkInfo = connectivityManager
                .getActiveNetworkInfo();
        return activeNetworkInfo != null && activeNetworkIn-
fo.isConnected();
    }

    public void enviarTextoAFacebook_async (final String textoQueEnviar) {
        // si no se puede publicar no hago nada
        if ( ! sePuedePublicar() ) {
            return;
        }

        // hago la petición a través del API Graph
        Bundle params = new Bundle();
        params.putString("message", textoQueEnviar);

        GraphRequest request = new GraphRequest(
                AccessToken.getCurrentAccessToken(),
                "/me/feed",
                params,
                HttpMethod.POST,
                new GraphRequest.Callback() {
```

```
                    public void onCompleted(GraphResponse response) {
                        Toast.makeText(THIS, "Publicación realizada: " +
textoQueEnviar, Toast.LENGTH_LONG).show();

                    }
                }
        );
        request.executeAsync();
    }

    public void enviarFotoAFacebook_async (Bitmap image, String comen-
tario) {
        Log.d("cuandrav.envFotoFBasync", "llamado");
        if (image == null){
            Toast.makeText(this, "Enviar foto: la imagen está vacía.",
Toast.LENGTH_LONG).show();
            Log.d("cuandrav.envFotoFBasync", "acabo porque la imagen es
null");
            return;
        }
        // si no se puede publicar no hago nada
        if ( ! sePuedePublicar() ) {
            return;
        }
        // convierto el bitmap a array de bytes
        ByteArrayOutputStream stream = new ByteArrayOutputStream();
        image.compress(Bitmap.CompressFormat.PNG, 100, stream);
        //image.recycle ();
        final byte[] byteArray = stream.toByteArray();
        try {
            stream.close();
        } catch (IOException e) { }
        // hago la petición a traves del Graph API
        Bundle params = new Bundle();
        params.putByteArray("source", byteArray); // bytes de la imagen
        params.putString("caption", comentario); // comentario
        // si se quisiera publicar una imagen de internet:
        // params.putString("url", "{image-url}");
        GraphRequest request = new GraphRequest(
                AccessToken.getCurrentAccessToken(),
                "/me/photos",
                params,
                HttpMethod.POST,
                new GraphRequest.Callback() {
                    public void onCompleted(GraphResponse response) {
                        Toast.makeText(THIS, "" + byteArray.length + "
Foto enviada: " + response.toString(), Toast.LENGTH_LONG).show();
                        //textoConElMensaje.setText(response.toString());
                    }
                }
        );
        request.executeAsync();
    }
```

```
    public void boton_Login_pulsado(View quien) {
        // compruebo la red
        if (!this.hayRed()) {
            Toast.makeText(this, "¿No hay red? No puedo abrir sesión",
                    Toast.LENGTH_LONG).show();
        }
        // login
        LoginManager.getInstance().logInWithPublishPermissions(this,
                  Arrays.asList("publish_actions"));
        // actualizar
        this.actualizarVentanita();
    }

    public void boton_Logout_pulsado(View quien) {
        // compruebo la red
        if (!this.hayRed()) {
            Toast.makeText(this, "¿No hay red? No puedo cerrar sesión",
                    Toast.LENGTH_LONG).show();
        }
        // logout
        LoginManager.getInstance().logOut();
        // actualizar
        this.actualizarVentanita();
    }

    private void obtenerPublicProfileConRequest_async (
                        GraphRequest.GraphJSONObjectCallback callback) {
```

```
if (!this.hayRed()) {           Toast.makeText(this,"¿No hay red?",
```

```
Toast.LENGTH_LONG).show();
        }
        // obtengo access token y compruebo que hay sesión
        AccessToken accessToken = obtenerAccessToken();
        if (accessToken == null) {
            Toast.makeText(THIS, "no hay sesión con Facebook",
                    Toast.LENGTH_LONG).show();
            return;
        }
        // monto la petición: /me
        GraphRequest request = GraphRequest.newMeRequest(accessToken,
               callback);
        Bundle params = new Bundle ();
        params.putString("fields", "id, name");
        request.setParameters(params);
        // la ejecuto (asíncronamente)
        request.executeAsync();
    }
```

```
    public void boton_enviarTextoAFB_pulsado(View quien) {
        // cojo el mensaje que ha escrito el usuario
        String mensaje = "msg:" + this.textoConElMensaje.getText() + " :"
                + System.currentTimeMillis();
        // borro lo escrito
        this.textoConElMensaje.setText("");
        // cierro el soft-teclado
        InputMethodManager imm = (InputMethodManager) getSystemService(
               Context.INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow(this.textoConElMensaje
               .getWindowToken(), 0);
        // llamo al método que publica
        enviarTextoAFacebook_async(mensaje);
    }
}
```

6.1.4. Aplicación de ejemplo (Share Dialog)

Veamos ahora un pequeño ejemplo sobre cómo utilizar la funcionalidad de Share Dialog para publicar en Facebook indirectamente (a través de la app oficial de Facebook) y sin hacer login desde nuestra propia aplicación.

Configuraremos nuestra aplicación como hemos explicado antes. Damos de alta nuestra aplicación en Facebook Developers y copiamos allí el keyhash y el nombre de la clase de nuestra actividad. Luego, en strings.xml definiremos el valor "facebook_app_id" con el número de aplicación que Facebook nos ha dado.

```
<string name="facebook_app_id">874076352708325</string>
```

En AndroidManifest.xml realizaremos cuatro modificaciones de forma que este quede como a continuación. Especialmente, remarcamos que para enviar fotos con Share Dialog, hay que añadir un aviso de "content provider" con un nombre que incluye el ID de nuestra aplicación:

```
com.facebook.app.FacebookContentProvider874076352708325
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="org.cuandrav.pruebafacebooksdk2">
<!-- 1. añadir estos dos permisos -->
   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission android:name=
                            "android.permission.ACCESS_NETWORK_STATE"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
<!-- 2. añadir por Facebook -->
```

```
        <meta-data android:name="com.facebook.sdk.ApplicationId"
                   android:value="@string/facebook_app_id" />
        <activity android:name=".MainActivity"
                  android:label="@string/app_name">
            <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
<!-- 3. añadir por Facebook -->
        <activity android:name="com.facebook.FacebookActivity" />

<!-- 4. añadir por Facebook -->
        <!--  para poder enviar fotos con Share Dialog. Fijaos dónde hay
que poner el Facebook App ID -->
        <!-- ver: https://developers.facebook.com/docs/android/getting-
started, sección Enviar Mensajes o Videos Sending Images or Videos -->
        <provider android:authorities=
                 "com.facebook.app.FacebookContentProvider874076352708325"
            android:name="com.facebook.FacebookContentProvider"
            android:exported="true" />
    </application>
</manifest>
```

Sobre el código de la actividad podemos comentar los siguientes aspectos:

- Crearemos un objeto ShareDialog en “onCreate()”.

```
this.elShareDialog = new ShareDialog(this);
```

- Disponemos del método canShow() para interrogar al Share Dialog si es capaz de realizar una determinada acción (publicar, publicar foto, publicar enlace, publicar vídeo…) Si tenemos instalada la app oficial de Facebook para Android, la respuesta será “verdadero”. En caso de que fuera falso, algunas las acciones se pueden realizar igualmente de forma externa a través de una página web que la biblioteca ya se encarga de mostrar.

```
    boolean respuesta = ShareDialog.canShow(ShareLinkContent.class);

    boolean respuesta = ShareDialog.canShow(SharePhotoContent.class);
```

- Para realizar una publicación el método siempre es el mismo, la única diferencia es la definición del objeto “content” que representa aquello que queremos publicar.

```
this.elShareDialog.show(content);
```

Esto abrirá una actividad diferente de la nuestra en la que podremos escribir y enviar a Facebook:

- Si queremos publicar en Facebook un enlace a una página web que nos ha gustado, crearemos un objeto "content" utilizando "ShareLinkContent.Builder"

```
ShareLinkContent content = new ShareLinkContent.Builder()
        .setContentUrl(Uri.parse("https://developers.facebook.com"))
        .setContentDescription("me ha gustado esta página")
        .build();
```

- Si queremos publicar un mensaje, podemos utilizar el ShareLinkContent sin especificar el enlace URL.

```
ShareLinkContent content = new ShareLinkContent.Builder().build();
```

- Si queremos publicar una foto, utilizaremos SharePhoto.builder

```
Bitmap Image =SharePhoto photo = new SharePhoto.Builder()
       .setBitmap(image).build();
SharePhotoContent content = new SharePhotoContent.Builder()
       .addPhoto(photo).build();
```

Por completitud, reproducimos el código completo de esta activity que utiliza Share Dialog:

```
package org.jordi.ejemplo2fb;

import ….

public class MainActivity extends AppCompatActivity {
   private Button boton2;
   private Button boton3;
    private Button boton1;
    private TextView textoEntrada1;
    private TextView textoSalida1;
    // gestiona los callbacks al FacebookSdk desde el método
    // onActivityResult() de una actividad
    private CallbackManager elCallbackManagerDeFacebook;
```

```
    private ShareDialog elShareDialog;
    // puntero a this para los callback
    private final Activity THIS = this;

    private void conseguirReferenciasAElementosGraficos () {
        boton1 = (Button) findViewById(R.id.button1);
        boton2 = (Button) findViewById(R.id.button2);
        boton3 = (Button) findViewById(R.id.button3);
        textoEntrada1 = (TextView) findViewById(R.id.textoEntrada1);
        textoSalida1 = (TextView) findViewById(R.id.textoSalida1);

    }

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("cuandrav.onCreate()", " .onCreate() llamado");
        // inicializar FacebookSDK
        FacebookSdk.sdkInitialize(this.getApplicationContext());
        // pongo el contenido visual de la actividad (hacer antes que
        // findViewById () y después de inicializar FacebookSDK)
        this.setContentView(R.layout.activity_main);
        conseguirReferenciasAElementosGraficos();
        // crear callback manager de Facebook
        this.elCallbackManagerDeFacebook = CallbackMana-
ger.Factory.create();
        //  crear objeto share dialog
        this.elShareDialog = new ShareDialog(this);

this.elShareDialog.registerCallback(this.elCallbackManagerDeFacebook, new
FacebookCallback<Sharer.Result>() {

            @Override
            public void onSuccess(Sharer.Result result) {
                Toast.makeText(THIS, "Sharer onSuccess()",
                               Toast.LENGTH_LONG).show();
            }

            @Override
            public void onCancel() {}

            @Override
            public void onError(FacebookException error) {
                Toast.makeText(THIS, "Sharer onError(): " +
                             error.toString(), Toast.LENGTH_LONG).show();
            }
        });
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
                                                          Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        this.elCallbackManagerDeFacebook.onActivityResult(requestCode,
```

```
                resultCode, data);
    }

    private void publicarMensajeConIntent () {
        Intent shareIntent = new Intent();
        shareIntent.setAction(Intent.ACTION_SEND);
        startActivityForResult(Intent.createChooser(shareIntent, "Share"),
              1234); //requestId);
    }

    public void boton1_pulsado(View quien) {
        Log.d("cuandrav.boton1_pulsado", " llamado ");
        textoSalida1.setText("boton1_pulsado");
        this.publicarMensajeConIntent();
    }

    private boolean puedoUtilizarShareDialogParaPublicarMensaje () {
        return puedoUtilizarShareDialogParaPublicarLink();
    }

    private boolean puedoUtilizarShareDialogParaPublicarLink () {
        return ShareDialog.canShow(ShareLinkContent.class);
    }

    private boolean puedoUtilizarShareDialogParaPublicarFoto () {
        return ShareDialog.canShow(SharePhotoContent.class);
    }

    public void boton2_pulsado(View quien) {
        Log.d("cuandrav.boton2_pulsado", " llamado ");
        textoSalida1.setText("boton2_pulsado");
        // llamar al metodo para publicar
        this.publicarMensajeConShareDialog();
    }

    private void publicarMensajeConShareDialog () {
        // https://developers.facebook.com/docs/android/share -> Using the
Share Dialog
        if ( ! puedoUtilizarShareDialogParaPublicarMensaje() ) {
            Log.d("cuandrav.boton2_pul()",
                        " ¡¡¡ No puedo utilizar share dialog !!!");
            return;
        }
        // llamar a share dialog aunque utilizamos ShareLinkContent,
        // al no poner link publica un mensaje
        ShareLinkContent content = new ShareLinkContent.Builder().build();
        this.elShareDialog.show(content);
    }

    public void boton3_pulsado(View quien) {
        Log.d("cuandrav.boton3_pulsado", " llamado ");
        textoSalida1.setText("boton3_pulsado");
        // llamar al metodo para publicar foto
```

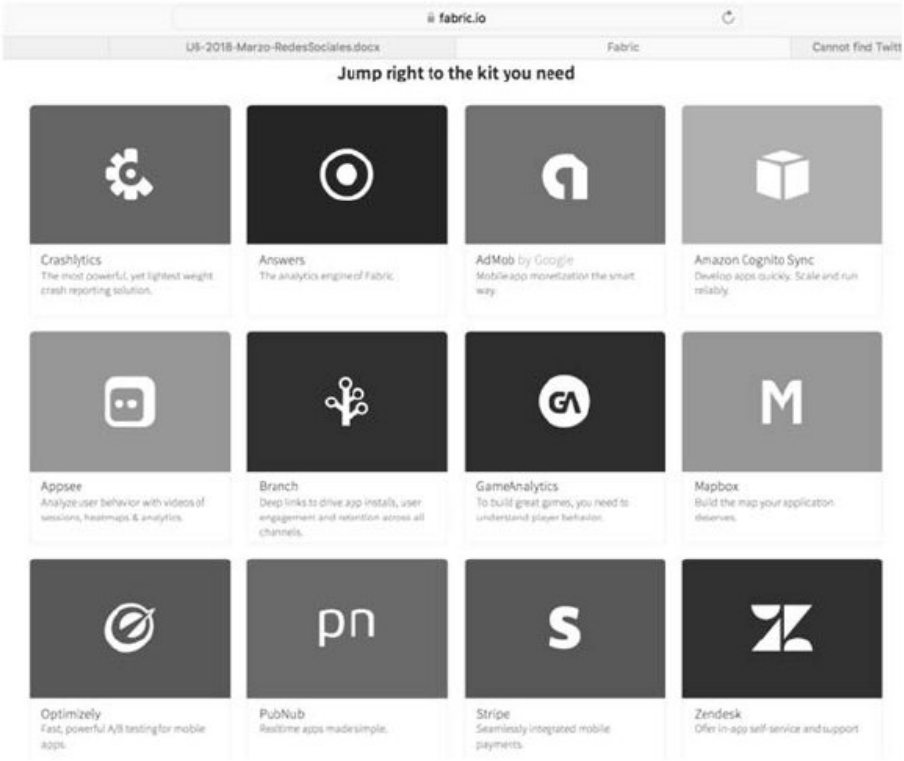
```
        this.publicarFotoConShareDialog();
    }

    private void publicarFotoConShareDialog () {
        // https://developers.facebook.com/docs/android/share -> Using the
Share Dialog
        if ( ! puedoUtilizarShareDialogParaPublicarFoto() ) {
            return;
        }
        // cojo una imagen directamente de los recursos para publicarla
        Bitmap image = BitmapFactory.decodeResource(
                getResources(), R.drawable.com_facebook_logo);
        // monto la petición
        SharePhoto photo = new SharePhoto.Builder()
                .setBitmap(image).build();
        SharePhotoContent content = new SharePhotoContent.Builder()
                .addPhoto(photo).build();
        this.elShareDialog.show(content);
    }
}
```

Práctica: Escribe una aplicación que pueda publicar mensajes y fotos en Facebook, utilizando los ejemplos anteriores. (Previamente hay que haber realizado el ejercicio para dar de alta la aplicación en Facebook Developers).

Preguntas de repaso: Facebook

6.2. Android y Twitter

Twitter desarrolló un conjunto de utilidades para integrar aplicaciones móviles con su plataforma, con el nombre de Fabric. Esta plataforma ha sido adquirida por Google. Una de las utilidades más destacadas de Fabric es Crashlytics (www.crashlytics.com), una herramienta para generar informes de errores y fallos en aplicaciones móviles, facilitando su corrección. Pero hay otros muchos kits (fabric.io/kits) disponibles



que incluyen mapas, pagos, almacenamiento en la nube, acreditación de usuarios. Recientemente, tras la compra de Fabric por parte de Google que hemos comentado, el **TwitterKit ya no forma parte de Fabric**. Sigue existiendo, por supuesto, una biblioteca para poder usar servicios de Twitter desde Android: **https://github.com/twitter/twitter-kit-android/wiki**

6.2.1. Instalando Twitter Kit en Android Studio

En primer lugar, hay que acceder a las bibliotecas de Twitter. Para ello, en el fichero build.gradle de la app, añadiremos en la sección dependencies:

```
compile 'com.twitter.sdk.android:twitter:3.1.1'

compile 'com.squareup.retrofit:retrofit:1.9.0'
```

La segunda biblioteca (retrofit) la utilizaremos si para manipular ficheros de imágenes que queremos enviar a Twitter usando su API REST.

Y en la sección ANDROID añadiremos:

```
repositories {
    jcenter()
}
```

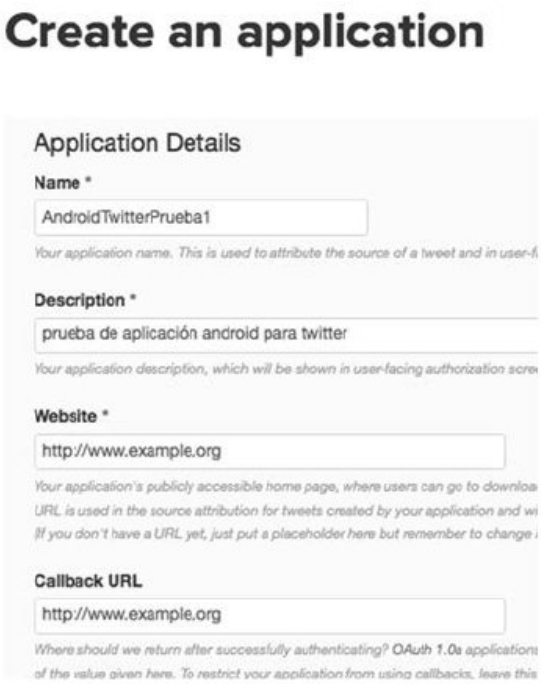
Ejercicio: Empieza un proyecto y realiza los anteriores cambios

6.2.2. Configurando nuestra aplicación en Twitter Apps

Cada aplicación de Android integrada con Twitter debe estar dada de alta en esta red social. Para ellos vamos a:

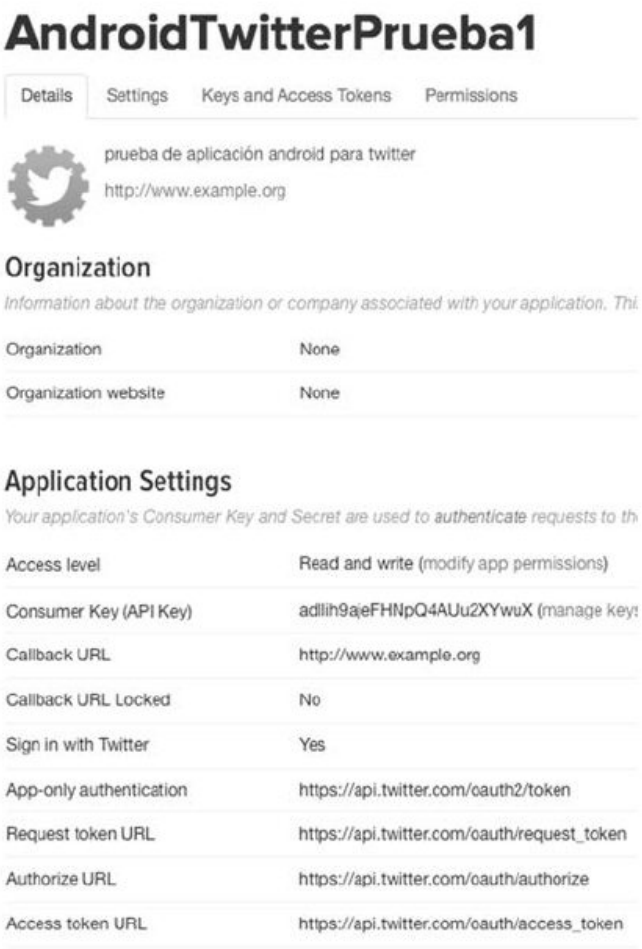
https://apps.twitter.com

y pulsamos en el botón **Create New App**. En la pantalla que aparece, rellenamos los 4 campos: nombre, descripción, sitio web y callback URL. Es importante rellenar también el callback, aunque sea con un URL ficticio como http://www.example.com, ya que esto va a permitir volver a nuestra aplicación de forma transparente desde la página web de Twitter en donde nos autentificaremos.

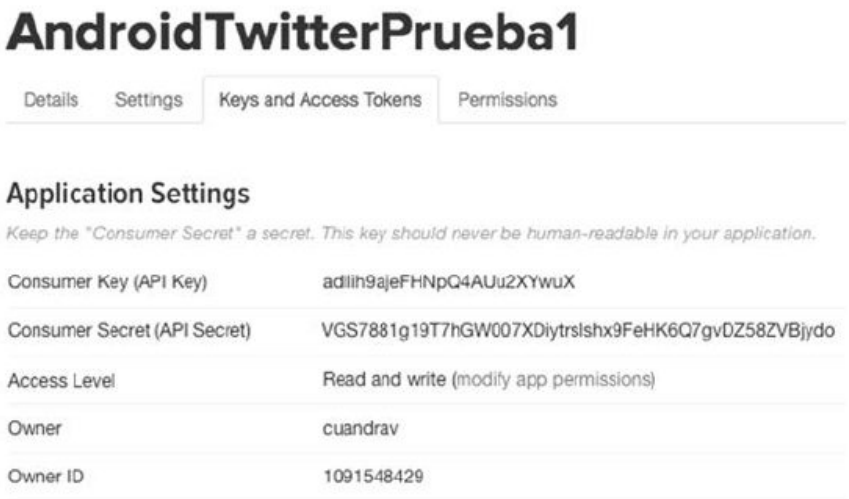


Ahora vamos a la pestaña **Permissions** de nuestra aplicación:

y marcamos Read and Write, pulsamos luego el botón **Update settings**. En la pestaña **Details** veremos algo como esto:



Finalmente, vamos a la pestaña **Keys and Access Tokens** para anotarnos Consumer Key (API Key) y Consumer Secret (API Secret), que identifican nuestra aplicación.



Ejercicio: Da de alta una aplicación en Twitter Apps (https://apps.twitter.com) como se ha explicado antes

6.2.3. Aplicación de Ejemplo

Igual que ocurría con Facebook, recomendamos tener la aplicación oficial de Twitter instalada en nuestro teléfono.

El código que vamos a estudiar como ejemplo simplemente sirve para hacer login en Twitter y publicar mensajes en nombre de un usuario existente.

Pasos a seguir:

- Añadir los valores "consumer key" y "consumer secret" que se encuentran en la secciónKeys and Access Tokens de nuestra aplicación en Twitter Apps al fichero strings.xml

```
<string name="com.twitter.sdk.android.CONSUMER_KEY">adlAUu2XYwuX</string>
<string
name="com.twitter.sdk.android.CONSUMER_SECRET">VGS7881g19T</string>
```

- Recordar añadir a AndroidManifest.xml el permiso para usar la red:

```
<uses-permission android:name="android.permission.INTERNET" />
```

- En el layout de la actividad (activity_main.xml) debemos incluir este código para utilizar el botón oficial de Twitter:

```
<com.twitter.sdk.android.core.identity.TwitterLoginButton
     android:id="@+id/twitter_login_button"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"/>
```

- En el código java de MainActivity.java, en onCreate() inicializamos la biblioteca:

```
Twitter.initialize( this);
```

- También en onCreate() instalararemos un callback en el botón oficial de Twitter para conocer el resultado del login. (Véase que podemos consultar datos del usuario autentificado en result.data).

```
botonEnviarATwitter = (Button) findViewById(R.id.boton_EnviarATwitter);
botonLoginTwitter.setCallback(new Callback<TwitterSession>() {
    @Override
    public void success(Result<TwitterSession> result) {
        Toast.makeText(THIS, "Autenticado en twitter: " + re-
sult.data.getUserName(), Toast.LENGTH_LONG).show();
    }

    @Override public void failure(TwitterException e) {
        Toast.makeText(THIS, "Fallo en autentificación: " +
                        e.getMessage(), Toast.LENGTH_LONG).show();
    }
});
```

- Reescribimos onActivityResult() para informar al botón oficial de Twitter (que es quien gestiona el proceso de login).

```
@Override protected void onActivityResult(int requestCode,
   int resultCode, Intent data) {

    super.onActivityResult(requestCode, resultCode, data);
    botonLoginTwitter.onActivityResult(requestCode, resultCode, data);
}
```

Publicar un tweet con imagen mediante TweetComposer

Se puede publicar un tweet acompañado de una imagen (si se quiere) mediante TweetComposer. Esta clase lo que hace en realidad es lanzar un intent que captura la app oficial de Twitter de nuestro teléfono y utilizar una de sus pantallas para llevar a cabo la tarea.

```
TweetComposer.Builder builder = new TweetComposer.Builder(this)
        .text("esto es un tweet");
        //.image(myImageUri); // se puede añadir una imagen dando su
// URI: content://lo/que/sea
builder.show();
```

Esta es, con diferencia, la opción más sencilla. El único inconveniente es que utilizar una actividad externa puede "romper" la estética de nuestra aplicación.

Publicar un tweet directamente desde nuestra aplicación

Podemos publicar un tweet directamente desde nuestra aplicación a Twitter (vía su interfaz REST) así:

```
StatusesService statusesService = Twitter.getApiClient( obtenerSe-
sionDeTwitter() ).getStatusesService();
Call<Tweet> call =  statusesService.update(textoQueEnviar, null, null,
                            null, null, null, null, null, null);
call.enqueue(new Callback<Tweet>() {
    @Override public void success(Result<Tweet> result) {
```

```
        Toast.makeText(THIS, "Tweet publicado: " +
                    result.response.message(), Toast.LENGTH_LONG).show();
    }
    @Override public void failure(TwitterException e) {
        Toast.makeText(THIS, "No se pudo publicar el tweet: "+
                              e.getMessage(), Toast.LENGTH_LONG).show();
    }
});
```

Publicar un tweet con imagen directamente desde nuestra aplicación

Publicar una imagen directamente vía REST necesita bastante código. Para simplificar, recomendaríamos "esconderlo" en una función. Primero hay que copiar la imagen al Twitter. Después de este paso la imagen aún no se ve. Hay que publicar un tweet que la referencie. El siguiente es un código de ejemplo comentado.

```
File photo = null;
try {

    // 1. Abrimos el fichero con la imagen
    // imagen que queremos enviar. Como necesitamos un path (para
    // TypedFile) debe estar fuera de /res o /assets porque estos
    // estan dentro del .apk y NO tiene path
    photo = new File ("/storage/sdcard0/DCIM/100ANDRO/DSC_0001.jpg");
} catch (Exception e) {
    Log.d("miApp", "enviarImagen : excepcion: " + e.getMessage());
    return;
}

// 2. ponemos el fichero en un TypedFile
 TypedFile typedFile = new TypedFile("image/jpg", photo);

// 3. obtenemos referencia al media service
MediaService ms = Twitter.getApiClient( obtenerSesionDeTwitter() )
            .getMediaService();

// 3.1 ponemos la foto en el request body de la petición
okhttp3.RequestBody requestBody = okhttp3.RequestBody.create(
            MediaType.parse ("image/png"), photo);

// 4. con el media service: enviamos la foto a Twitter
Call<Media> call1 = ms.upload(
        requestBody, // foto que enviamos
        null, null);

call1.enqueue (new Callback<Media>() {
 @Override public void success(Result<Media> mediaResult) {
   // he tenido éxito:
   Toast.makeText(THIS, "imagen publicada: " +
   mediaResult.response.toString(), Toast.LENGTH_LONG);

   // 5. como he tenido éxito, la foto está en twitter, pero no en el
   // timeline (no se ve) he de escribir un tweet referenciando la foto
   // 6. obtengo referencia al status service}});
```