Volumen II

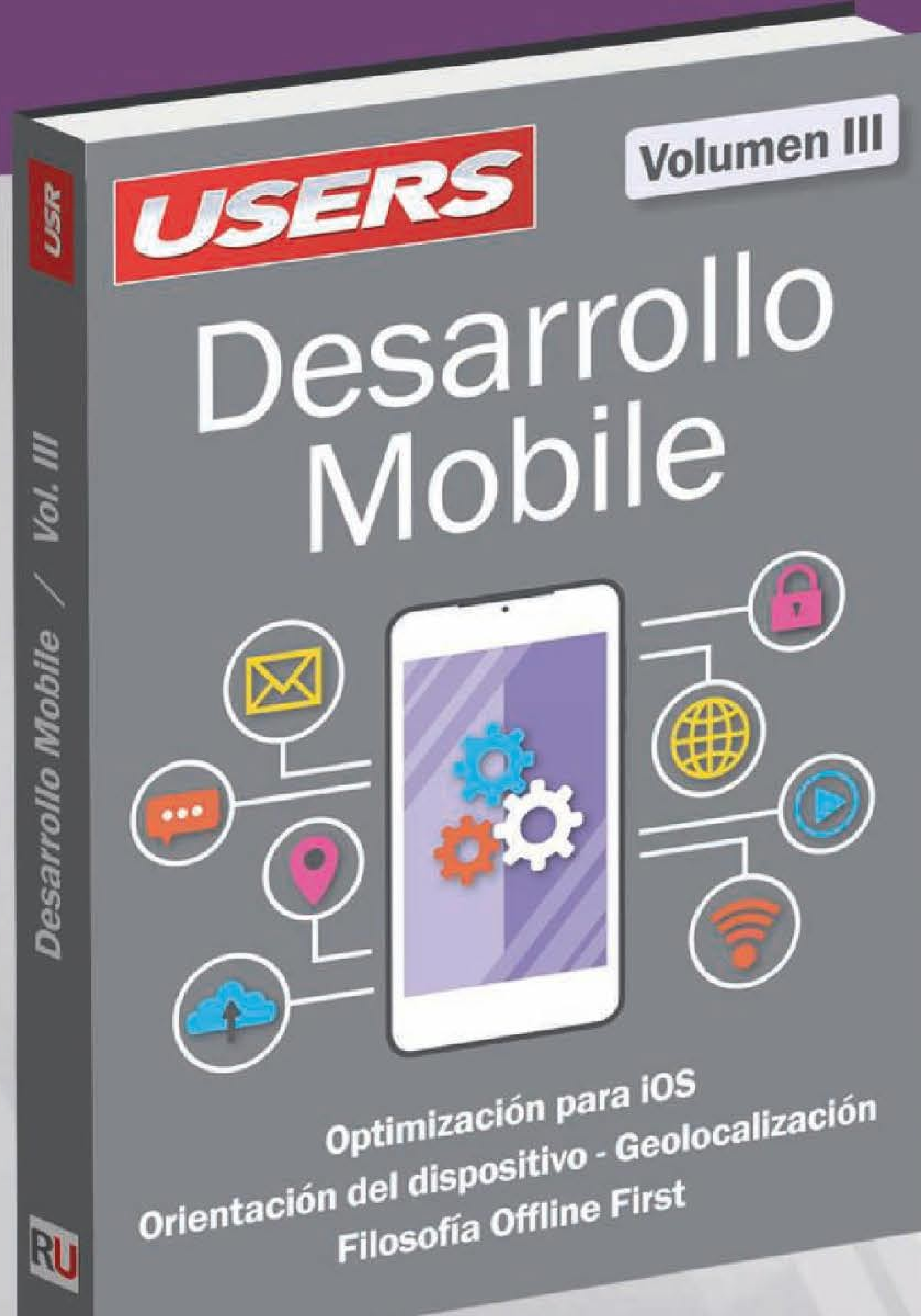
Desarrollo Mobile

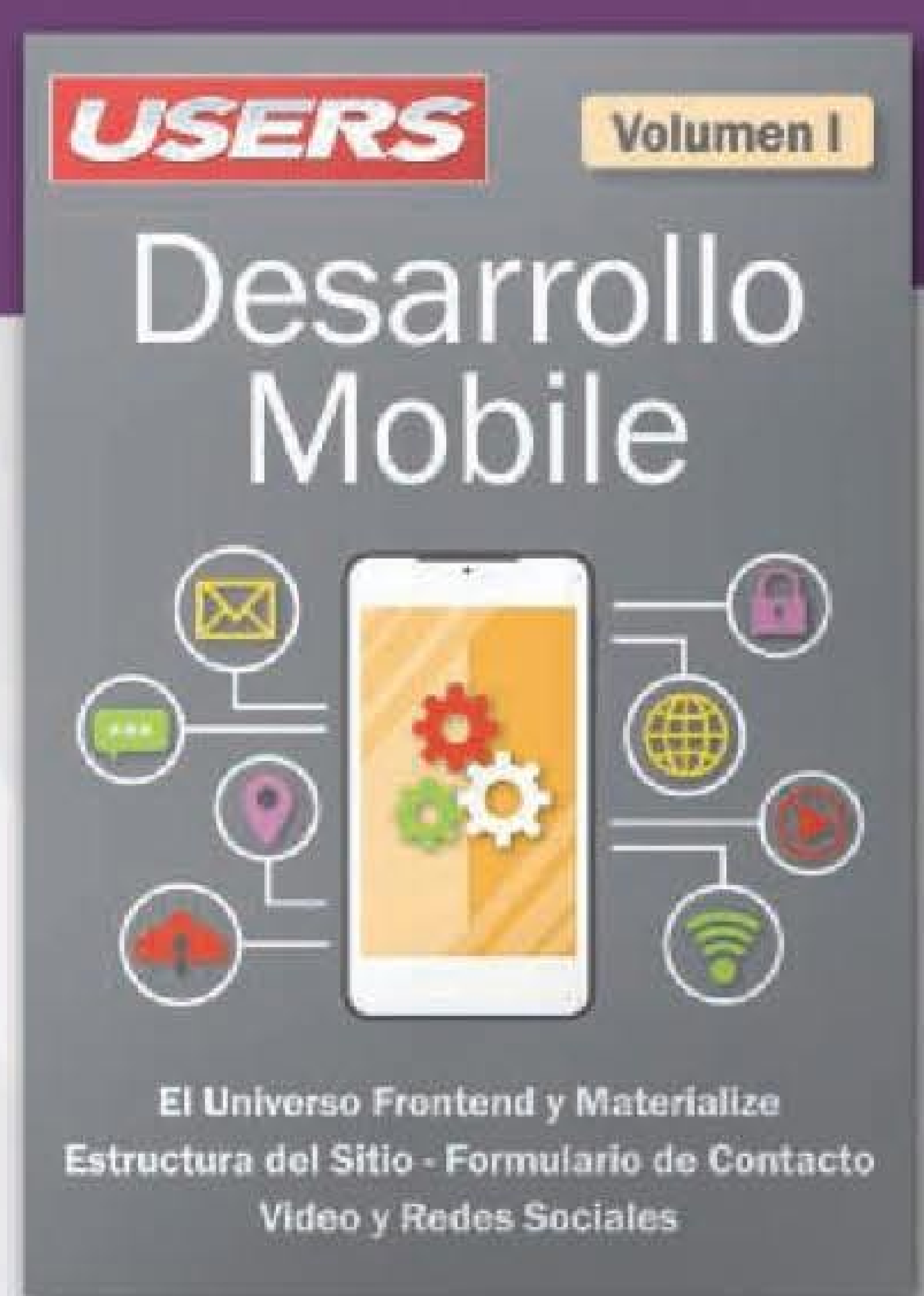
Sidebar, tablas y contenedores

Componentes anidados

Datos remotos - Mapas

CURSO DE
DESARROLLO MOBILE





Diseña aplicaciones web Mobile First, desde una web estática hasta 100% dinámica. Convierte tu web en PWA, la próxima generación de aplicaciones móviles y de escritorio.

Desarrollo Mobile Vol.II

Sidebar, tablas y contenedores
Componentes anidados
Datos remotos - Mapas

Título: Desarrollo Mobile - Vol. II / **Autor**: Fernando Omar Luna
Coordinador editorial: Miguel Lederkremer / **Edición**: Claudio Peña
Diseño y Maquetado: Marina Mozzetti / **Colección**: USERS ebooks - LPCU295

Copyright © MMXIX. Es una publicación de Six Ediciones. Hecho el depósito que marca la ley 11723. Todos los derechos reservados. Esta publicación no puede ser reproducida ni en todo ni en parte, por ningún medio actual o futuro, sin el permiso previo y por escrito de Six Ediciones. Su infracción está penada por las leyes 11723 y 25446. La editorial no asume responsabilidad alguna por cualquier consecuencia derivada de la fabricación, funcionamiento y/o utilización de los servicios y productos que se describen y/o analizan. Todas las marcas mencionadas en este libro son propiedad exclusiva de sus respectivos dueños. Libro de edición argentina.

Luna, Fernando O.
Desarrollo mobile 2 / Fernando O. Luna. - 1a ed . - Ciudad Autónoma de Buenos Aires : Six Ediciones, 2019.
Libro digital, PDF

Archivo Digital: online
ISBN 978-987-4958-20-4

1. Diseño de Software. 2. Aplicaciones Web. I. Título.
CDD 005.25

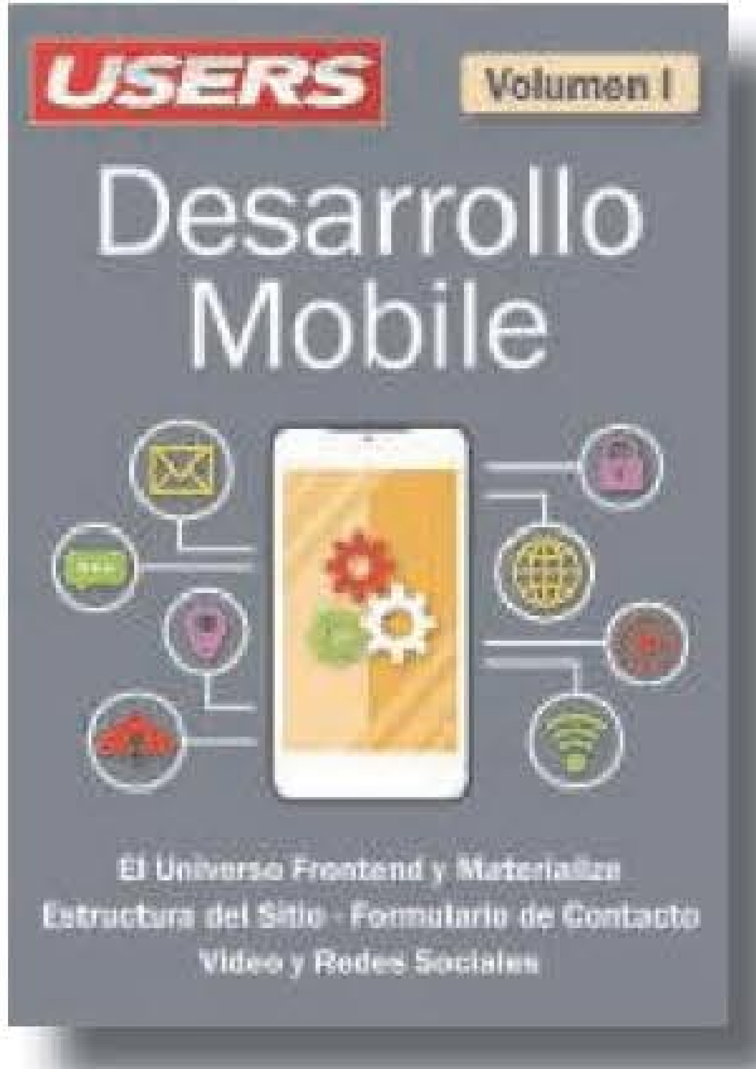
96% 23:17

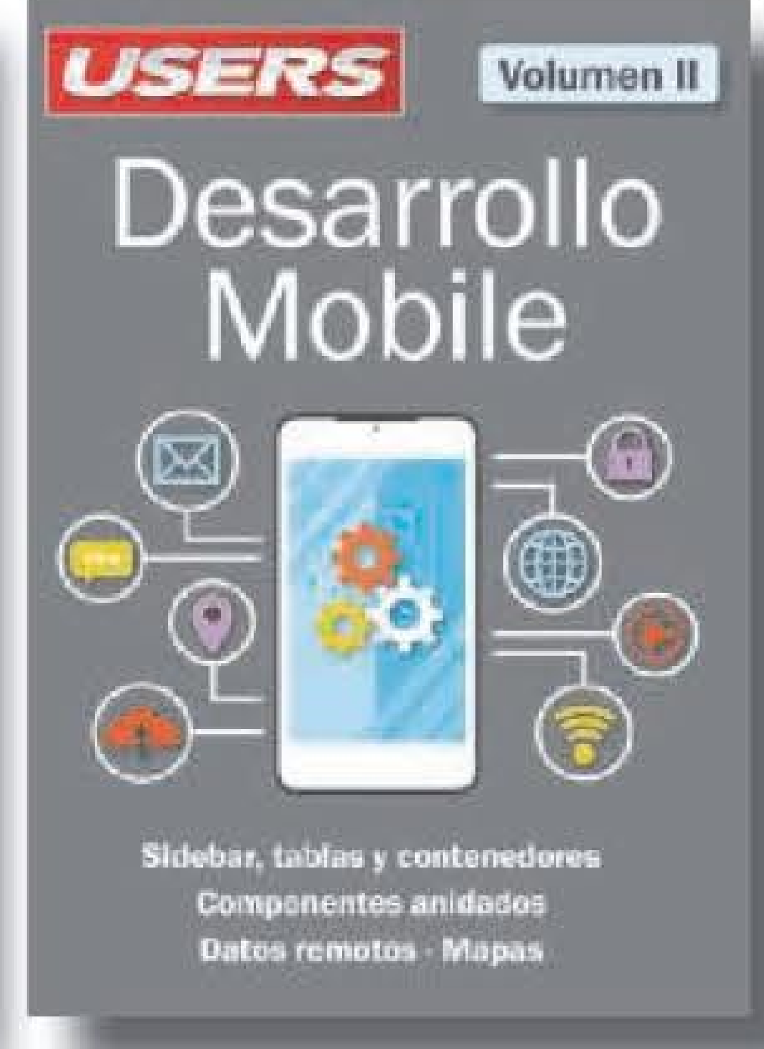
Acerca de este curso

Este curso se divide en tres e-books. En el **primer** volumen conocimos Materialize CSS, y comenzamos a ver cómo implementar la barra de navegación, los títulos, los párrafos y las imágenes en el diseño de un sitio web adaptado para móviles. Luego profundizamos en la manera de integrar audio y video en nuestros desarrollos, y en la forma de sumar contenido desde y hacia las diferentes redes sociales más populares a la fecha.

En el **segundo** volumen de la colección potenciaremos la barra de navegación, y la integraremos en modo lateral, tal como la utilizan las aplicaciones nativas. Luego aprenderemos las diferentes opciones de botones que podemos aplicar con Materialize, cómo integrar mensajes de tipo tooltip y toast, cómo integrar mapas dinámicos, de qué forma leer y mostrar datos utilizando JSoN, y cómo explotar al máximo las herramientas para desarrollador de Google Chrome.

En el **tercer** volumen aprenderemos a acondicionar nuestros desarrollos web adaptándolos a las diferentes plataformas móviles. Veremos cómo integrar metatags para la plataforma iOS, qué nos ofrece Onsen.io, cómo convertir nuestra web en una PWA e instalarla en Android e iOS, y de qué modo hacer una web 100% desconectada de Internet. También veremos cómo optimizar las imágenes integradas en una web y la forma de ofrecer una experiencia 360º con imágenes panorámicas.





Prólogo

La Web es todo. Desde ella no solo nos informamos, sino que también compramos productos y servicios, pagamos impuestos, consumimos películas y música, realizamos reservas en restaurantes y volcamos opiniones sobre lugares visitados. Además leemos mails, ponemos recordatorios en un calendario e interactuamos varias veces al día con las redes sociales, entre un sinfín de cosas más.

Y quienes desarrollamos software sabemos que la Web fue, es y será durante muchísimo tiempo más el puntapié inicial de nuestro día a día. Cada vez estará más comprometida con nosotros y, a su vez, más integrada; y todo esto seguirá ocurriendo desde nuestro bolsillo, desde nuestro dispositivo móvil. Por eso, los invitamos a recorrer la serie de e-books que integran esta obra, para conocer los principales secretos del desarrollo web para dispositivos móviles y saber cómo encarar el diseño e integración de las PWA, la próxima generación de aplicaciones móviles y de escritorio, que ya lleva un tiempo conquistando las pantallas móviles.

Autor

Fernando Omar Luna

Es Analista de Sistemas y Technical Writer. Lleva más de 25 años desarrollando software para diferentes plataformas, y algo más de una década colaborando como autor de publicaciones técnicas orientadas a la programación y las nuevas tecnologías. Su pasión se divide entre la programación, la electrónica y la educación, siendo este último el motivo por el cual se encuentra cursando un profesorado que le permita ejercer profesionalmente su pasión por el desarrollo del software.

23:17 96%

Contenido

01 Sidebar, botones e iconos con diseño Material....6

Sidenav: barra de navegación / Cómo implementar Sidenav desde HTML / Ejercicio práctico: barra de navegación lateral para móviles / Parámetros de inicialización / Declarar variables para los parámetros / Manejo de botones / Raised Buttons / Ejemplo funcional del componente Buttons / Iconos en botones / Flat Buttons / Cambiar el tamaño de los botones / Deshabilitar el botón / Manejo de la clase disabled desde JavaScript / Floating button / Fijar el botón

02 Tablas y contenedores....26

Tablas / La etiqueta Table / Estructura / Clase de la etiqueta Table / Stripped / Highlight / Centered / Responsive / Ejemplo de prueba / Contenedor (container) / Grid / Ejercicio práctico / Offset / Más clases para manipular filas y columnas / Clases sections y dividers

03 Tarjetas....42

Card panel / Carrusel de imágenes / Propiedades y valores de inicialización de Carousel / Métodos / Ejercicio práctico: compra -venta / Controlar la carga del documento HTML / Elementos expansibles / Componentes HTML / Inicialización del componente / Propiedades y valores del componente Collapsible / Métodos del componente Collapsible / Tipos de efectos en Collapsible / Preselected

04 Mensajes de información e interacción....56

Colecciones / Ejemplo práctico: navegación por categorías / History.back / Location.href / Navegación hacia atrás / Hipervínculo hacia el producto / Colección activa / Avatares / Badges / Ejemplos del uso de la clase badge / Tooltip / Tag HTML / Manos a la obra / Inicialización JavaScript / Toast: mensajes temporales / Propiedades personalizables / Propiedades personalizadas / Modo de invocarlo / Ejemplo práctico: menú de Compupedia / Ventanas modales / Ejercicio práctico: ventana de comentarios / Estructura HTML de Modal Dialog / Estructura JS de Modal Dialog / Abrir la ventana Modal / Control de caracteres

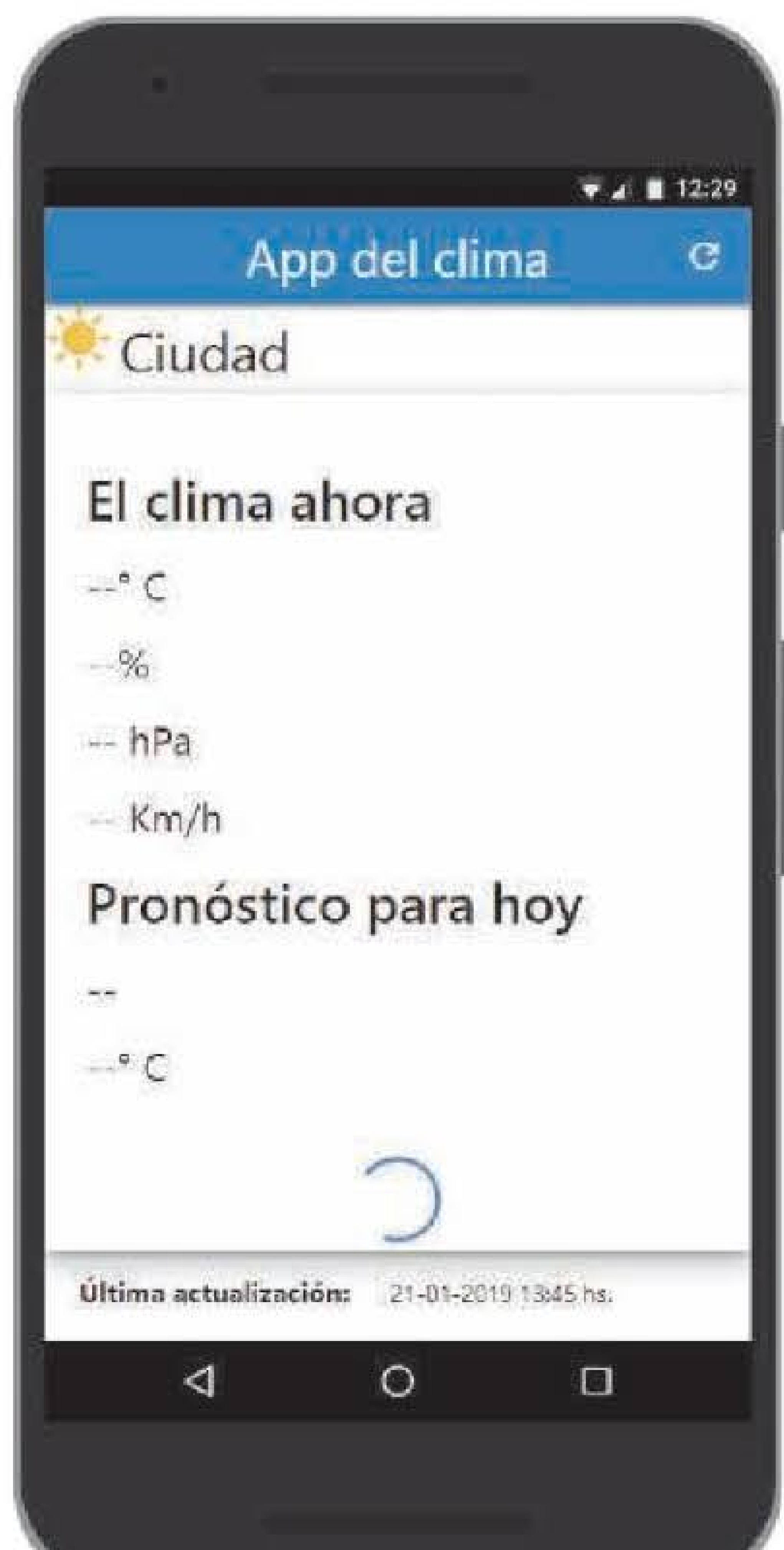
Contenido

Integración de datos remotos..80

*La ténica Ajax / Ejemplo de código JSoN /
Segmentos de aplicación / Servicios Web / Diseño de una aplicación
para el clima / Open Weather Map / Generar una API Key /
Mostrar información del clima en pantalla /
Círculo progresivo de carga / Posibles estados de servidor /
JSoN Formatter / Procesar la respuesta JSoN /
Registrar la última hora*

Geolocalización y mapas dinámicos.........................94

*Google Maps / Bing Maps / Open Street Map / HERE Maps /
Las ventajas de HERE Maps / Suscripción y gestión de claves /
Paso a paso: suscripción y gestión de claves /
Sistema de dcumentación y ayuda / Estructura de la API /
El objeto Map / Proyecto práctico: localizador de cafeterías /
Documento HTML base / Qué haremos desde JS /
Agregar la referencia hacia la API de Mapas / Archivo de proyecto /
Hardcoding / Evento del listado de cafeterías /*



96% 23:17

01 Sidebar, botones e iconos con diseño Material

Continuamos avanzando en el desarrollo web para dispositivos móviles. Conoceremos a continuación el uso de la barra de navegación lateral, los distintos tipos de botones que podemos crear y las diferentes formas de aplicar iconos en nuestros diseños.

El primer ejercicio práctico de diseño web del e-book 1 (Capítulo 02) se lo dedicamos a la creación de una barra de navegación responsiva, que visualiza botones con textos en pantallas amplias, y botones con icono solamente en pantallas medianas y pequeñas.

En este nuevo libro, comenzaremos aprendiendo qué otras alternativas de barras de navegación encontramos en el uso de Materialize CSS.

Sidenav: barra de navegación lateral

Tal como su nombre lo indica, este componente permite crear barras de navegación laterales, como las que encontramos en casi cualquier aplicación móvil, y que también suelen utilizarse en algunos sitios web de escritorio.

Como podemos ver en las imagenes, **Sidenav** está presente tanto en las web móviles (**PWA de Twitter**), como en las aplicaciones nativas de Android (**APK Google Drive**).

Este componente permite desplegar una barra de menús desde el lateral derecho o izquierdo de la pantalla, para así contar con todo el display limpio para la aplicación web o nativa, sin tener que resignar espacios dedicados a la barra de menús.

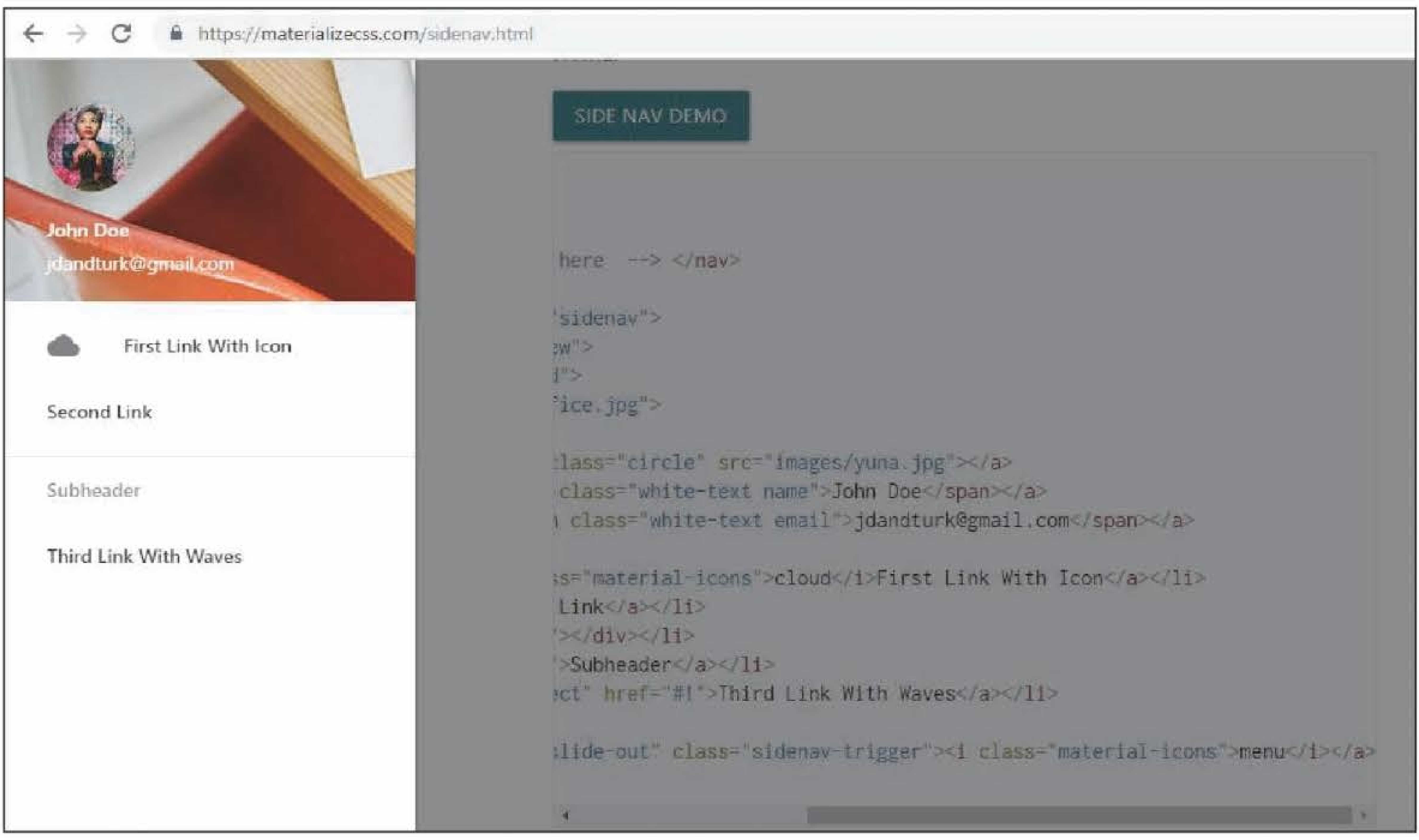
Cómo implementar Sidenav desde HTML

Materialize CSS incluye una clase CSS dedicada a la creación de la barra de navegación lateral. Se llama **sidenav**, y se debe aplicar como clase a un tag **<ul>**, usualmente a continuación o dentro de un tag **<nav>**. Lo que hace dicha clase es formatear con una estructura lógica el uso de viñetas HTML, para que cada uno de los componentes de dicha viñeta asimile una opción de menú de la barra de navegación lateral. Veamos a continuación un ejemplo de su código:

96% 23:17


```
<ul id=”slide-out” class=”sidenav”>
  <li>
    <div class=”user-view”>
    <div class="background”><img src=”images/office.jpg”></div>
    <a href=”#user”><img class=”circle” src=”images/yuna.jpg”></a>
    <a href=”#name”><span class=”white-text name”>John Doe</span></a>
    <a href=”#email”><span class=”white-text email”>jdandturk@gmail.
com</span></a>
    </div>
  </li>
  <li><a href=”#!”><i class=”material-icons”>cloud</i>First Link With
Icon</a></li>
  <li><a href=”#!”>Second Link</a></li>
  <li><div class=”divider”></div></li>
  <li><a class=”subheader”>Subheader</a></li>
  <li><a class=”waves-effect” href=”#!”>Third Link With Waves</a></li>
</ul>
```

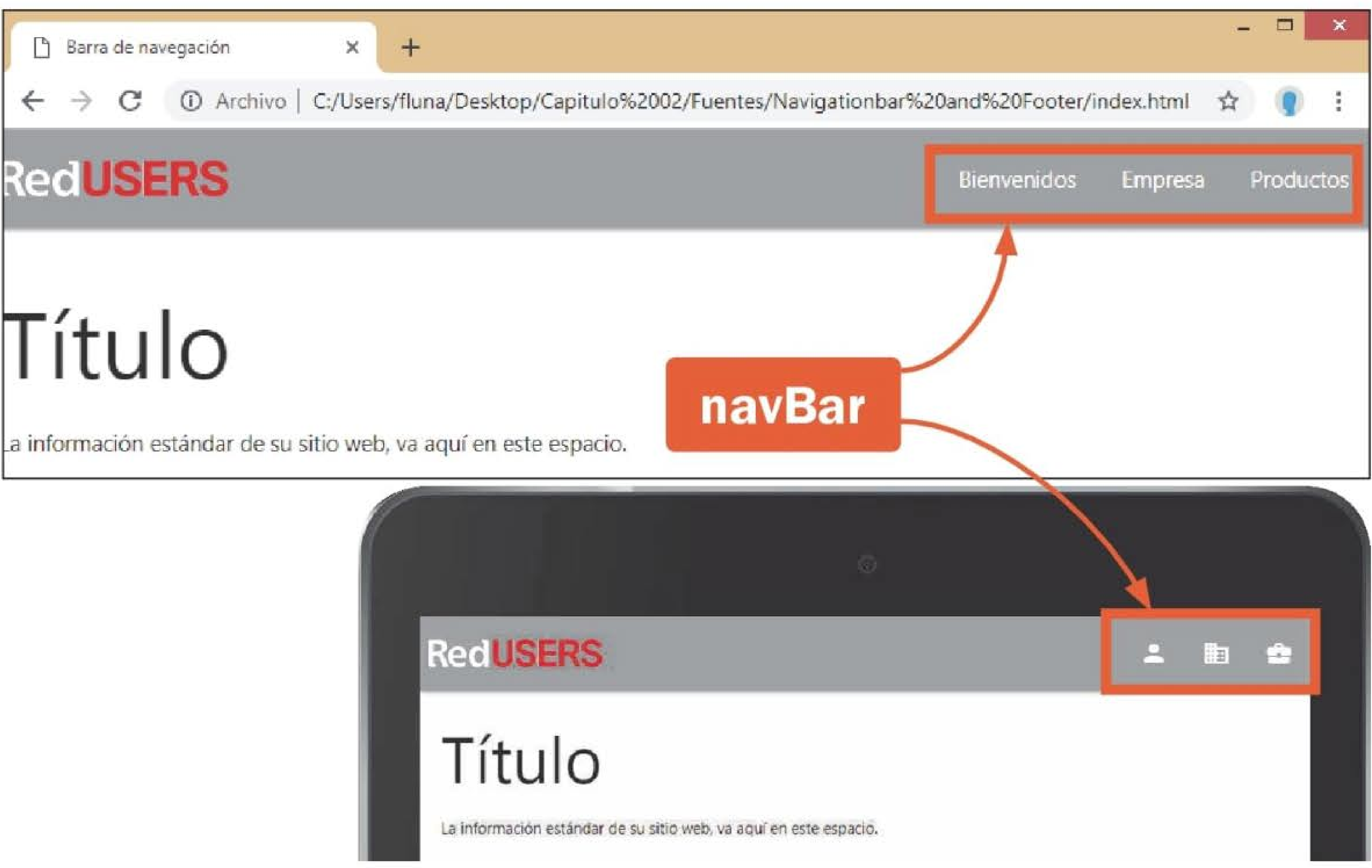
El código de ejemplo anterior es el que podemos encontrar en el apartado oficial de Materialize CSS: https://materializecss.com/sidenav.html. El resultado es el que vemos en la siguiente imagen:



Como podemos apreciar, **Sidenav** tiene varios condimentos que nos permitirán armar un menú completo y complejo, para poder cubrir cualquier tipo de necesidad de desarrollo.

Ejercicio práctico: barra de navegación lateral para móviles

Para poder afianzar los conocimientos referentes a esta clase, realizaremos a continuación, un ejercicio práctico donde modificaremos la estructura de una web responsiva, para habilitar en ella un menú lateral de navegación solo cuando dicha web sea cargada desde un smartphone.



Descargamos del repositorio de archivos de esta obra el denominado **NavToSidebar.zip**. Este contiene una ligera modificación del ejercicio realizado en el **E-book 1, Capítulo 02**, cuando creamos una barra de navegación usando **navBar**.

Descomprimimos el archivo en nuestra computadora y creamos un nuevo proyecto en **Visual Studio Code**, referenciando la carpeta que contiene este proyecto descargado. Editamos el archivo **index.html** y ubicamos, dentro de él, el código que comienza con la siguiente línea:

```
<ul id="mobile-menu" class="hide-on-large-only right">
...
```

Seleccionamos todo el bloque de código, desde la apertura del tag **
** hasta su cierre **
**, y lo comentamos o borramos directamente. Seguido a esto, en el espacio donde se encontraba este código agregamos lo siguiente:

```
<ul id="slide-out" class="sidenav sidenav-fixed hide-on-med-and-up">
...
</ul>
```

Del bloque de código anterior, obviemos los tres puntos centrales. Lo que aquí hacemos es crear un elemento HTML **sidenav**, con la clase **sidenav-fixed**, la cual lo hará fijo; y la clase **hide-on-med-and-up**, que lo ocultará cuando la web se despliegue en pantallas que no sean de smartphones. Su atributo **id** será **slide-out**, y estará referenciado con otros componentes HTML y con el código CSS que le indica su comportamiento.

Ahora, reemplazamos los tres puntos visualizados en el bloque de código anterior, por el siguiente fragmento:

```
<li class="grey left" style="height: 56px" id="menulateralcerrar"><a
class="sidenav-close" href="#!"><i class="material-icons">close</i></
a></li>
<li class="grey" style="height: 56px"><a href="#" class="center"><img
src="images/logo_redusers.png" width="50%"/></a></li>
<li><a class="waves-effect" href="#!"><i class="material-
icons">person</i>Bienvenidos</a></li>
<li><a class="waves-effect" href="#!"><i class="material-
icons">business</i>La Empresa</a></li>
<li><a class="waves-effect" href="#!"><i class="material-
icons">business_center</i>Productos</a></li>
```

Lo que hacemos con este otro bloque de código es crear un elemento HTML **<li>** de menú, dentro del panel lateral, con cada uno de los elementos que lo compondrán. Veamos a continuación la **Tabla 1** con la explicación de cada uno.

TABLA 1

Elemento	Clase	Icono o Imagen	Descripción
id menulateralcerrar	**sidenav-close**	**close**	Visualiza el botón Cerrar, que se ocupará de ocultar la barra lateral cuando el usuario lo presione.
	grey	**logo_redusers.png**	En este caso, usamos el logo como encabezado de menú. De igual forma, Materialize CSS posee varias opciones, como mostrar los datos de un usuario si este se encuentra logueado en la web.
	waves-effect	**person**	Genera un punto de menú en el cual podemos adicionar un link hacia un documento HTML relacionado.
		bussines	Genera un punto de menú en el cual podemos adicionar un link hacia un documento HTML relacionado.
		bussines_center	Genera un punto de menú en el cual podemos adicionar un link hacia un documento HTML relacionado.

Para hacer más cómoda la lectura del menú de opciones, entre un punto de menú y el otro, podemos intercalar la siguiente línea de código:

```
<li class="grey"><div class="divider"></div></li>
```

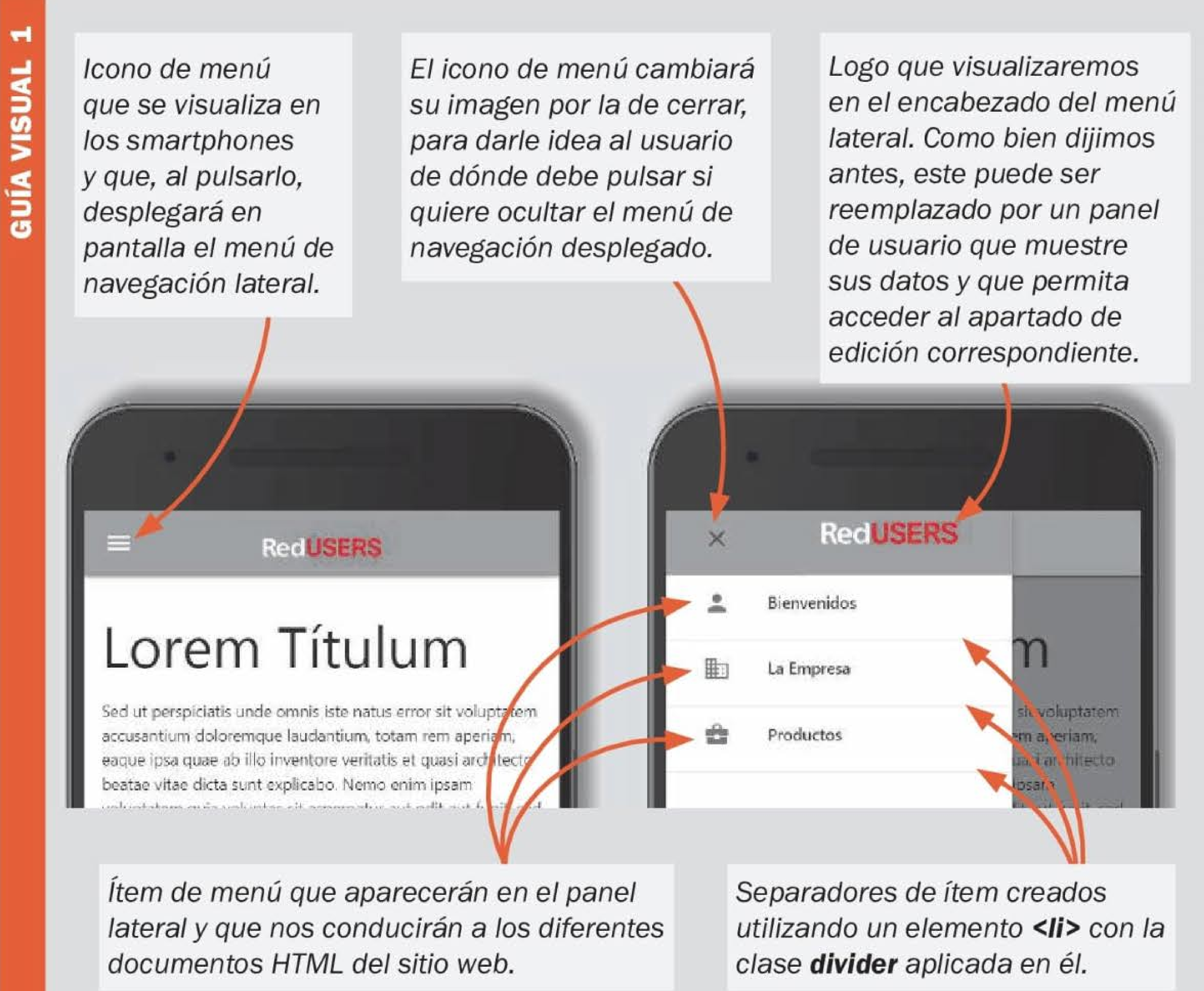
Este último bloque de código generará una barra espaciadora entre los ítem, para una mejor lectura e interacción con ellos. Finalmente, seguido al cierre del tag **
**, agregamos esta otra línea:

```
<a href="#" data-target="slide-out" class="sidenav-trigger hide-on-med-
and-up" id="menulateralabrir"><i class="material-icons">menu</i></a>
```

Se trata del botón de menú que se visualizará en las pantallas de los smartphones y que nos permitirá acceder al menú lateral.

Veamos a continuación (**Guía Visual 1**) una infografía de cómo quedará constituido el menú y de qué manera se visualizarán sus elementos.

96% 23:17



Parámetros de inicialización

Sidenav es un componente que no funciona tan solo con su clase CSS; necesita una serie de parámetros que debemos definir previamente, para luego inicializar el componente desde JavaScript y hacer que, finalmente, Sidenav pueda desplegarse como una barra que se desliza desde un lateral de la pantalla.

En la **Tabla 2** vemos los diferentes parámetros que podemos utilizar.

La implementación de estos parámetros desde JavaScript es muy simple. Lo primero que debemos hacer es seleccionar cuáles deseamos modificar. Luego, los agregamos en una variable JS, que inicializaremos pasando dicha variable como parámetro del método JS de inicialización. Para comprender mejor este punto, veamos un ejemplo de código en la imagen de la página siguiente (ver figura).

En este caso, agregaremos los parámetros **inDuration**, **outDuration**, **onOpenStart**, **onCloseEnd** y **preventScrolling**, y les asignaremos diferentes valores, de acuerdo con el tipo de dato que podemos especificar, para finalmente inicializar la barra lateral.

TABLA 2

Parámetro	Tipo	Valor por defecto	Descripción
Edge	String	**left**	Especifica con el parámetro left o right (izquierda o derecha) desde qué lado de la pantalla se despliega el menú lateral.
draggable	Boolean	**true**	Permite gestos del tipo Swipe, para abrir u ocultar la barra de navegación lateral.
inDuration	Number	**250**	Tiempo en milisegundos de la transición de ingreso del panel lateral.
outDuration	Number	**200**	Tiempo en milisegundos de la transición de cierre del panel lateral.
onOpenStart	Function	**null**	Llama una función personalizada cuando la barra lateral inicia su entrada en pantalla.
onOpenEnd	Function	**null**	Llama una función personalizada cuando la barra lateral finaliza su entrada en pantalla.
onCloseStart	Function	**null**	Llama una función personalizada cuando la barra lateral inicia su salida de pantalla.
onCloseEnd	Function	**null**	Llama una función personalizada cuando la barra lateral finaliza su salida de pantalla.
preventScrolling	Boolean	**true**	Previene que la barra lateral se desplace hacia arriba o abajo, si el usuario realiza scroll.

```
document.addEventListener('DOMContentLoaded', function() {
    var parametros = {
        edge: 'left',
        inDuration: 600,
        outDuration: 800,
        preventScrolling: true,
        onOpenStart: function() { l.style.visibility = 'hidden'; }
        onCloseEnd: function() { l.style.visibility = 'visible'; }
    };
    var componente = document.querySelectorAll('.sidenav');

    var m = M.Sidenav.init(componente, parametros);
```

Declarar variables para los parámetros

Creamos el documento JS denominado **barralateral.js**. A continuación, debemos declarar la inicialización del **DOM** HTML para que, una vez cargada toda la página web, recién entonces se interprete e inicie nuestro código JS. Luego, dentro de este bloque de código, declaramos la variable que contendrá estos parámetros, de la siguiente manera:

```
var parametros = {
  edge: 'left',
  inDuration: 600,
  outDuration: 800,
  preventScrolling: true,
  onOpenStart: function() { //especifique un código a ejecutar },
  onCloseEnd: function() { //especifique otro código a ejecutar }
};
```

La variable declarada funciona como un **string** o cadena del tipo **JSoN**, que proveerá al método de inicialización todos los parámetros de comportamiento de la barra de navegación lateral que deseamos personalizar. Como vemos en el código anterior, en los parámetros **onOpenStart** y **onCloseEnd**, declaramos a continuación una función JavaScript, que contendrá un código específico que puede afectar a la barra de navegación en sí o a cualquier otro componente HTML. Entre las llaves que acompañan a cada función, indicaremos el evento que ocurrirá cuando se dispare el trigger asociado a ellas. Para esto, declaramos una nueva variable, seguida a la anterior, tal como vemos en este código:

```
var l = document.getElementById('brandlogo');
```

En la variable **l**, capturamos el ID correspondiente a la imagen utilizada en el elemento **navbar**. Lo que hacemos a continuación es, en el evento **onOpenStart**, ocultar la imagen en cuestión y, en el evento **onCloseEnd**, volver a visualizarla. Esto se lleva a cabo modificando los parámetros declarados dentro de la variable homónima, tal como lo muestra el siguiente código:

```
onOpenStart: function() { l.style.visibility = 'hidden'; },
onCloseEnd: function() { l.style.visibility = 'visible'; }
```

Modificamos el estilo **visibility** alternando su valor booleano, para mostrar u ocultar el logo en el momento en que se abre o cierra el menú lateral.

Nos queda ahora declarar otra variable en la cual capturaremos la clase del componente CSS **sidenav**. Para esto, escribimos la siguiente línea de código:

```
var componente = document.querySelectorAll('.sidenav');
```

Finalmente, resta inicializar el componente CSS que dará vida a la barra de navegación lateral. Para hacerlo, agregamos la siguiente línea de código:

```
var m = M.Sidenav.init(componente, parametros);
```

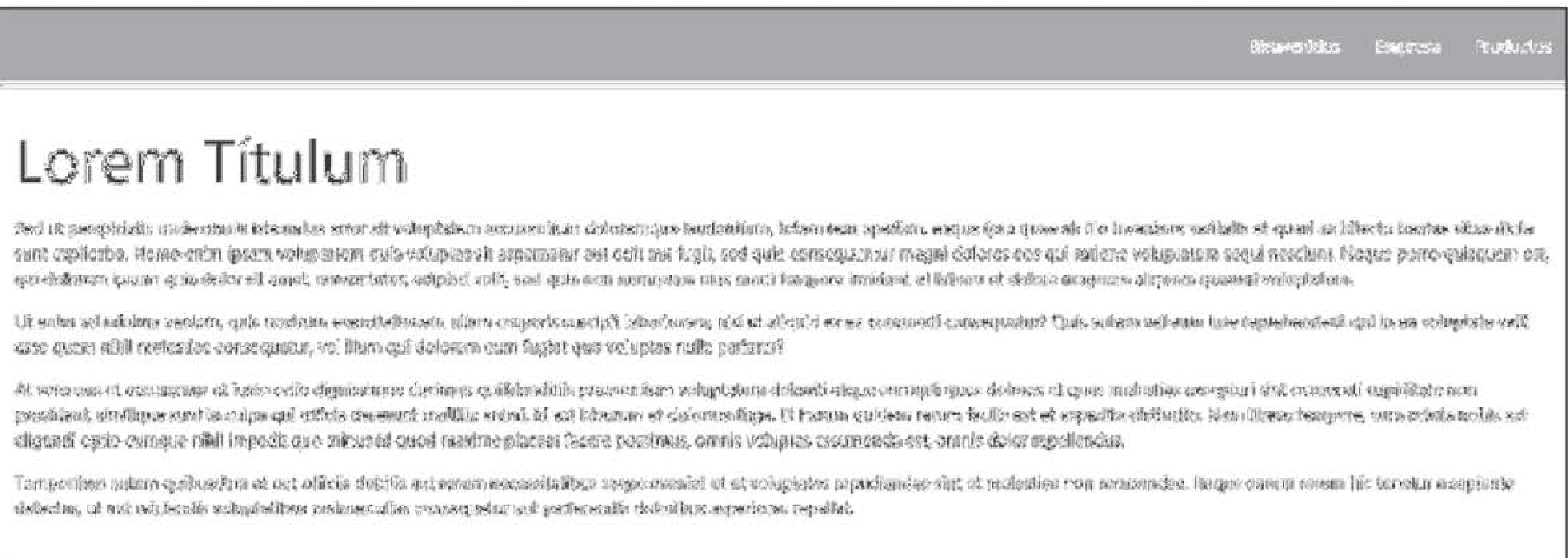
En esta última línea, invocamos a la función **init()**, propia de la clase CSS **sidenav**, y le parametrizamos la variable **componente** correspondiente a la **clase CSS**, y la variable **parametros**.

Todo este código debe quedar estructurado tal como lo muestra la siguiente figura:

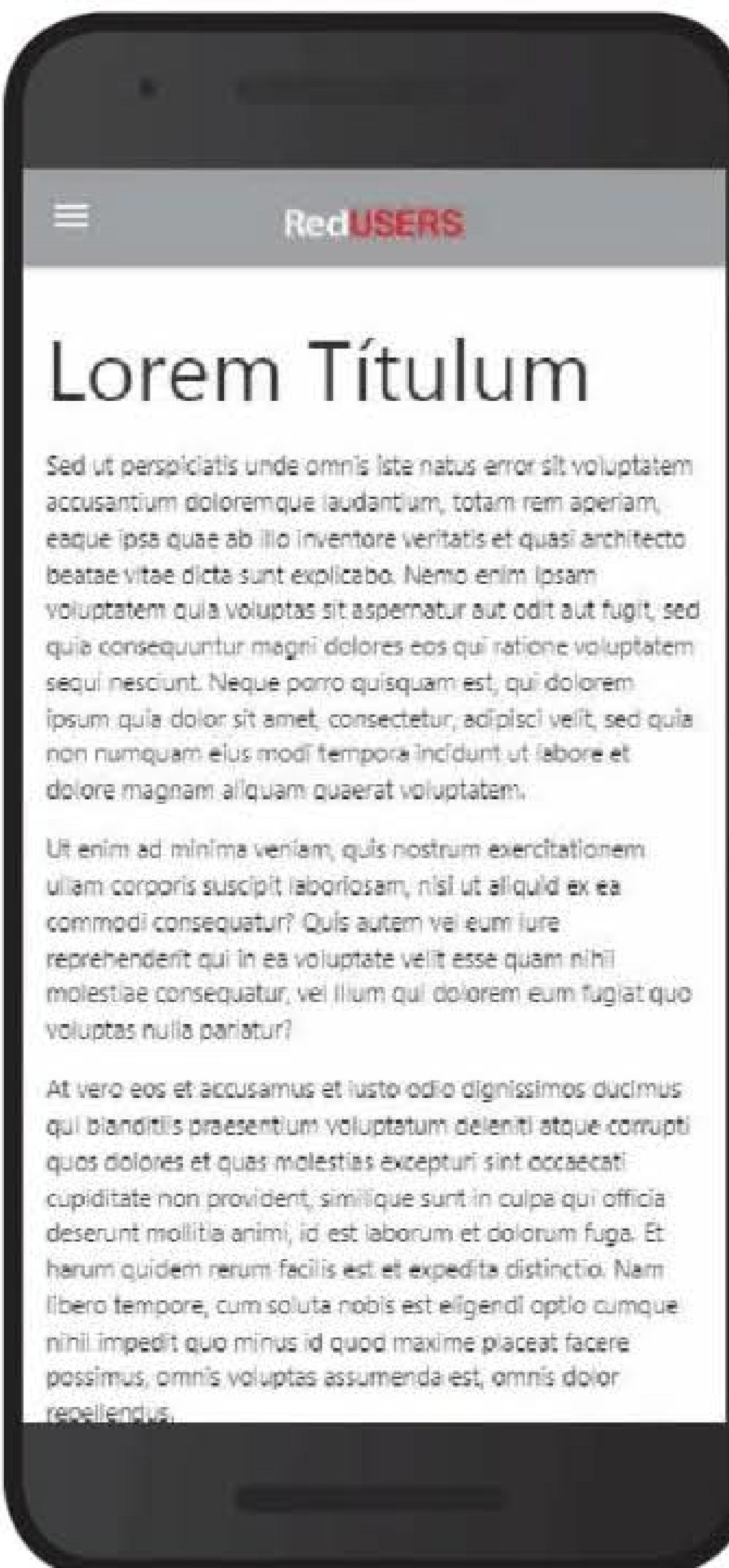
```
document.addEventListener('DOMContentLoaded', function() {
    var parametros = {
        edge: 'left',
        inDuration: 600,
        outDuration: 800,
        preventScrolling: true,
        onOpenStart: function() { l.style.visibility = 'hidden'; },
        onCloseEnd: function() { l.style.visibility = 'visible'; }
    };
    var l = document.getElementById('brandlogo');
    var componente = document.querySelectorAll('.sidenav');

    var m = M.Sidenav.init(componente, parametros);
});
```

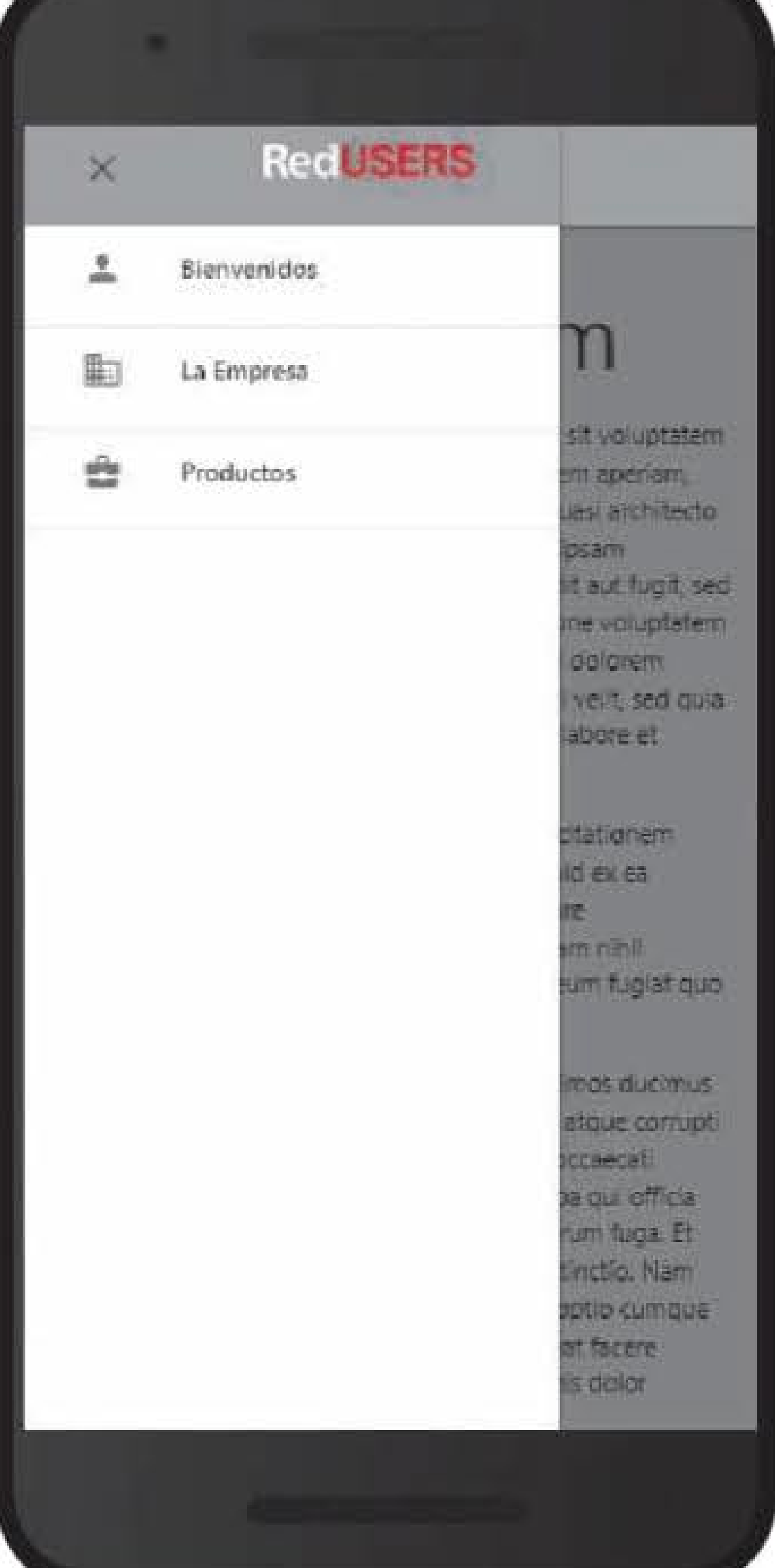
Con esto último ya tenemos inicializada y funcional nuestra barra de navegación lateral. Ahora solo nos queda probar el comportamiento del menú. Para hacerlo, ejecutamos el documento index.html en un navegador web de escritorio. La página web visualizada en modo escritorio deberá mostrar el menú normal, tal como vemos en la siguiente imagen:

96% 23:18



Si pulsamos la tecla **F12** y seleccionamos alguno de los simuladores de smartphones que incluye Google Chrome, como **Nexus 6P**, deberemos visualizar el icono de menú mobile, el cual, al pulsarlo, desplegará el menú móvil acorde a lo que diseñamos en este ejercicio.





Cabe recordar que los lectores tienen a su disposición el código completo de los ejercicios realizados en este e-book. En este caso, se encuentra bajo el nombre **Sidebar_Resuelto.zip**, dentro de los archivos que acompañan a esta obra.

También hay un video en el que testeamos el correcto comportamiento de nuestro menú de navegación lateral, tanto en el simulador de escritorio de Chrome, como en los emuladores iOS y Android: https://youtu.be/-IF9OIv13Sw.

96% 23:18

Manejo de botones

Materialize incluye una serie de botones que nos permiten integrarlos en todo tipo de diseños y desarrollos web. Veamos a continuación cómo sacar el máximo provecho de ellos, y de sus diferentes clases y estilos, completamente adaptables a cualquier necesidad.

Raised Buttons

Los botones tipo **raised** (con relieve, en inglés) son los clásicos botones que encontramos en la mayoría de las aplicaciones que incluyen **Material Design** como estilo gráfico.

Ejemplo funcional del componente Buttons

Descargamos el ejemplo **Raised and Flat buttons.zip** incluido en los archivos que acompañan a esta obra e iniciamos un Nuevo proyecto con él en Visual Studio Code. Dentro del documento **index.html** buscamos la siguiente línea de código:

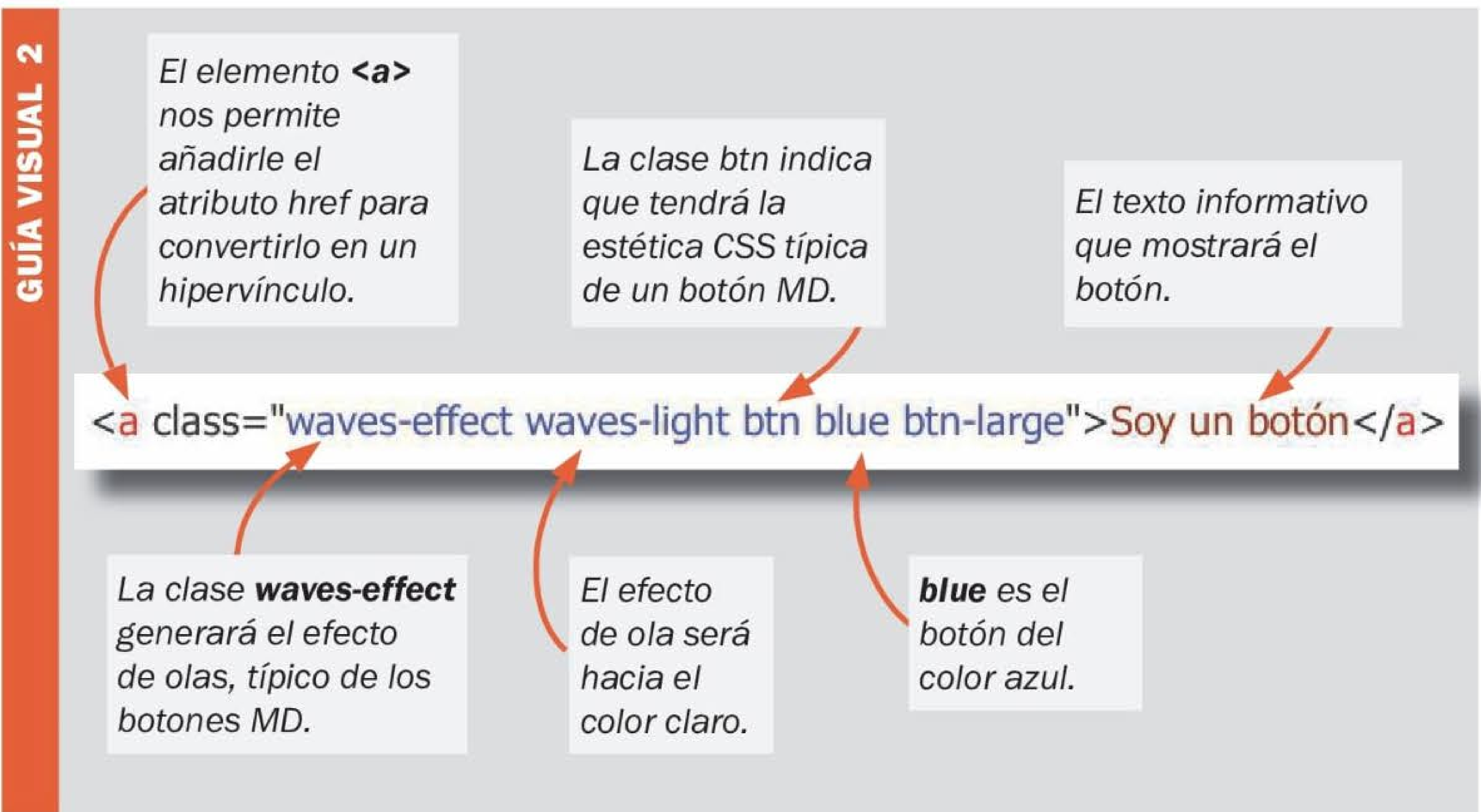
```
<!-- Inserte aquí el componente HTML button -->
```

Y la reemplazamos por esta otra:

```
<a class=”waves-effect waves-light btn blue”>Soy un botón</a>
```

Si ejecutamos el proyecto en el navegador web o emulador, obtendremos como resultado una web similar a la de la imagen.

Como podemos ver al final del documento HTML, conseguimos agregar un botón de color azul. Si pulsamos sobre él, no tendrá interacción; solo hará un efecto propio de los componentes button de Material Design. Analicemos a continuación el código agregado en la **Guía Visual 2**.



Iconos en botones

Si queremos que el botón contenga un icono, dentro del tag **<a>** y **** anidamos el siguiente código:

```
<i class="material-icons left">person</i>
```

De esta forma, veremos este elemento ubicado, en este caso, a la izquierda del texto del botón. Si queremos cambiar el lugar del icono, simplemente agregamos la clase **right** (derecha) en vez de **left**.

En la imagen anterior podemos ver la alineación derecha e izquierda del icono dentro del botón, utilizando por supuesto **right** o **left** como atributo adicional a la clase **material-icons**.

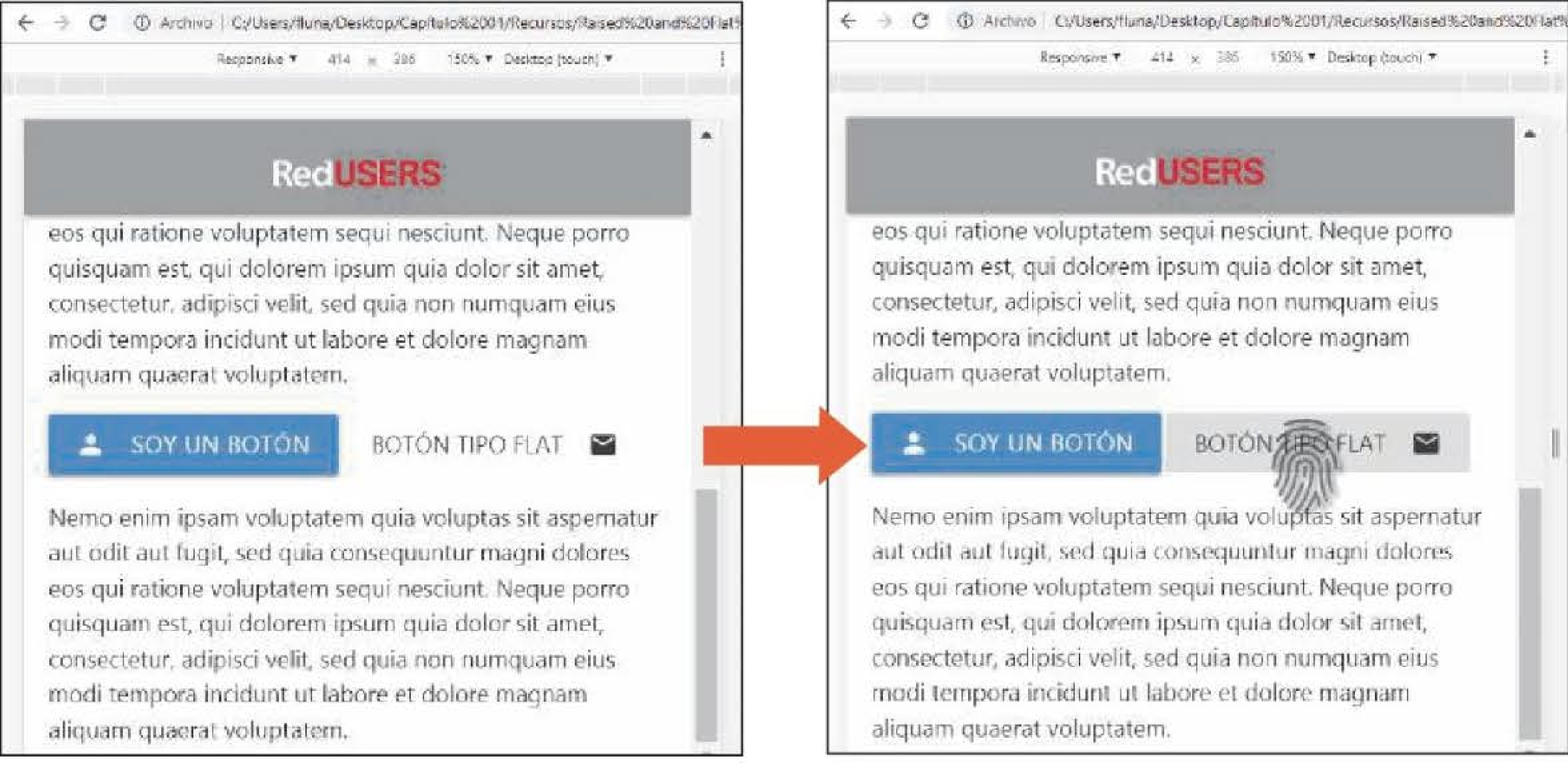
Flat Buttons

Los botones tipo **flat** (planos, en inglés) son aquellos que no contienen relieve y que prácticamente se encuentran integrados en el diseño de un sitio web. Apenas conseguirán desplegar algún efecto gráfico asociado en su clase cuando sean pulsados.

Editemos el ejemplo anteriormente configurado en VS Code, y agreguemos la siguiente línea de código a continuación del componente **button**:

```
<a class=”waves-effect waves-grey btn-flat”><i
class=”material-icons right”>email</i>Botón tipo Flat</a>
```

La clase **btn-flat** le da un aspecto plano al botón, donde apenas vemos resaltado el título y el icono correspondiente. Dejamos los efectos de ola al pulsarlo, cambiando la clase **waves-light** por **waves-grey**, para poder distinguirlo cuando lo presionemos.



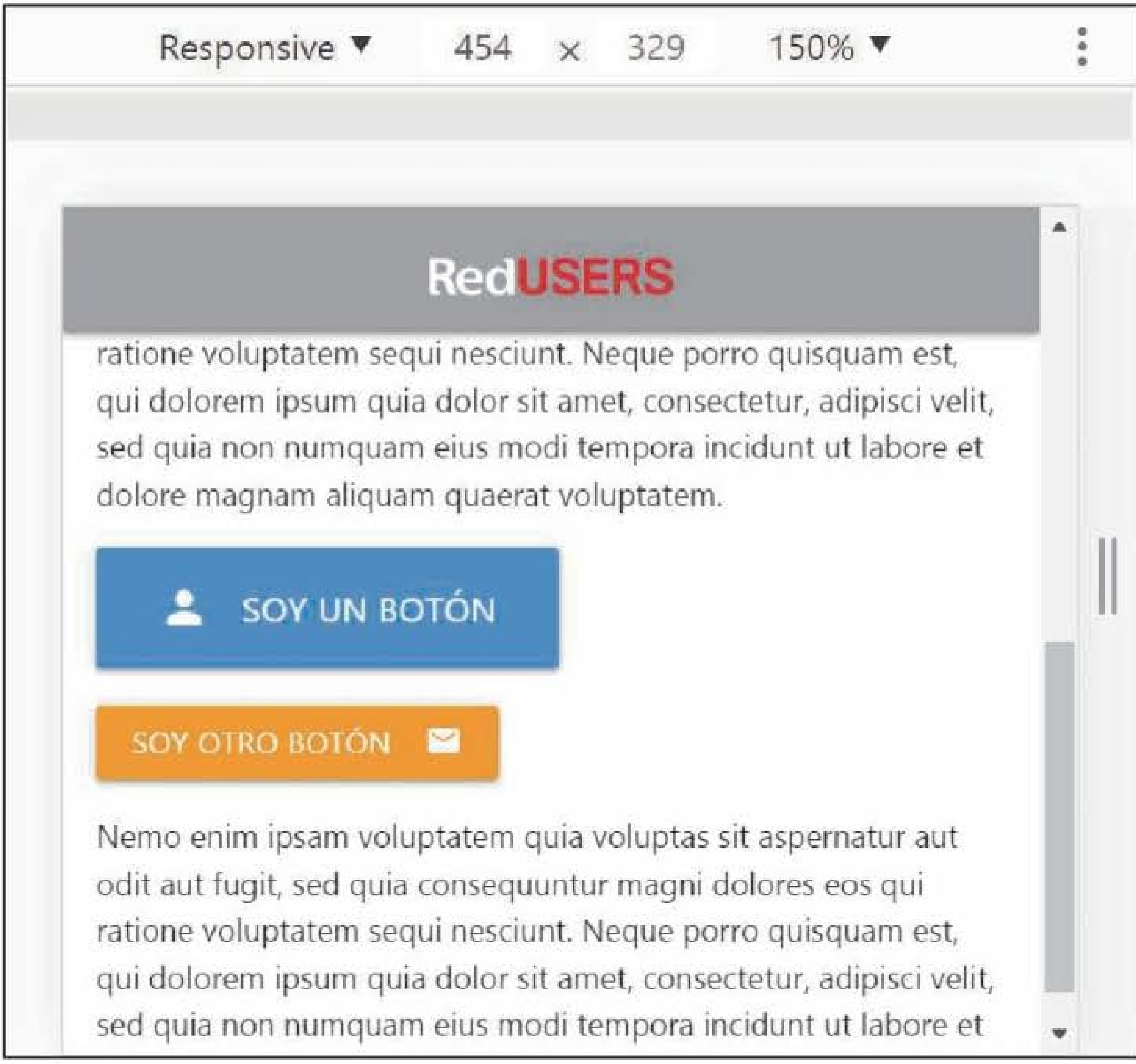
El resultado de la prueba sobre este último botón agregado debe ser similar al que se representa en la imagen anterior.

Cambiar el tamaño de los botones

En la clase **btn**, tenemos la posibilidad de establecer diferentes tamaños para los botones. Por defecto, el tamaño de botón que vimos hasta aquí es el estándar o medio. Materialize nos ofrece dos clases adicionales; **btn-large** y **btn-small**, para hacer más grande o más pequeño el botón. Su implementación es muy simple:

```
<a class="waves-effect waves-light btn blue btn-large"><i
class="material-icons left">person</i>Soy un botón</a>
<a class="waves-effect waves-grey btn orange btn-small"><i
class="material-icons right">email</i>Soy otro Botón</a>
```

En las líneas de código anterior, podemos ver cómo aplicamos las clases **btn-small** y **btn-large**, una en cada botón, para así dimensionar de manera diferente el tamaño que tendrán. El resultado se muestra en la siguiente imagen:



Deshabilitar el botón

Materialize CSS también nos permite manejar el estado del botón a través de la clase **disabled**. Cuando creamos un botón con Materialize, este aparece habilitado por defecto, pero si incluimos la clase CSS disabled en el atributo **class**, se creará por defecto desactivado.

Para manejar el estado del botón sobre la base de una determinada condición (por ejemplo, que se habilite solo cuando todos los campos de un formulario estén completos), deberemos manipular la clase **disabled** desde JavaScript. Para ver cómo hacerlo, descarguemos el ejemplo llamado **Desactivar botones.zip** del repositorio de archivos de esta obra. Lo descomprimimos y creamos un nuevo proyecto en VS Code.

Manejo de la clase disabled desde JavaScript

Nuestro proyecto consta de una simple página HTML con un botón y dos hipervínculos: **Desactivar botón** y **Activar botón**. Ubicamos dentro del documento HTML el script que apunta al archivo JS llamado **estadobotones.js**. Pulsamos sobre el archivo y, luego, aceptamos su creación. A continuación, agregamos el

96% 23:18

control de la carga del DOM, que utilizamos en ejemplos anteriores de esta colección de e-books.

```
document.addEventListener("DOMContentLoaded", function() {

})
```

Dentro de la sentencia anterior, declaramos dos variables las cuales se ocuparán de controlar el evento clic, o tap, de los hipervínculos incluidos en el documento HTML:

```
var b1 = document.getElementById("desactivarbtn");
var b2 = document.getElementById("activarbtn");
```

Seguido a esto, creamos una variable la cual se ocupará de controlar el botón principal, que activaremos y desactivaremos:

```
var sb = documento.getElementById("soyunboton");
```

A continuación, creamos un **Event Listener** para cuando se pulse el hipervínculo **Desactivar botón**:

```
b1.addEventListener("click", function() {

})
```

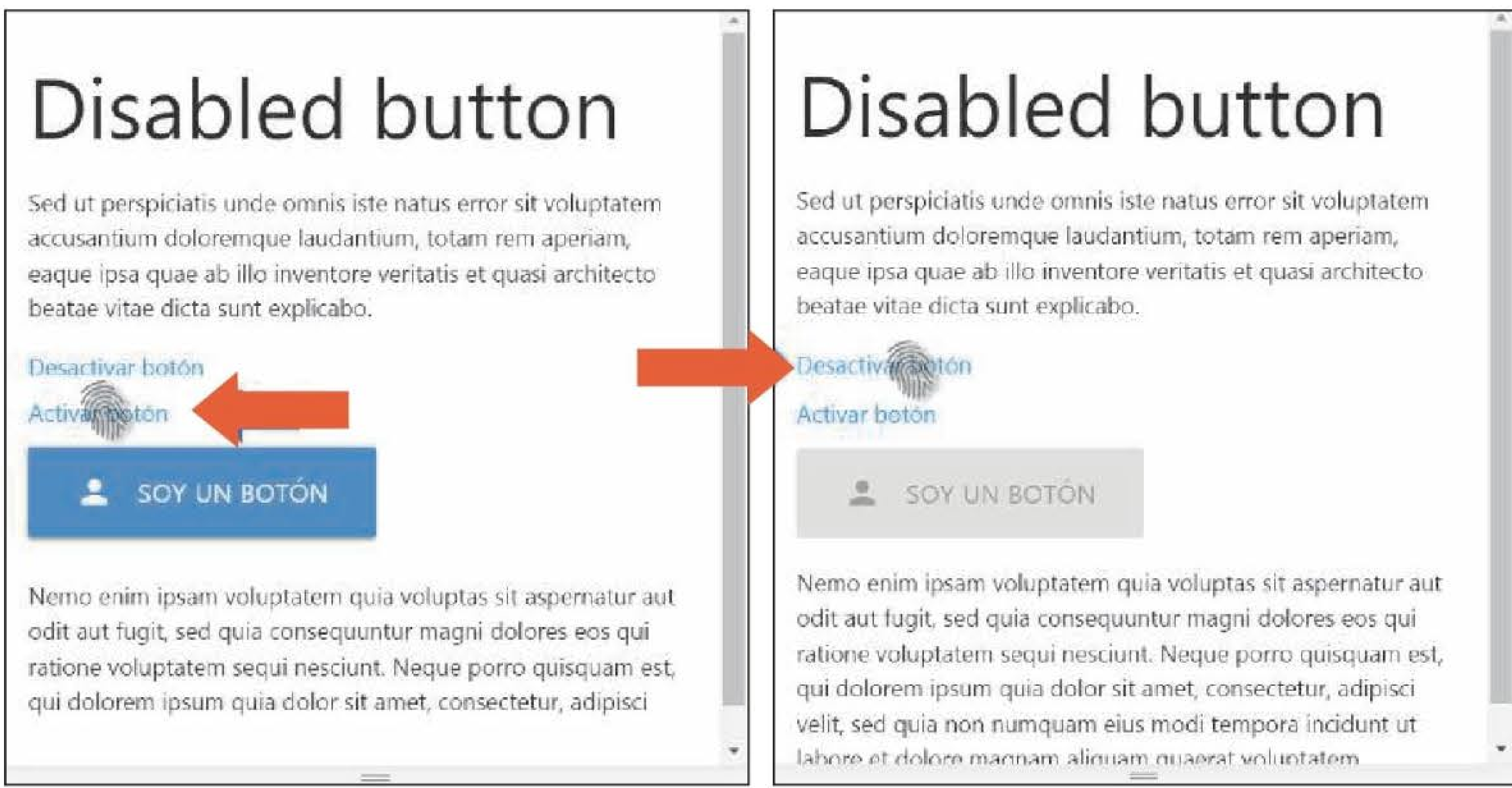
Y dentro de este, escribimos el siguiente código:

```
sb.classList.add("disabled");
```

Repetimos este último paso para darle vida al hipervínculo **Activar botón**, cuyo control lo realizaremos a través de la variable **b2**; en vez de usar el método **add** de la propiedad **classList**, utilizaremos **remove**. El resultado de esta sentencia debe quedar como en el siguiente bloque de código:

```
b2.addEventListener("click", function() {
  sb.classList.remove("disabled");
})
```

Como resultado, cuando se ejecuta el evento **add()** de la propiedad **classList**, el botón queda deshabilitado por completo, tanto visualmente como también cualquier evento que tenga asociado, como un hipervínculo en su atributo **href**, o mediante el Event Listener que controla el evento **click** desde JavaScript:



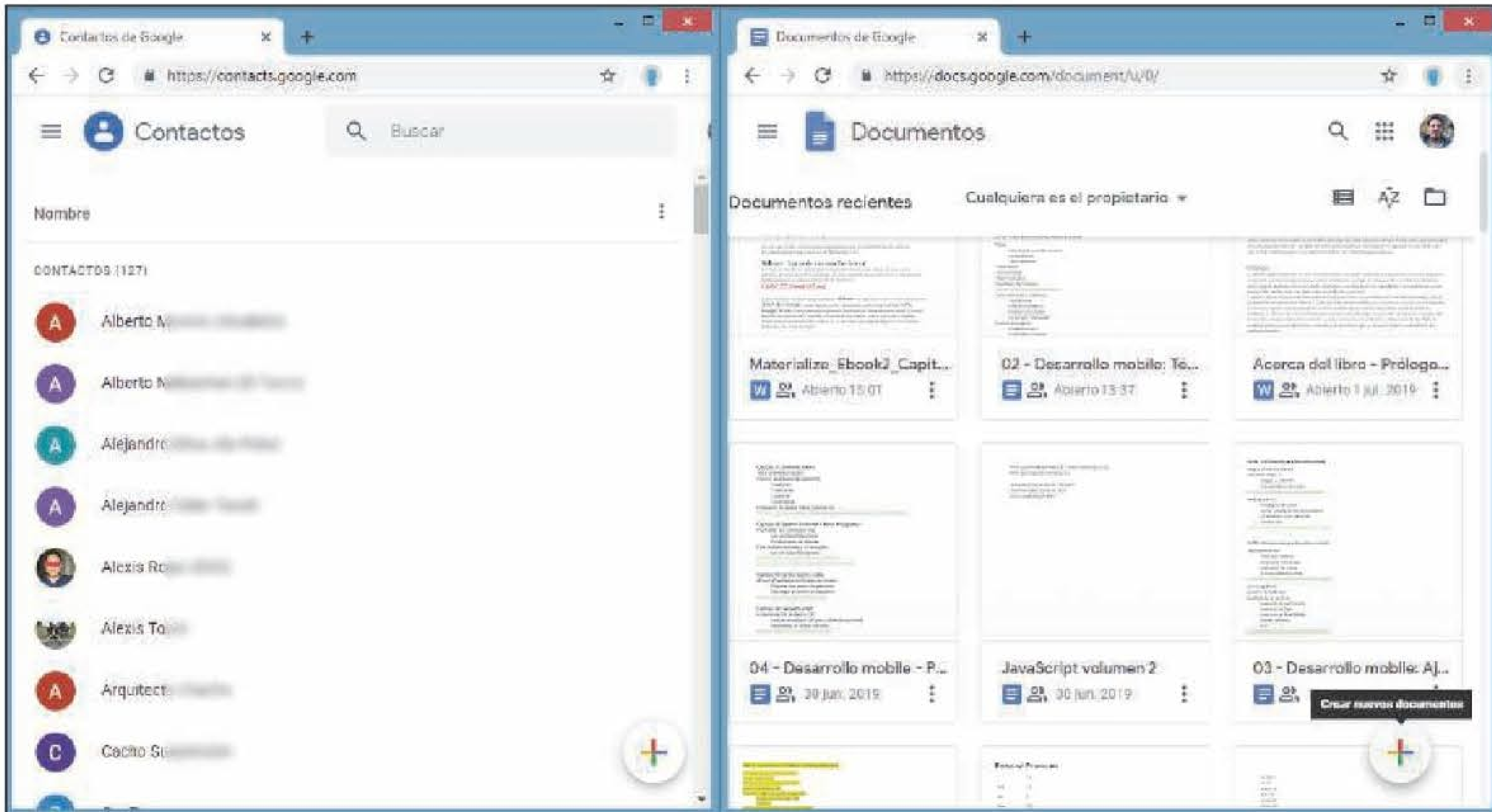
Por el contrario, el evento **remove()** de la propiedad **classList** se ocupa de eliminar la clase o clases que se le indican como parámetro. Así, el botón en cuestión volverá a quedar habilitado visual y funcionalmente.

De esta manera, combinando las diferentes clases que ofrece Materialize CSS para los elementos **Button** personalizados, podremos controlar las funcionalidades básicas de estos utilizando tan solo muy pocas líneas de código JS. Ante cualquier duda, es posible descargar el archivo **Desactivar botones Resuelto.zip** para observar el código completo y funcional.

Floating button

Materialize CSS incluye también la posibilidad de crear botones del tipo **floating**, clásicos en el entorno Material Design, Webs de Google y Android. Estos botones permiten simplificar el acceso a la función más popular que puede contener un sitio web, web mobile o aplicación móvil, como vemos en la imagen de la página siguiente.

Su implementación es muy sencilla. Debemos crear un botón de Materialize CSS, tal como vimos anteriormente, pero en vez de utilizar la clase CSS **btn**, la reemplazamos por **btn-floating**. Construyamos a continuación un ejemplo para comprender y apreciar el uso de este botón; para hacerlo, descargamos del repositorio de archivos el proyecto **Floating button.zip**, lo descomprimimos y



creamos en Visual Studio Code un nuevo proyecto con dicha carpeta. Editemos a continuación el documento index.html, donde luego del elemento **<p>**, agregamos el siguiente código:

```
<div class="flotar">
  <a class="btn-floating btn-large blue waves-effect waves-
light"><i class="large material-icons">add</i></a>
</div>
```

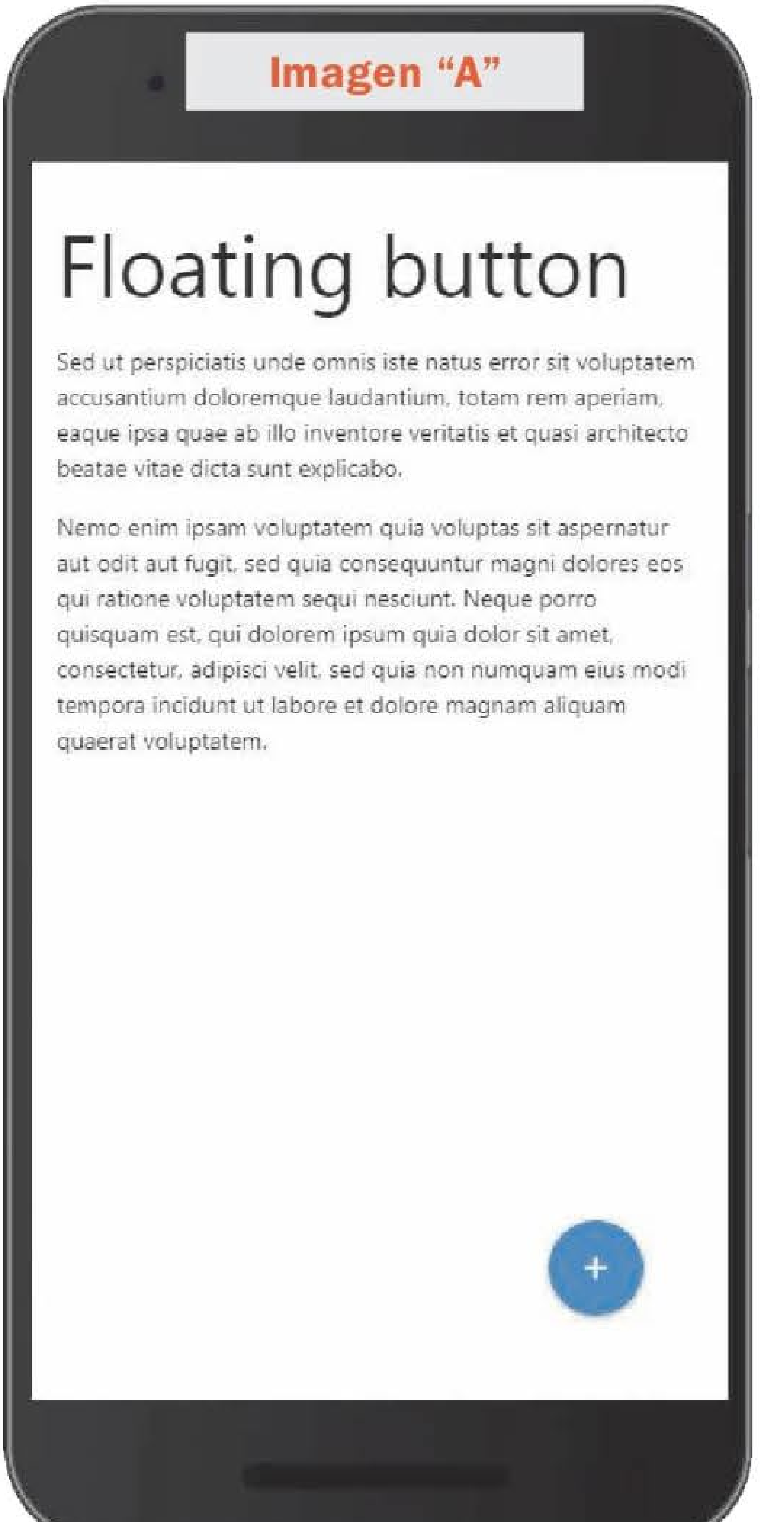
Como podemos apreciar, creamos un elemento **<div>** y, dentro de él, contenemos el botón flotante en cuestión. Ahora nos ocuparemos de alinear dicho botón sobre el extremo inferior derecho. Para hacerlo, utilizaremos CSS creando una clase denominada **flotar**, que es la que incluimos en el código anterior, dentro del atributo **class** de dicho div.

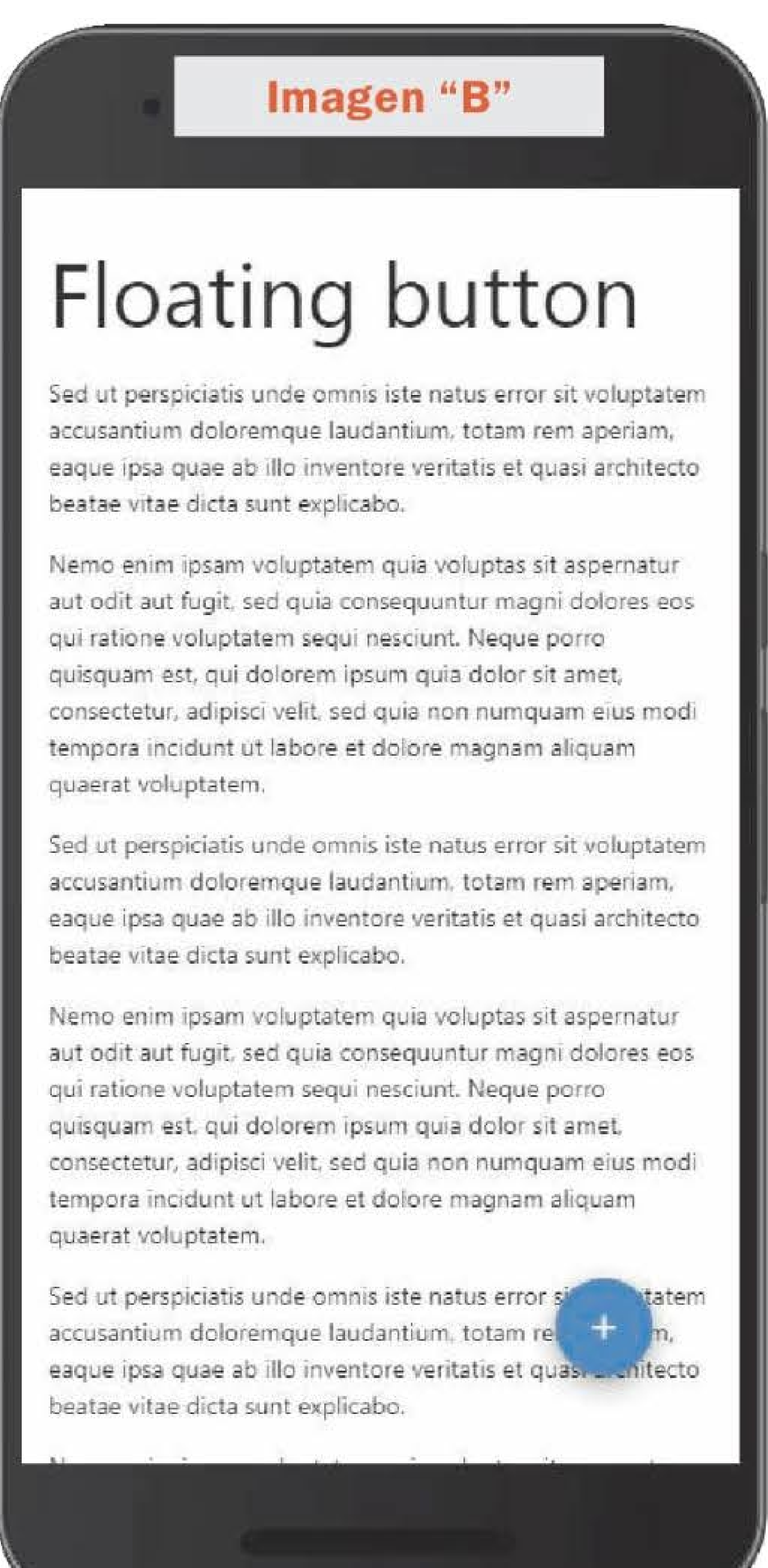
Vamos a editar el archivo **styles.css** que está incluido en este proyecto, agregándole el siguiente bloque de código:

```
.flotar {
  position: absolute;
  bottom:50px;
  right: 50px;
}
```

96% 23:18

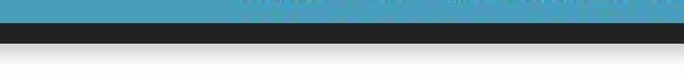
El resultado, en la siguiente imagen ”A”. Si queremos que el botón disponga de un poco más de sombra, dentro del atributo **class** añadimos la propiedad **z-depth-3**. A continuación, nos aseguramos de que el botón flotante quede siempre por encima del contenido del documento HTML. Para probar si efectivamente es así, replicamos al menos cuatro o cinco veces más el contenido del elemento **<p>** que tiene el documento HTML. El resultado debe ser similar al de la siguiente imagen “B”:





Fijar el botón

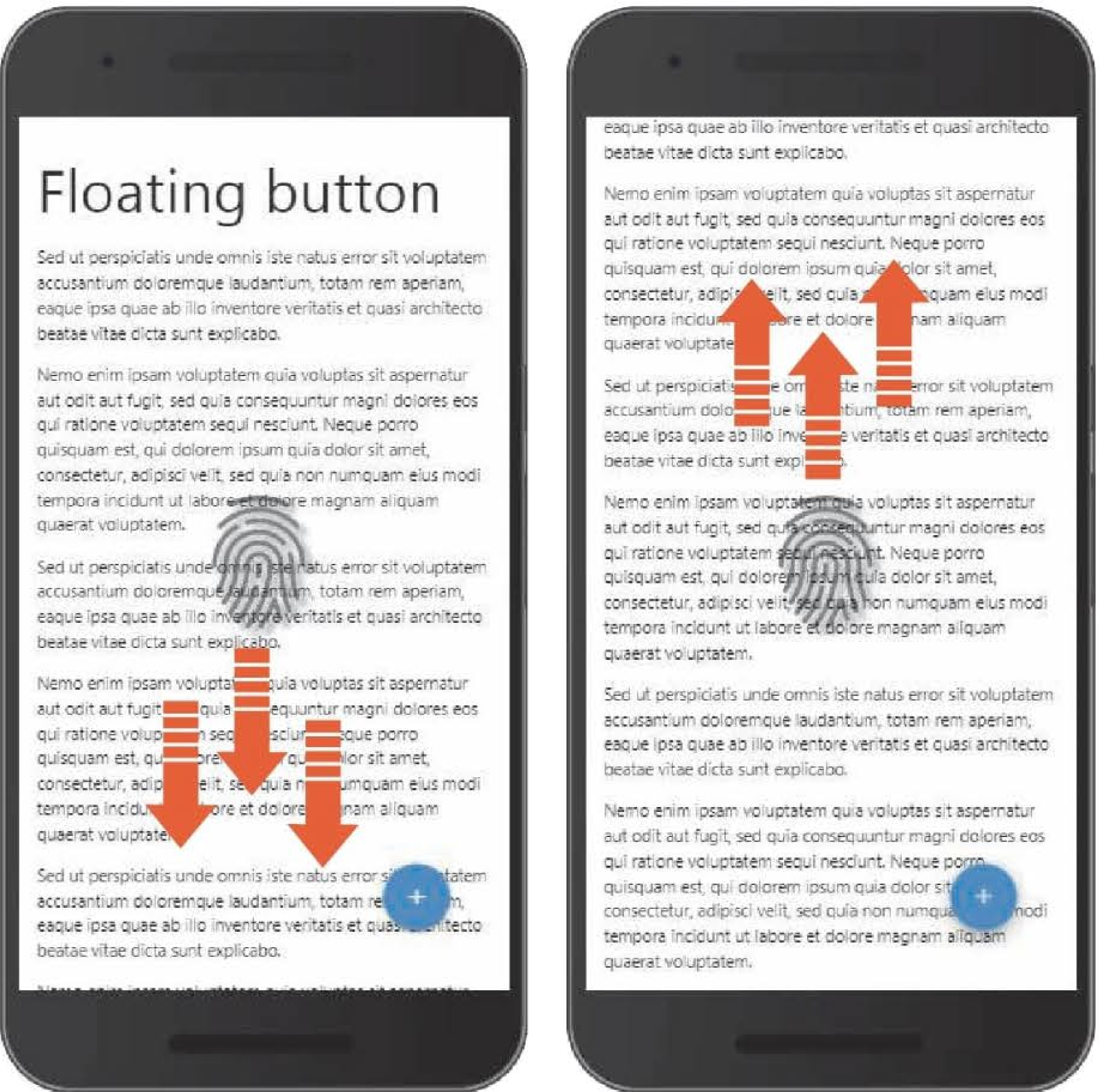
Sobre la base de la configuración CSS aplicada, si el contenido del documento HTML es extenso y genera scroll en la página web, podemos fijar el botón flotante en la ubicación otorgada, cambiando el valor de la propiedad **position** en el archivo CSS, de **absolute** a **fixed**. Eso se ocupará de fijar el div y, con él, el botón flotante.



Finalmente, agregamos un código JavaScript para probar la funcionalidad de nuestro botón flotante. Agregamos el atributo **id = " btnflotante"** en el elemento HTML en cuestión. Luego, editamos el archivo JS relacionado a este documento y escribimos el siguiente código:

```
var bf = document.getElementById("btnflotante");
bf.addEventListener("click", function() {
  alert('Felicitaciones: tu botón flotante funciona correctamen-
te!');
})
```

Probamos en nuestro emulador o a través de **Fenix Webserver** que el mensaje a visualizar funcione de manera correcta.

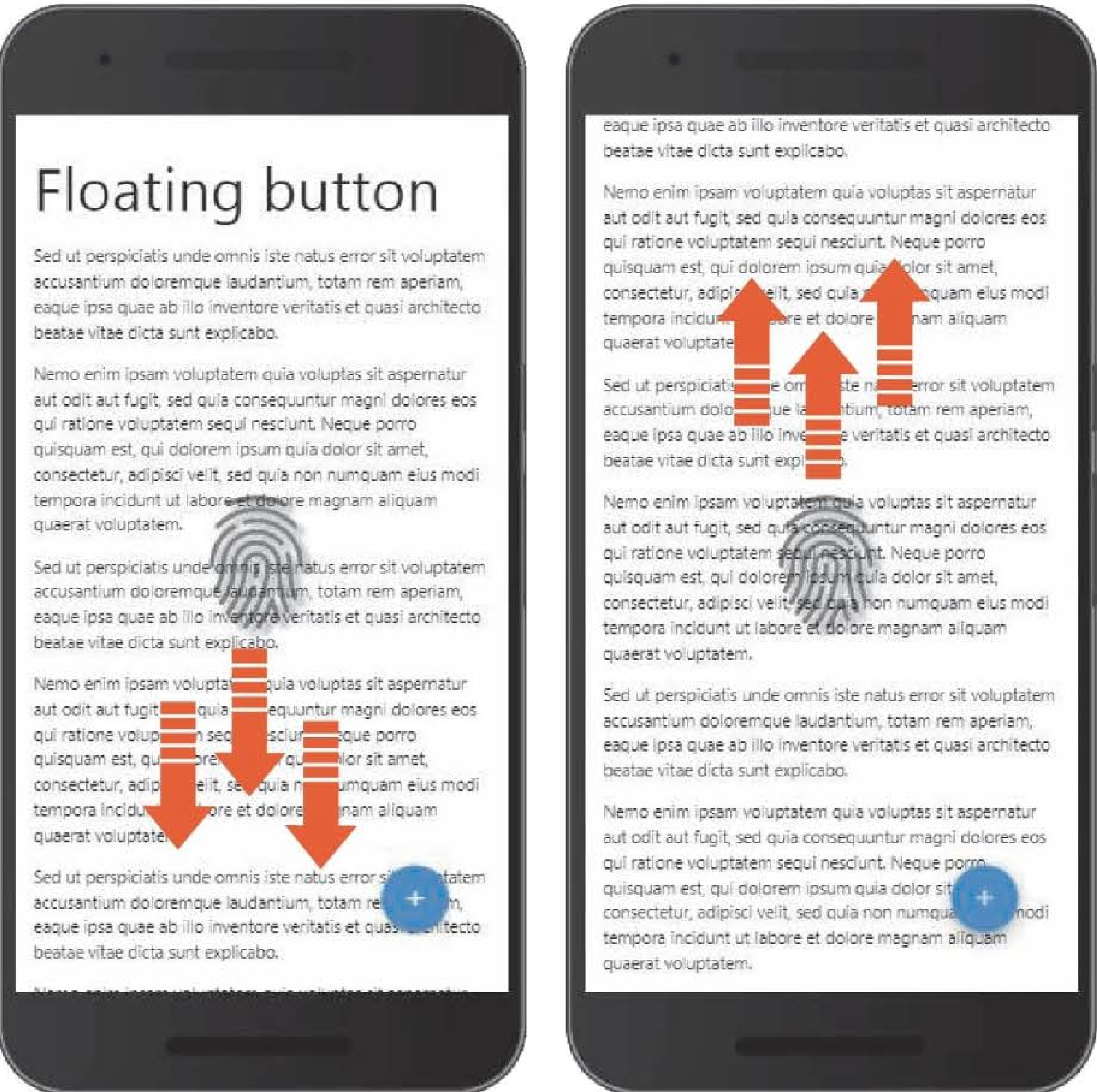


Para terminar, cuando probamos el resultado de este ejercicio, debemos poder desplazarnos por todo el documento HTML, y este debe realizar el correspondiente scroll sin alterar la posición indicada para **Floating button**.

Finalmente, agregamos un código JavaScript para probar la funcionalidad de nuestro botón flotante. Agregamos el atributo **id = " btnflotante"** en el elemento HTML en cuestión. Luego, editamos el archivo JS relacionado a este documento y escribimos el siguiente código:

```
var bf = document.getElementById("btnflotante");
bf.addEventListener("click", function() {
  alert('Felicitaciones: tu botón flotante funciona correctamen-
te!');
})
```

Probamos en nuestro emulador o a través de **Fenix Webserver** que el mensaje a visualizar funcione de manera correcta.



Para terminar, cuando probamos el resultado de este ejercicio, debemos poder desplazarnos por todo el documento HTML, y este debe realizar el correspondiente scroll sin alterar la posición indicada para **Floating button**.

96% 23:22

02 Tablas y contenedores

Veremos a continuación cómo sacarles provecho a dos elementos fundamentales para estructurar la información de manera clara y eficaz: las tablas y los contenedores que propone Materialize CSS.

Tablas

Como desarrolladores de software, debemos tener presente que las tablas son un elemento fundamental en el momento de presentar información y datos de modo estructurado. Gracias a Materialize CSS, contamos con diversos elementos que, a través de una serie de clases, nos permitirán controlar práctica y fácilmente cualquier cúmulo de datos que debamos presentar en una solución web.

La etiqueta Table

Materialize CSS hace uso de la etiqueta **<table>** y metatags asociados para estructurar los datos en pantalla. Vemos en la **Tabla 3** cuáles son todos sus metatags y para qué se utiliza cada uno.

TABLA 3

Etiqueta	Descripción
<table></table>	Indica la creación de un contenido dentro de una tabla. Cada ítem que se detallará en esta tabla estará contenido dentro de estas etiquetas.
<thead></thead>	**Table head:** indica el encabezado de la tabla, donde se detallará el nombre de cada campo.
<tr></tr>	**Table row:** define que se inicia una fila de datos dentro de la tabla, por supuesto, horizontal. Se utiliza dentro de **<thead>** y también dentro de **<tbody>**.
<th></th>	**Title head:** define el título de cada columna de datos que tendrá la tabla.
<tbody></tbody>	**Table body:** delimita el inicio de la sección de la tabla que muestra los datos. Dentro de este metatag, definimos nuevamente un **<tr>** por cada fila de datos que agregaremos y, dentro de estos, definiremos cada conjunto de celdas **<td>**.
<td></td>	**Table definition:** es una celda que contendrá los datos por mostrar. Debemos crear un **<td>** por cada dato que contendrá la fila.

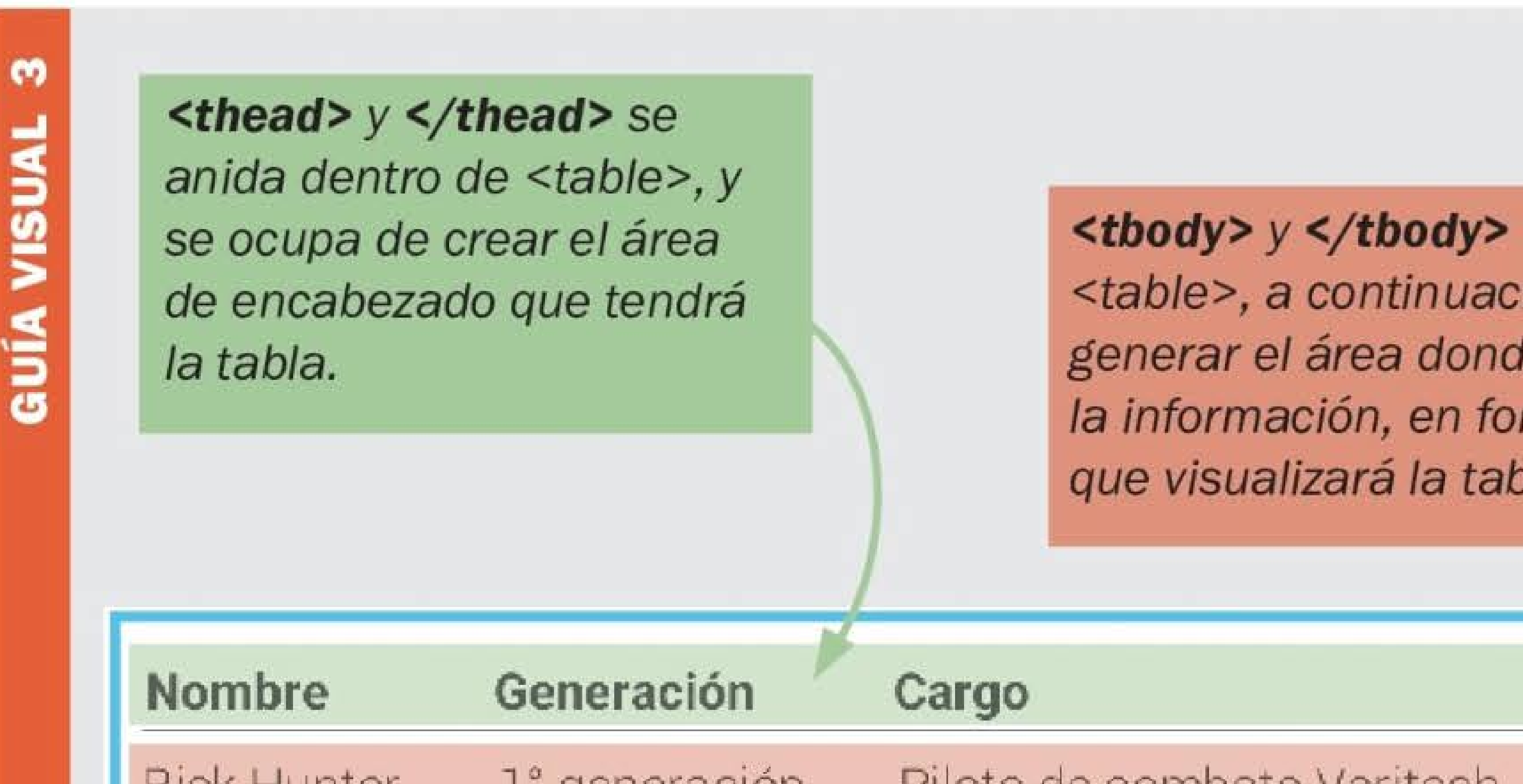
Veamos a continuación una figura que representa una tabla de datos armada con Materialize CSS.

Como podemos apreciar en la imagen, la estructura de una tabla creada con Materialize CSS es simple y, como separador, solo integra una línea horizontal por cada fila de datos.

Name	Item Name	Item Price
Alvin	Eclair	$0.87
Alan	Jellybean	$3.76
Jonathan	Lollipop	$7.00
Shannon	KitKat	$9.99

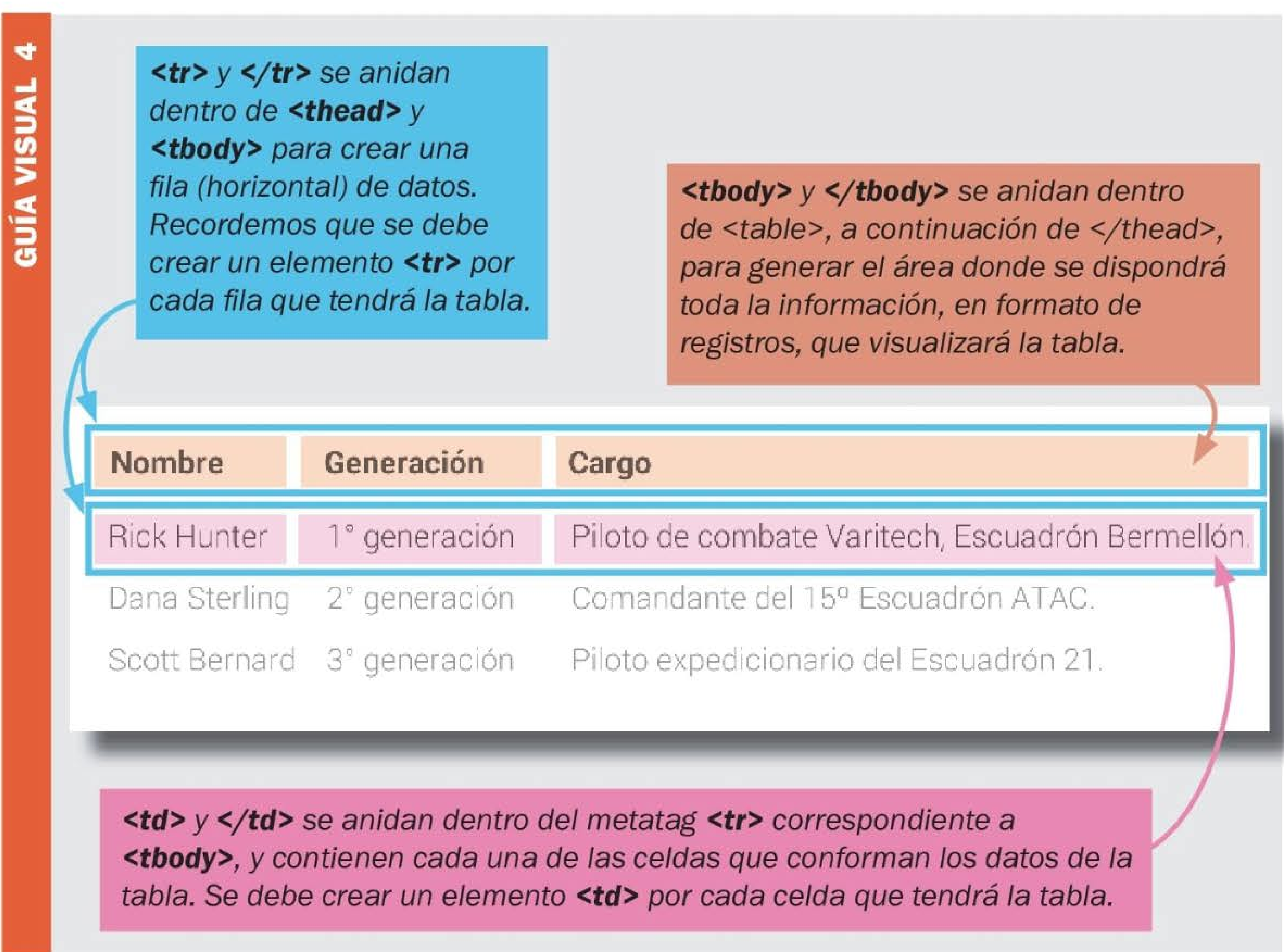
Estructura

Exploremos en las siguientes **Guías Visuales 3** y **4**, dónde y cómo se ubica cada uno de los metatags explicados anteriormente que componen una tabla.



Nombre	Generación	Cargo
Rick Hunter	1° generación	Piloto de combate Varitech, Escuadrón Bermellón.
Dana Sterling	2° generación	Comandante del 15° Escuadrón ATAC.
Scott Bernard	3° generación	Piloto expedicionario del Escuadrón 21.

Y, en referencia a la estructura interna de los elementos de la tabla, tenemos:



Clases de la etiqueta Table

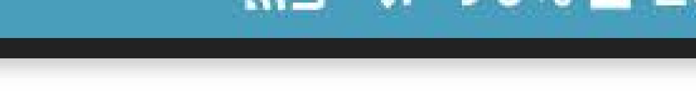
Materialize CSS ofrece una serie de clases predeterminadas que nos permitirán ver de manera más clara y organizada el contenido que necesitemos cargar en una o más tablas. Son cuatro clases simples, que, a su vez, se pueden combinar entre sí para mejorar aún más la visualización del contenido.

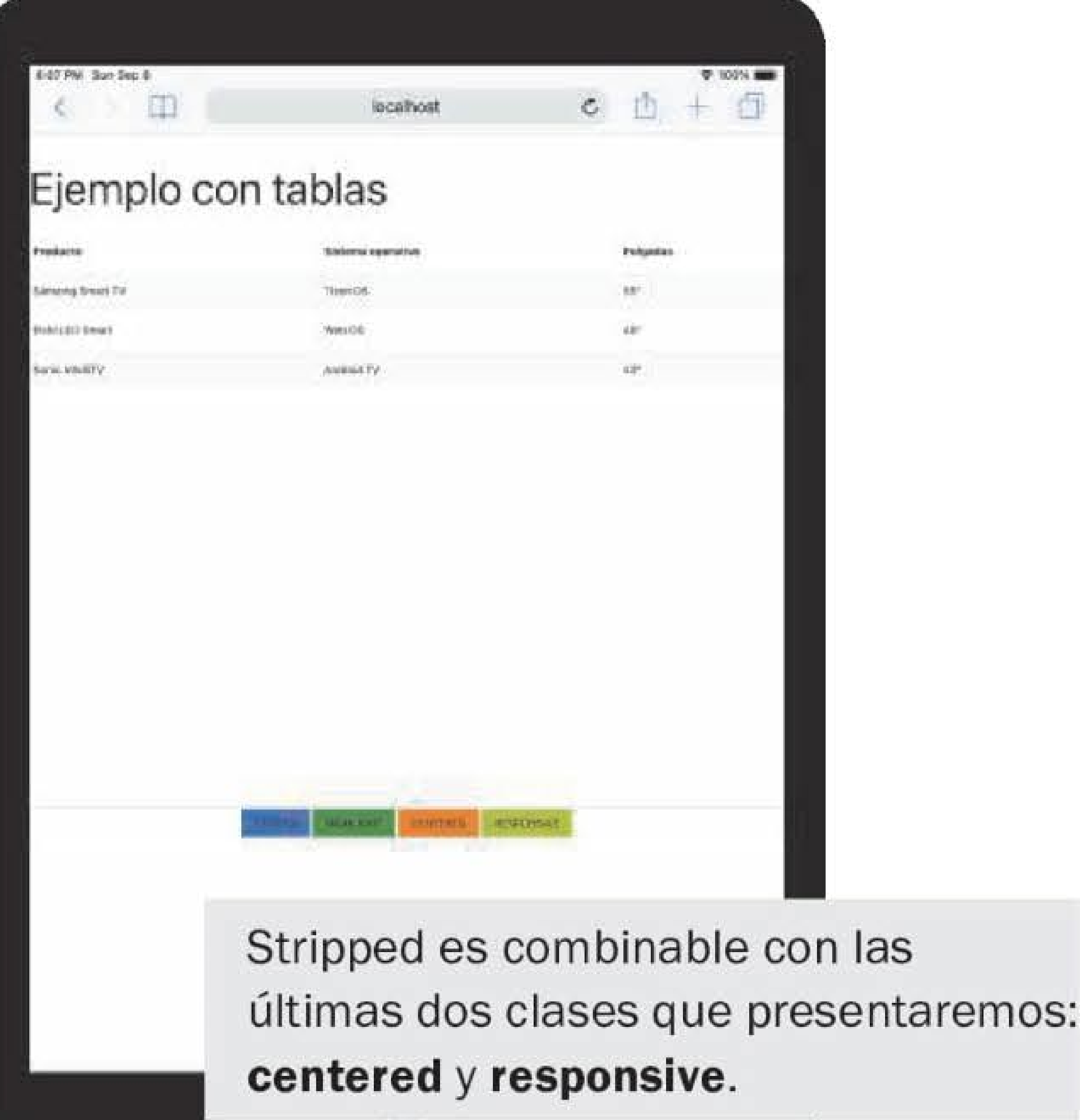
En las figuras que presentamos en la página siguiente, veremos el uso de las diferentes clases, cada una representada a la derecha de la imagen.

Stripped

La clase **stripped** nos permite visualizar el contenido de una tabla en modo plano, resaltando el fondo de cada fila por medio, para poder entregar una mejor lectura cuando haya muchos ítem listados. De una forma clara, podremos mejorar la visualización de su contenido sin nada de esfuerzo.

Stripped es combinable con las últimas dos clases que presentaremos: **centered** y **responsive**.

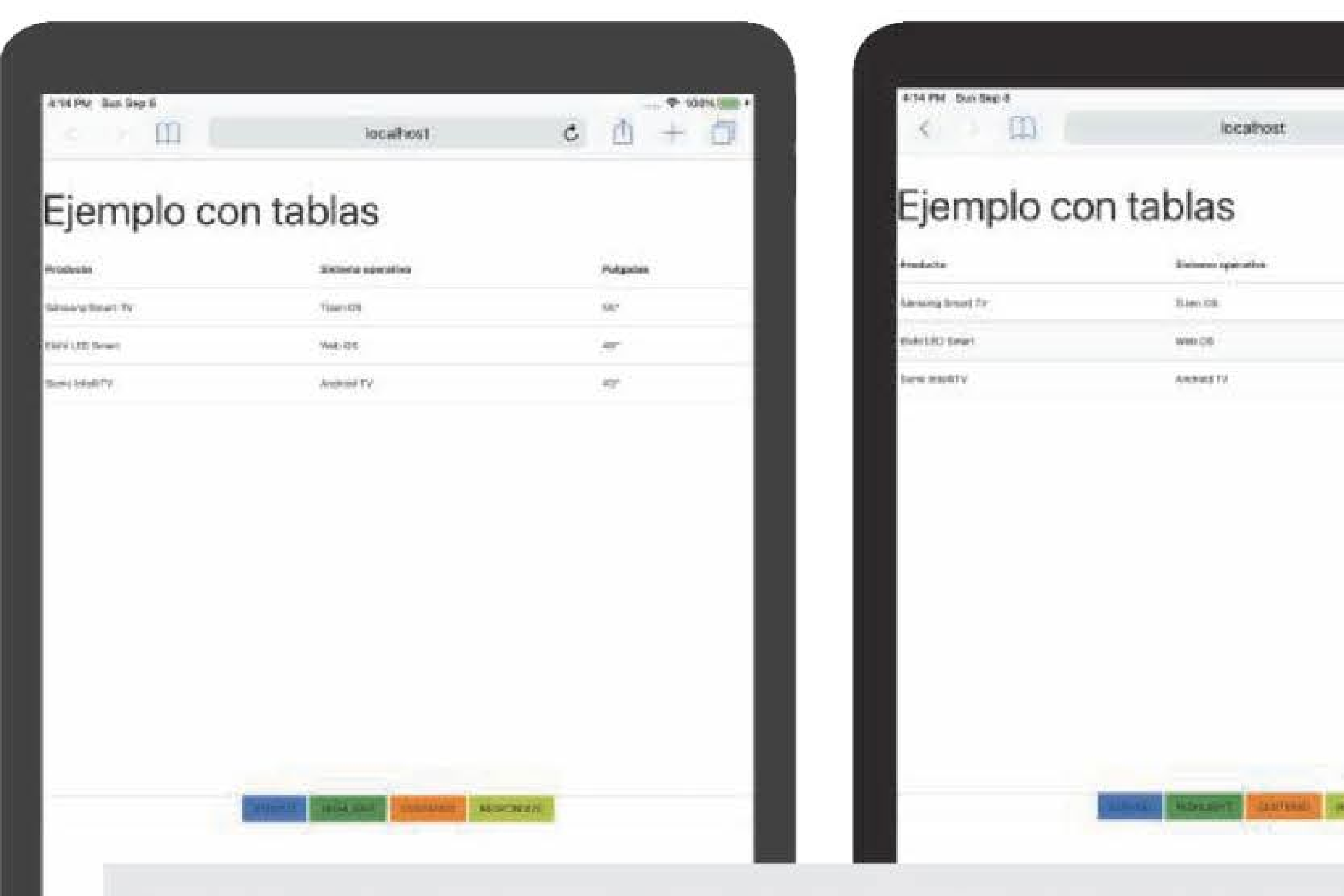




Stripped es combinable con las últimas dos clases que presentaremos: **centered** y **responsive**.

Highlight

Esta clase visualiza la tabla en modo normal o predeterminado, y solo resalta la fila que seleccionamos, ya sea posicionando el mouse sobre una fila (en modo desktop) o haciendo un tap sobre una fila determinada, en modo dispositivo móvil. En cualquiera de estos casos siempre se resaltará solo una de las filas listadas.



Highlight es combinable con las siguientes clases que presentaremos, y si una vez seleccionada una fila queremos volver a su estado anterior, el efecto highlight puede desactivarse simplemente seleccionando otra o haciendo tap sobre el título de la tabla.

96% 23:22

Centered

Como el nombre de esta clase lo indica, se ocupa de alinear todo el contenido en el centro de una celda. La estructura de la tabla que visualiza es la estándar, y podemos combinarla con las clases **Stripped**, **Highlight** y **Responsive**.

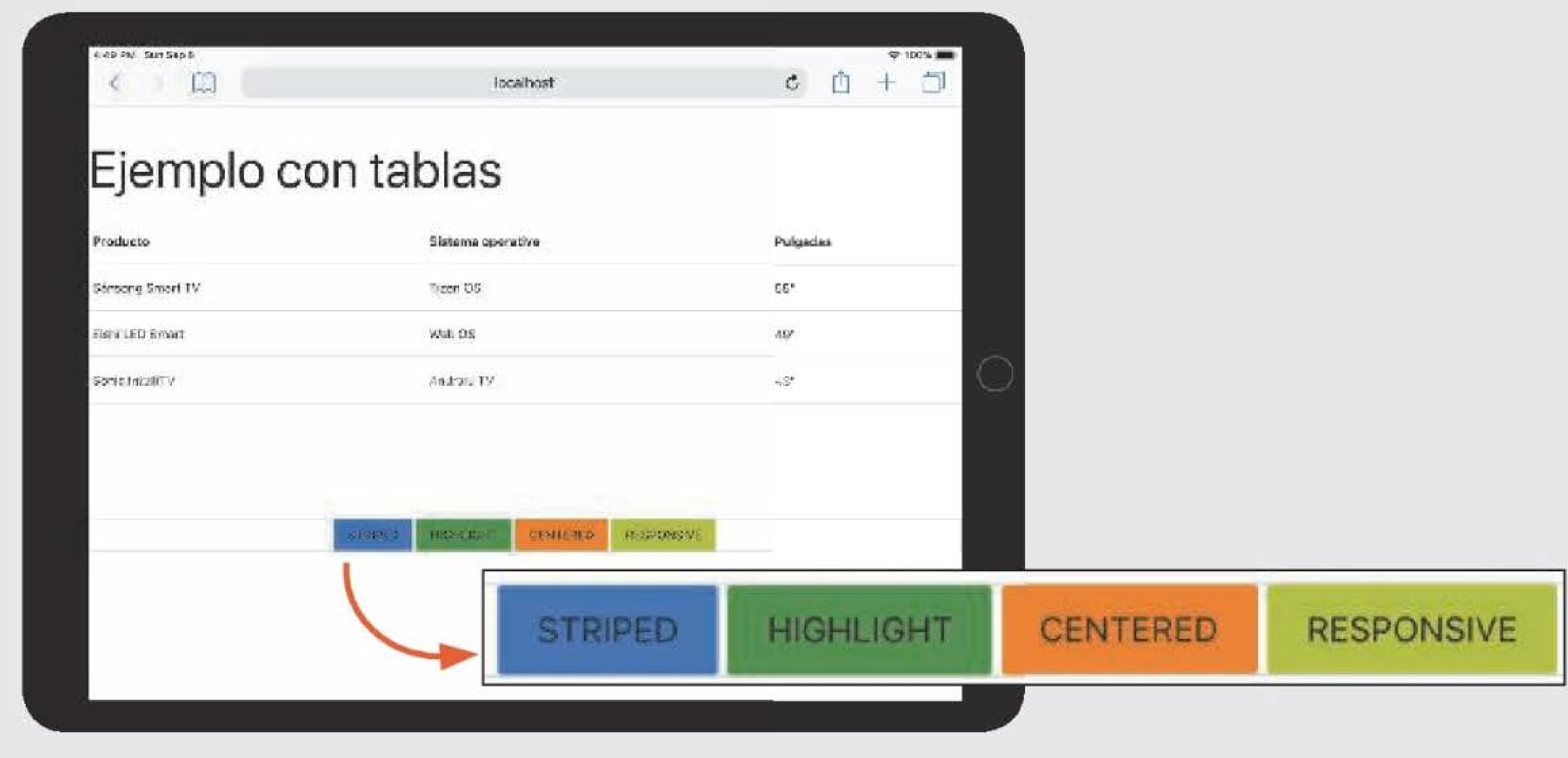
Responsive

Se ocupa de redistribuir todo el contenido de la tabla de manera inteligente, lo cual nos permite mejorar la visualización de sus datos cuando tenemos pantallas de ancho muy limitado y mucho contenido en la tabla.

En la siguiente figura, podemos ver cómo la clase responsive, aplicada sobre el simulador de **iPhone 5S**, cambia la visualización de la tabla respecto a cómo esta se muestra en el dispositivo **iPad**. iPad tiene una gran resolución y puede desplegar cómodamente la tabla, mientras que la pantalla del iPhone 5S limita el ancho y permite que saquemos provecho de la clase responsive, para reestructurar el contenido de la tabla y que no quede superpuesto entre sí cuando se visualiza en pantallas pequeñas. Combinando un pequeño código JS, podremos detectar el ancho de pantalla del dispositivo, y allí definir qué clase aplicar sobre el tag **<table>**.

Ejemplo de prueba

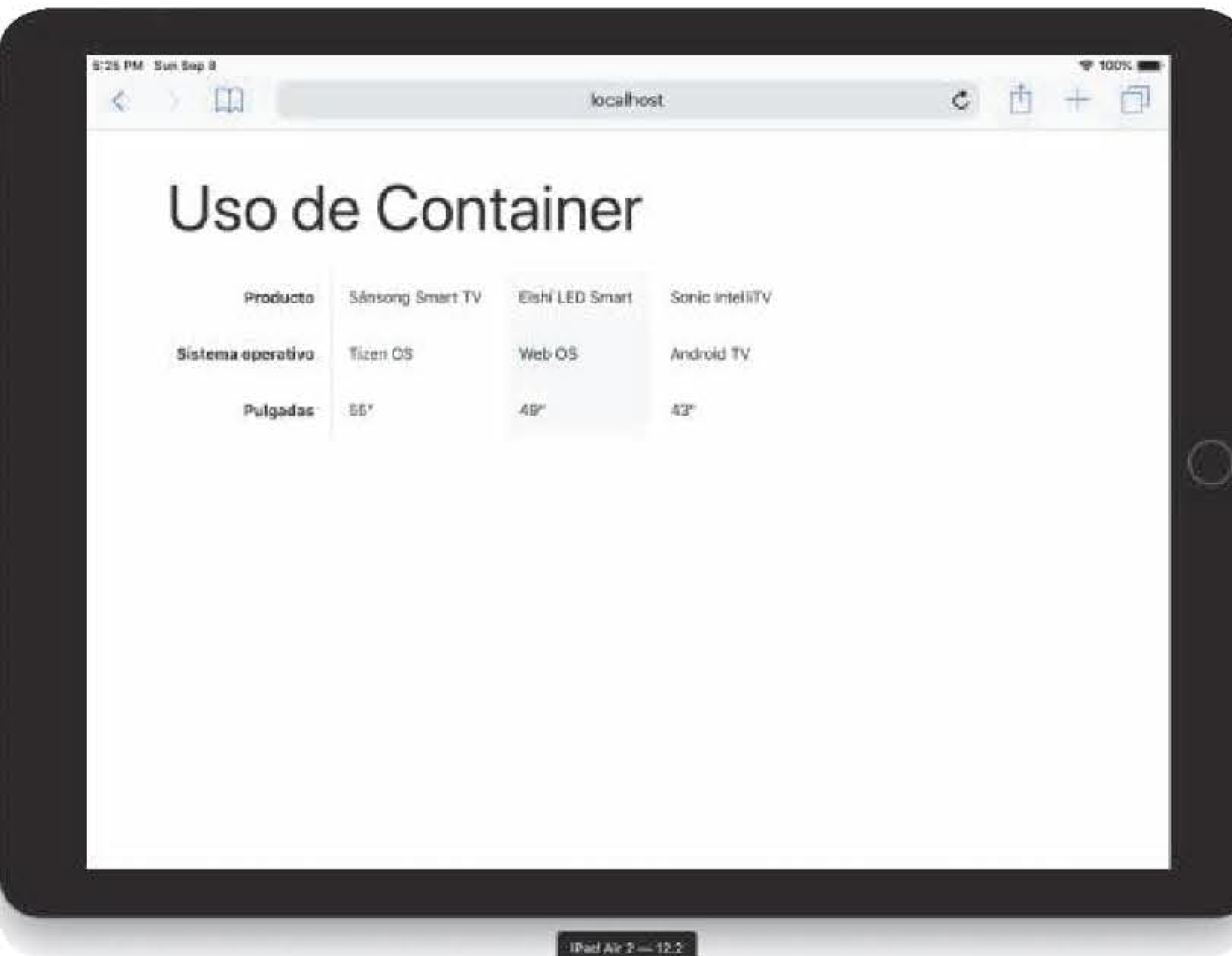
Los invitamos a descargar el ejercicio Tablas Ejemplo.zip del respositorio de material que acompaña a esta obra. Allí está el ejemplo utilizado para representar estas imágenes, con un menú al pie del documento HTML que permite ir aplicando y combinando estas diferentes clases representadas, sobre la tabla de ejemplo.



Contenedor (container)

El sistema Materialize, ciento por ciento abocado al diseño responsivo, permite integrar containers para facilitar el manejo de contenido adaptado a múltiples pantallas. Veamos cómo sacarle provecho.

Materialize piensa su sistema responsivo, al igual que Bootstrap y otros grandes frameworks CSS, como una grilla dividida en doce columnas que aprovechan de forma fluida y simple la visualización y el reordenamiento de los datos según la pantalla donde se carga el proyecto. Este sistema de grilla es ideal para aplicar cuando el contenido de nuestra solución es importante y debe soportar un impensado número de resoluciones de pantalla. Para poder sacar mejor provecho del sistema grid de Materialize, se ideó la clase **container**, la cual ayuda a reestructurar el contenido que se quiere mostrar de una forma más fácil aún.



En la figura anterior vemos el ejemplo de tablas readaptado con la clase **container**. Esta se ocupa de estructurar todo lo que se agrega adentro de un div con dicha clase, ocupando un 70% de la pantalla del dispositivo donde se carga. Su código es muy simple:

```
<body>
  <div class=”container”>
    <!-- Aquí adentro va el contenido de <table> -->
  </div>
</body>
```

En la siguiente imagen representamos cómo se ve y cómo no (respectivamente) el contenido de una web responsiva con y sin la clase container.

Ya con un <div> y su clase **container** creado, nos falta ver de qué manera podemos estructurar el resto del contenido HTML5 de nuestros proyectos. Esto lo realizamos mediante la clase **grid**. Veamos a continuación cómo se utiliza.

Grid

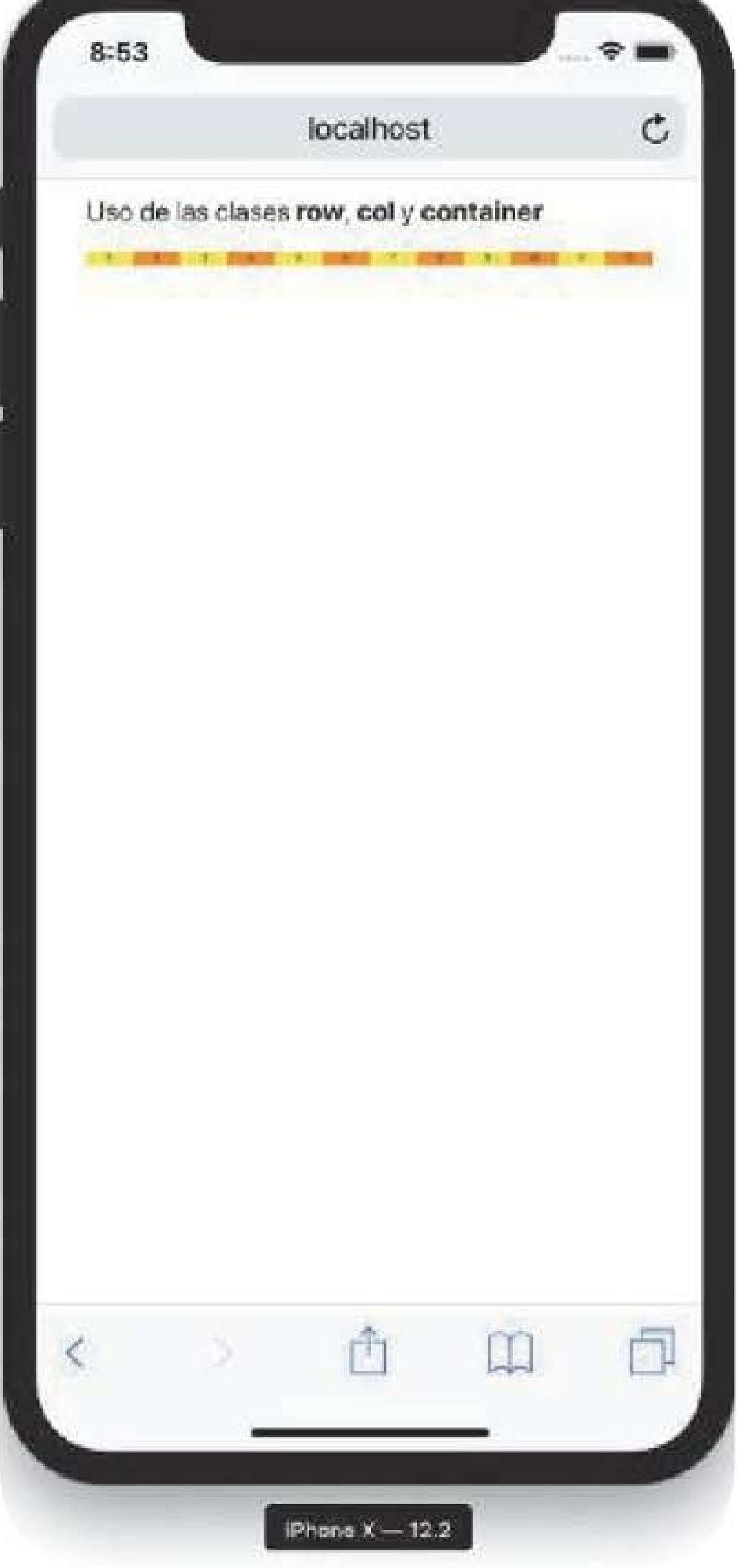
A través del estándar grid y su división de contenido en doce columnas, podemos reestructurar correctamente todos los componentes HTML que debemos incluir en nuestro desarrollo. Las **doce columnas** utilizadas en grid tienen un ancho individual proporcional al de las otras.

Para recrear lo representado en este gráfico, escribimos lo siguiente:

```
<div class="row">
  <div class="col s1">1</div>
  <div class="col s1">2</div>
```

```
    <div class="col s1">3</div>
    <div class="col s1">4</div>
    <div class="col s1">5</div>
    <div class="col s1">6</div>
    <div class="col s1">7</div>
    <div class="col s1">8</div>
    <div class="col s1">9</div>
    <div class="col s1">10</div>
    <div class="col s1">11</div>
    <div class="col s1">12</div>
  </div>
```

Dentro del ejemplo de código anterior, vemos la propiedad **col** aplicada a cada **div**, y la propiedad **s1**, cuyo significado es **small-1**. Traducido a nuestro idioma sería **1 columna en pantallas pequeñas**. A continuación, agregamos color intercalado a las celdas, para entender mejor cómo queda representado el código propuesto anteriormente.

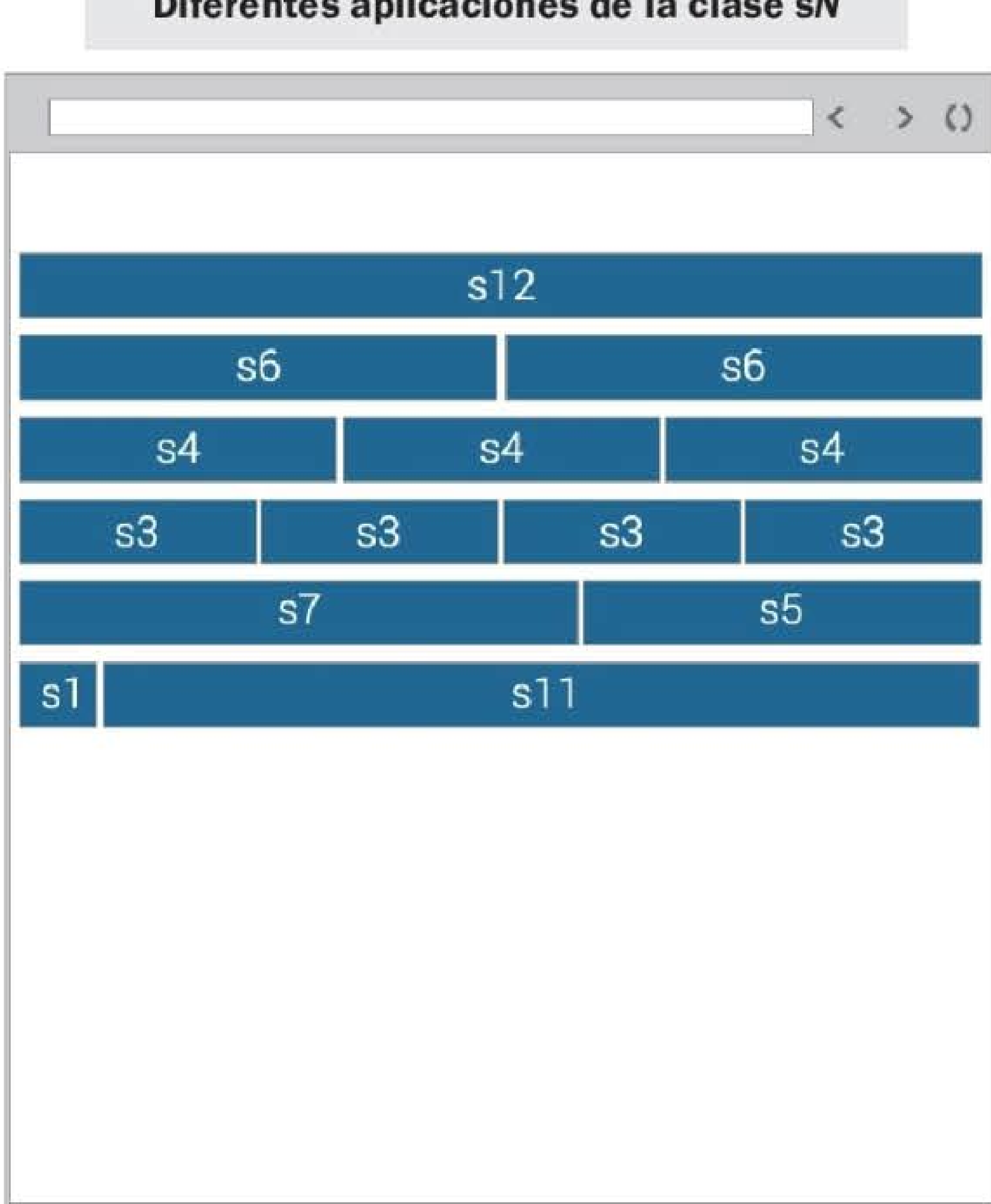


96% 23:22

Este claro ejemplo muestra, como resultado, la división exacta de las doce columnas de manera proporcional, dentro de **container**. Si prestamos atención a la imagen del smartphone, por más que este modelo tenga una pantalla generosa, dicho contenido se verá bastante ajustado. Entonces, ¿cómo podemos resolver este problema? ¡Sigamos leyendo!

Si necesitamos resolver el contenido de nuestro proyecto en diferentes filas, pensando en una pantalla tipo scroll, debemos y podemos establecer distintas columnas que oficien de filas, y hacer que cada "fila" se comporte de determinada manera en cuanto a la definición de ancho. Veamos el siguiente gráfico representativo y, luego, el código que lo construye:

Diferentes aplicaciones de la clase s*N*



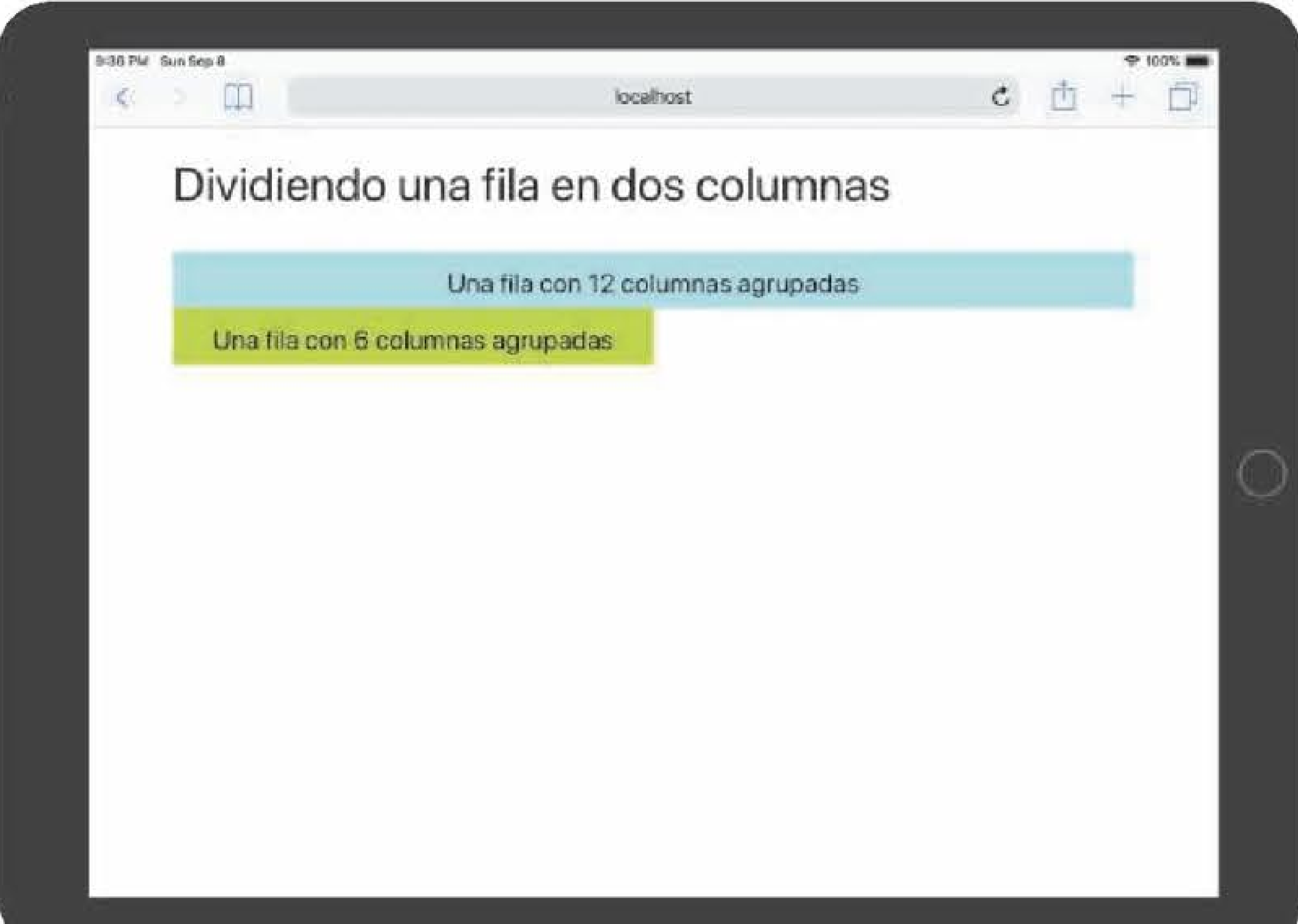
El siguiente código es el que compone las primeras dos filas de la imagen anterior:

```
<div class="row">
    <div class="col s12 orange"><h5>Una fila con 12 columnas agrupadas</
h5></div>
    <div class="col s6 yellow"><h5>Una fila con 6 columnas agrupadas</
h5></div>
    <div class="col s6 grey lighten-1"><h5>Y otra fila con el resto</
h5></div>
</div>
```

De la forma representada en el gráfico y código anterior, vemos de qué manera podemos dividir las columnas de una fila. Gracias a las clases **s12** y **s6**, podemos ocuparnos de reestructurar correctamente el esqueleto de nuestro desarrollo, para luego acomodar el resto de los componentes HTML en el espacio correspondiente.

Si, por ejemplo, deseamos utilizar en una fila solo la mitad de la pantalla para visualizar contenido, podemos definir (pensando en el código anterior) únicamente el primero de los **<div>**, con la clase **s6**. Si lo reemplazamos por **s8**, **s7** o **s10**, utilizaremos entonces un porcentaje mayor o menor del total, **s12**, que acepta la clase grid. Veamos este último ejemplo a continuación:

Aplicando solo la clase **s6** a la columna de una nueva fila, esta ocupa la mitad de pantalla solamente.

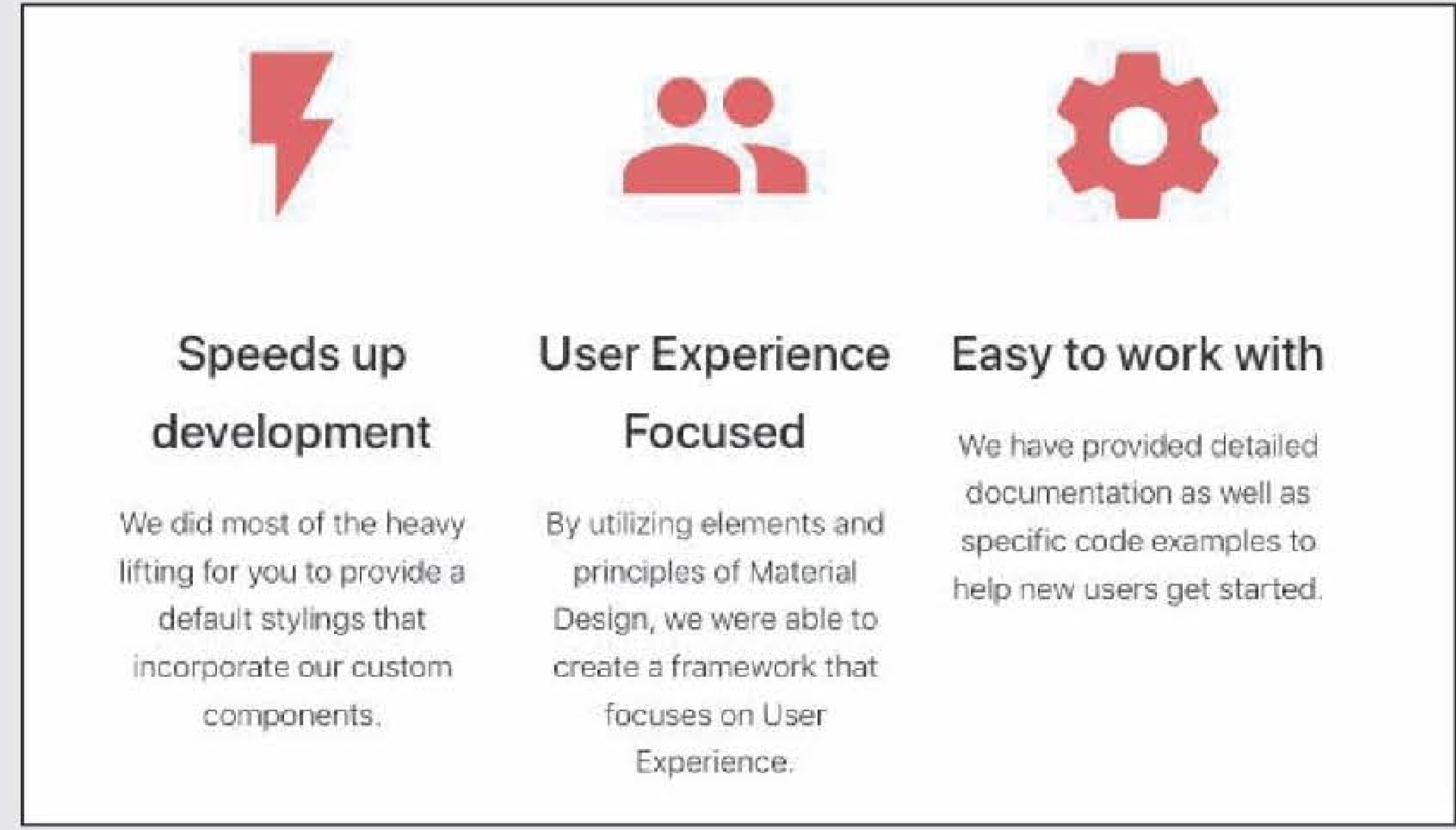


Ejercicio práctico

Para afianzar el conocimiento y poner en práctica las opciones que nos proponen row y container, realizaremos a continuación un ejercicio práctico. La idea es poder estructurar un documento HTML aprovechando lo visto hasta aquí.

El ejercicio debe cumplir con las siguientes consignas:

- ✓ *Utilizar **container**, **rows**, **col** y las clases **sXX** para estructurar su contenido.*
- ✓ *Agregar un título.*
- ✓ *Agregar a continuación un breve texto.*
- ✓ *Luego, una nueva fila con una imagen cualquiera.*
- ✓ *Incluir más párrafos de texto.*
- ✓ *Estructurar luego una sección similar a la de la siguiente imagen.*



✓ *Incluir los iconos, subtítulos y resto de los párrafos, tal como se representa en la figura anterior.*
✓ *Por último, evaluar su comportamiento en diferentes pantallas, para ver si encontramos alguna dificultad en el despliegue del código.*

El resultado de dicho ejercicio debe ser similar al que representamos en la siguiente imagen:

Como ayuda, recomendamos utilizar la etiqueta ***<span>*** *junto a la clase* ***flow-text****, en reemplazo del clásico tag* ***<p>****. Esto permitirá manipular automáticamente el tamaño de letra de los párrafos, ya que cuando tenemos que pensar un único proyecto para varias pantallas sobre las cuales no tenemos control, la etiqueta span y flow-text son los mejores aliados.*

Resolución

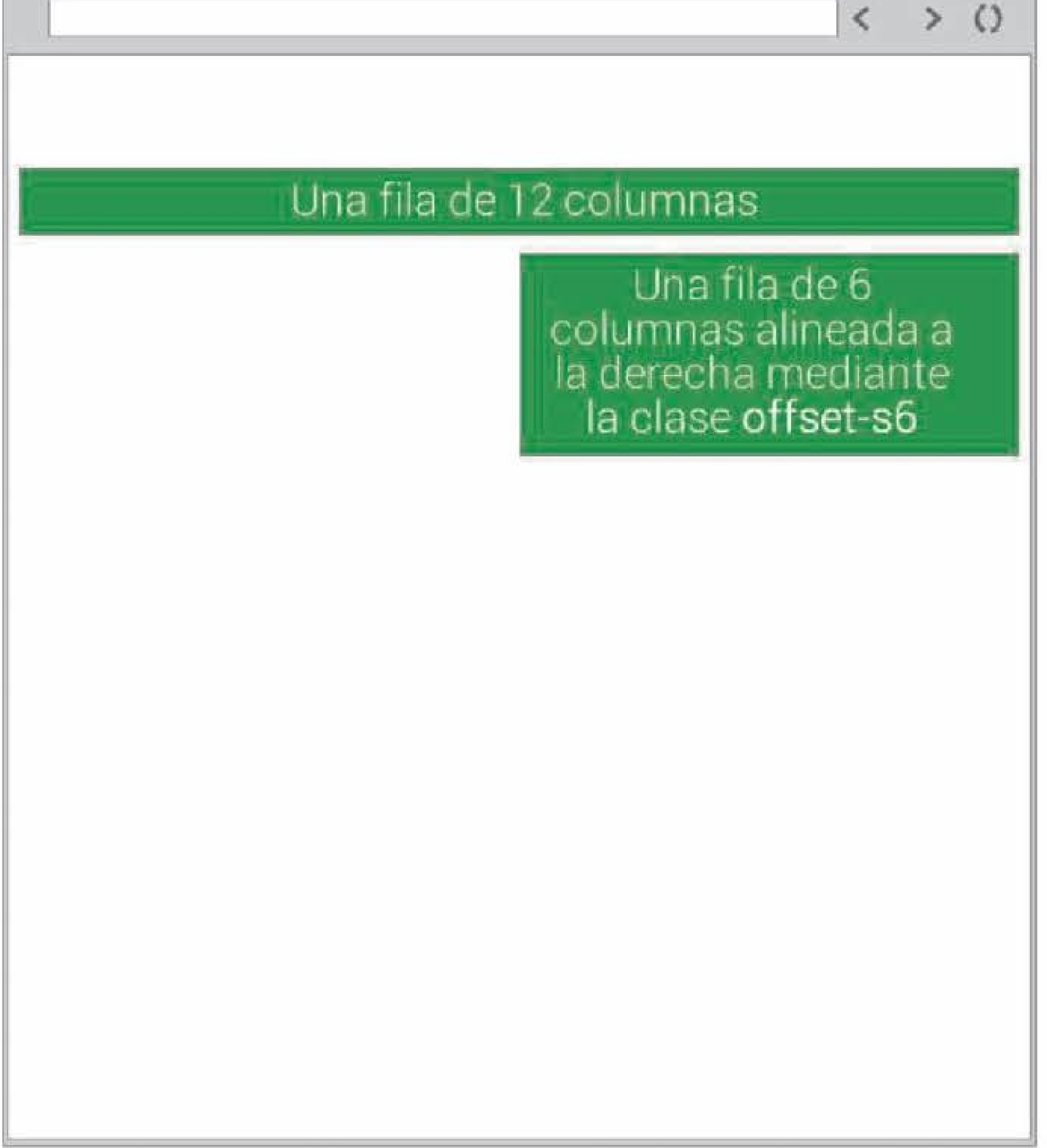
Junto al material que acompaña a esta obra, hay un archivo denominado **Container - Grid - Col - Ejercicio Resuelto.zip**, donde podremos comparar la correcta implementación de cada uno de los puntos de esta práctica.

Offset

En la Figura de la página 36, vimos un ejemplo de cómo aprovechar la clase **s6** para crear un espacio contenedor que ocupe solo la mitad de la pantalla. La alineación de esta fila por defecto es a la izquierda del display, pero ¿qué pasa si queremos hacer lo mismo alineando la fila del lado derecho? Es posible realizarlo, agregando en la fila en cuestión la clase **offset-s6**:

```
<div class="row">
    <div class="col s12"><span class="flow-text">Una fila de 12 colum-
nas</span></div>
    <div class="col s6 offset-s6"><span class="flow-text">6 columnas
alineadas a la derecha</span></div>
</div>
```

Como vemos en este ejemplo de código, en vez de utilizar **<p>** o nada para escribir el texto de la fila en cuestión, usamos **<span>** seguido de la clase **flow-text**. De esta forma, lograremos una mejor fluidez al momento de tener que reacomodar el contenido de cada fila en las diferentes pantallas donde se mostrará el proyecto.



Más clases para manipular filas y columnas

Existe una serie de clases predeterminadas que nos permiten manipular de manera ágil las diferentes filas y columnas, según el tipo de display que cargue nuestra solución. Veamos en la siguiente tabla cuáles son las opciones disponibles, y su significado.

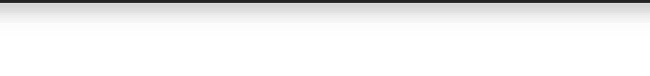
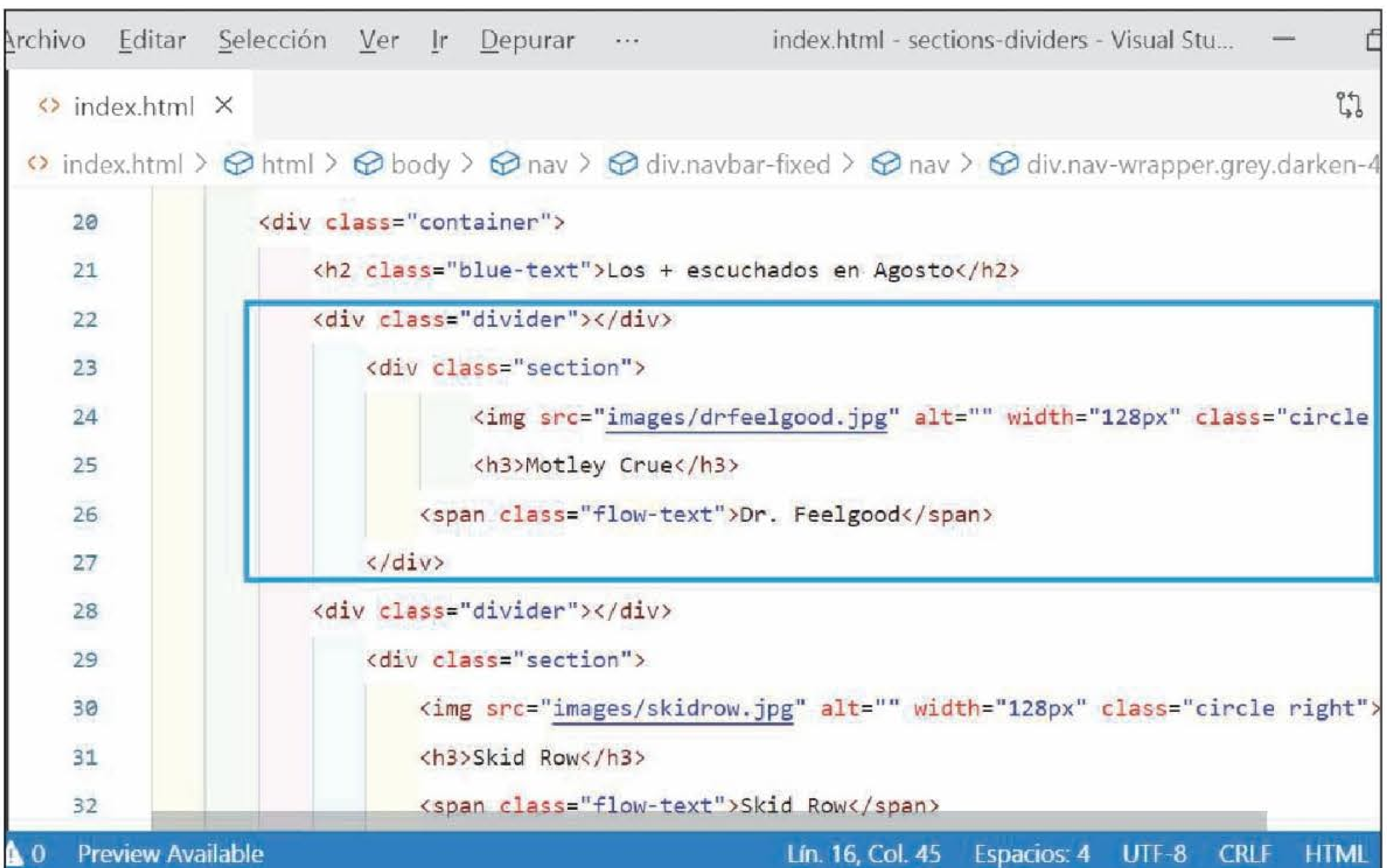


TABLA 4

Clase	Definición
push-sN	Esta clase **push** permite “empujar” el contenido de la fila contigua, hasta que pueda alcanzar a cubrir el resto del ancho del documento HTML. Donde indicamos la letra **N**, se debe reemplazar por un número, que va desde **1** hasta **12**, según el ancho que queramos darle a la celda en cuestión.
pull-sN	Al contrario de la clase anterior, **pull** arrastra hacia su lado la fila contigua, hasta alcanzar a cubrir el resto del ancho del documento HTML. Al igual que en el caso anterior, **N** debe ser reemplazado por un número entre **1** y **12**, según el ancho que le otorgamos a la celda en cuestión.
s- **m-** **l-**	El prefijo de las clases que acompañan a **push**, **pull** y otras se debe definir acorde al ancho de pantalla que deseamos cubrir. **s-** corresponde a **small** (pequeños displays) **m-** corresponde a **medium** (tablets y smartphones de alta gama) **l-** corresponde a **large** (tablets de alta gama y computadoras desktop)

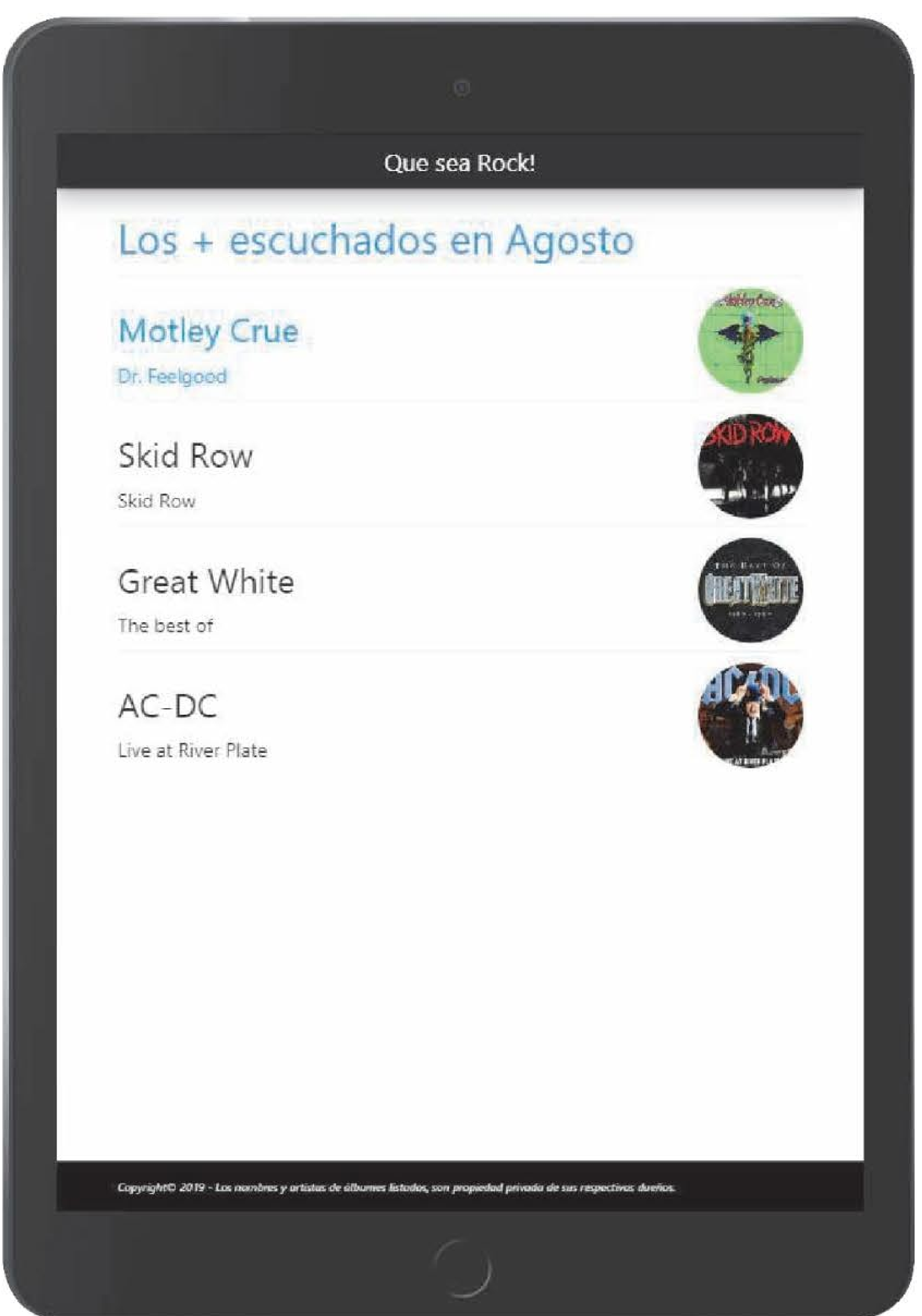
Clases sections y dividers

Finalmente, nos queda repasar la forma de listar contenido de una manera amigable. Materialize nos propone las clases **sections** y **dividers**, para poder armar una lista rápida y de fácil lectura en las diferentes pantallas. Veamos un ejemplo de cómo aprovecharlas.



En este caso vemos la clase **divider** aplicada a un **<div>**, que traza una línea de división como lo haría el tag **<hr>**. Seguido a esto, se declara otro <div> con la clase **section** y, dentro de él, agregamos los diferentes tags que queremos aprovechar para generar contenido específico. Nuestro ejemplo dispone de un título, una breve descripción y una imagen ilustrativa.

Repitiendo estos pasos uno debajo del otro, podemos construir rápidamente un listado informativo rápido y de fácil acceso. En nuestro ejemplo, agregamos también un tag **<img>** con las clases **circle** y **right**, para tornar la figura redonda y alinearla a la derecha de la sección. Como resultado, obtendremos algo similar a lo que se muestra en la siguiente figura:



Si deseamos agregar un hipervínculo en cada **section**, simplemente englobamos su contenido dentro de un tag **<a>**. Veamos un ejemplo a continuación:

```
<div class="divider"></div>
<a href="http://www.google.com.ar/" target="_blank">
    <div class="section">
            <img src="images/drfeelgood.jpg" alt=""
width="128px" class="circle right">
            <h3>Motley Crue</h3>
        <span class="flow-text">Dr. Feelgood</span>
    </div>
</a>
```

Pueden descargar el ejemplo **Sections-Dividers.zip** del repositorio de archivos que acompaña a esta obra, para entender mejor el comportamiento de los componentes HTML sections, tal como los representa este último ejemplo.

Tarjetas

Las tarjetas, o cards, son la solución fundamental aplicada hoy en día por Android para agrupar y ordenar el contenido a visualizar. Veamos a continuación cómo implementarlas de manera efectiva.

Las tarjetas Android son la opción más cómoda para agrupar, de una manera óptima para la visión, todo aquel contenido proveniente de diferentes fuentes. Si utilizamos con frecuencia el sistema operativo Android, veremos que el ofrecimiento de contenido de noticias, clima, y otra información relevante de este sistema se rige usualmente a través de la aplicación conocida, en un principio, como **Google Now**, y todo se agrupa en tarjetas de diferente tamaño y/o contenido gráfico a visualizar.

Veamos a continuación un ejemplo de código:

```
<h5 class=”header”>Ex Ingeniera de Google advierte sobre los robots de
guerra sin supervisión humana</h5>
<div class=”card horizontal z-depth-4”>
    <div class=”card-image”>
        <img src=”http://www.redusers.com/noticias/.../robots.jpg”>
    </div>
    <div class=”card-stacked”>
        <div class=”card-content”>
            <p>Laura Nolan ha sumado su voz a un movimiento internacio-
nal que pide que se prohíba la implementación de...</p>
        </div>
```

Tarjetas

Las tarjetas, o cards, son la solución fundamental aplicada hoy en día por Android para agrupar y ordenar el contenido a visualizar. Veamos a continuación cómo implementarlas de manera efectiva.

Las tarjetas Android son la opción más cómoda para agrupar, de una manera óptima para la visión, todo aquel contenido proveniente de diferentes fuentes. Si utilizamos con frecuencia el sistema operativo Android, veremos que el ofrecimiento de contenido de noticias, clima, y otra información relevante de este sistema se rige usualmente a través de la aplicación conocida, en un principio, como **Google Now**, y todo se agrupa en tarjetas de diferente tamaño y/o contenido gráfico a visualizar.

Veamos a continuación un ejemplo de código:

```
<h5 class=”header”>Ex Ingeniera de Google advierte sobre los robots de
guerra sin supervisión humana</h5>
<div class=”card horizontal z-depth-4”>
    <div class=”card-image”>
        <img src=”http://www.redusers.com/noticias/.../robots.jpg”>
    </div>
    <div class=”card-stacked”>
        <div class=”card-content”>
            <p>Laura Nolan ha sumado su voz a un movimiento internacio-
nal que pide que se prohíba la implementación de...</p>
        </div>
```

96% 23:23


```
        <div class="card-action">
            <a href="http://www.redusers.com/noticias/ex-ingenie-
ra-google-advierte-robots-guerra-sin-supervision-humana/" target="_
blank">Leer nota</a>
        </div>
    </div>
</div>
```

El código anterior da como resultado la tarjeta representada en el siguiente gráfico:

GUÍA VISUAL 5

*En este ejemplo, el título de la tarjeta queda externo a ella misma. Agregamos simplemente un tag del tipo **<h2>** o **<h4>**, o el que más convenga según la pantalla de destino. Y luego añadimos dentro del atributo **class** del título, la clase **header**.*

*En este caso, creamos una tarjeta horizontal. Para hacerlo, declaramos un **<div>** y le damos las clases **card horizontal** a su atributo **class**. El contenido que sigue se agrega en otros **divs** anidados a este.*

*Para agregarle una imagen, utilizamos un nuevo div, y en él agregamos el atributo **class** junto con la clase **card-image**. Luego, dentro de este **<div>**, añadimos el metatag **<img>** convencional, propio de HTML. La clase **card-image** dentro del div anterior se ocupará de controlar la dimensión de la misma.*

*En otro **<div>** colocamos, en el atributo **class**, la clase **card-stacked**. En este caso, como el atributo padre es **card-horizontal**, el contenido de la tarjeta será apilado en forma horizontal. Dentro de este div, añadimos otro con la clase **card-content**. Anidado a él, podemos incluir un párrafo con un texto parcial. Si no agregamos estos últimos divs, la tarjeta no mostrará texto, y quedará la imagen a la izquierda y el hipervínculo inferior a la derecha.*

*Un nuevo **<div>**, con la clase **card-action**, generará contenido al pie de la tarjeta, donde, por ejemplo, podemos incluir un hipervínculo hacia la nota original, o abrir una ventana emergente para mostrar el desarrollo de la nota completa, tal como actualmente lo hace Android.*

*Finalmente, si aplicamos al atributo **class** del div padre la propiedad **z-depth-N** (donde N es un número entero entre 1 y 6), generaremosun efecto de sombra con diferente profundidad sobre la tarjeta creada.*

Como podemos observar, Materialize CSS hace que la creación de componentes HTML del tipo cards resulte muy fácil e intuitiva. En el ejemplo anterior vimos cómo se crea una tarjeta simple del tipo horizontal. Por suerte, Materialize CSS nos ofrece otros tipos de tarjetas que podemos personalizar fácilmente, cambiando la clase del div padre y anidando otros divs con forma de componentes.

Veamos a continuación una tabla con las diferentes opciones de clases disponibles:

TABLA 5

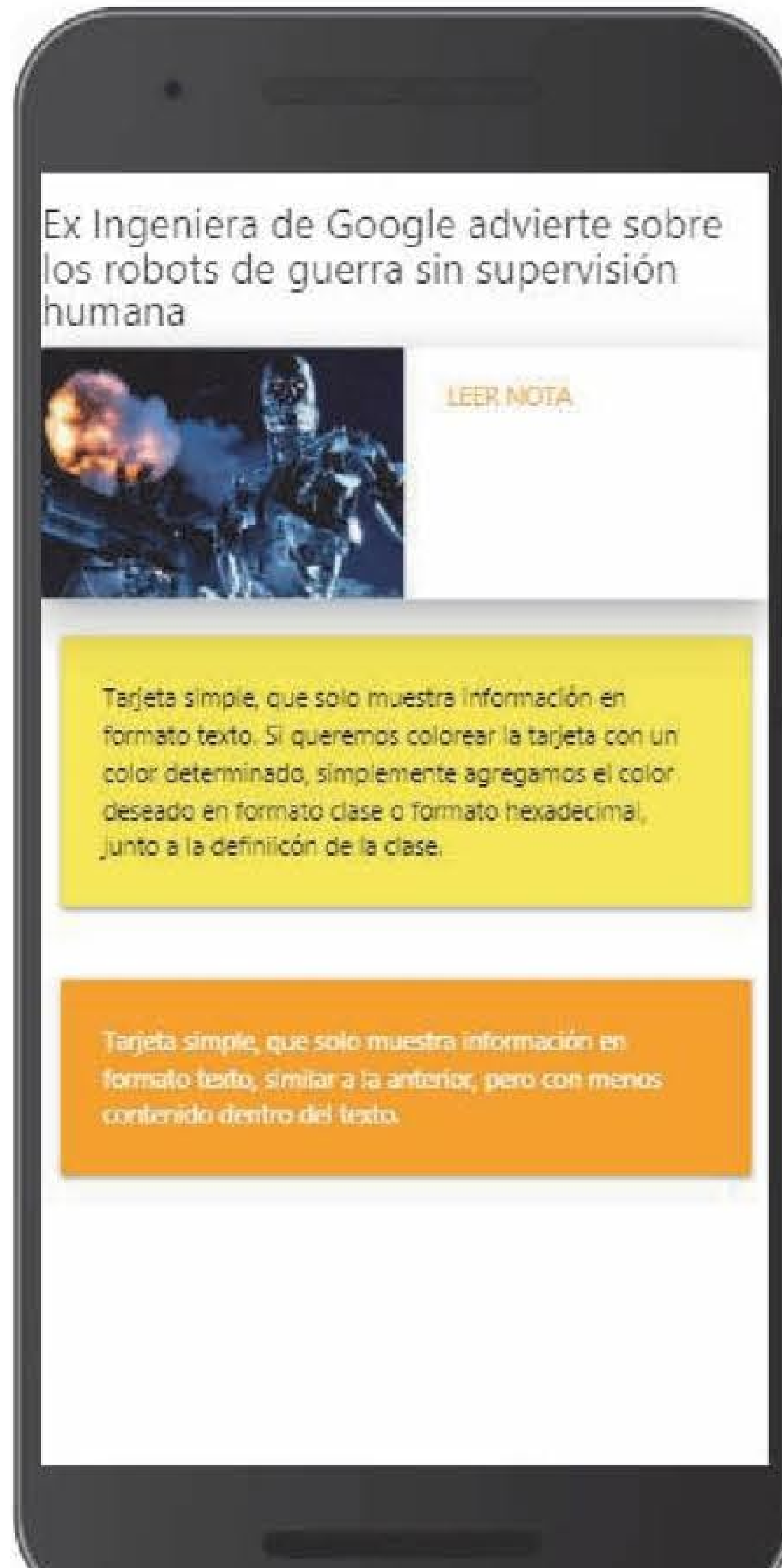
Clase	Div para implementar	Descripción
card-content	Padre	Tarjeta básica, la cual incluye solamente título y texto como contenido.
card-title	Anidado dentro de Padre	Se utiliza dentro del atributo class del metatag **<span>**, y permite crear un título dentro de la tarjeta en sí.
card	Padre	Se usa para crear una tarjeta del tipo vertical y, si le agregamos una imagen, el uso del atributo **card-title** será reflejado sobre la figura que incluyamos.
card-horizontal	Padre	Igual a la clase **card**, pero define la tarjeta en formato horizontal en vez de vertical.
card-reveal	Anidado dentro de Padre	Permite agregar información adicional a la tarjeta, desplegando una ventana emergente que la cubrirá y mostrará contenido del tipo texto. En el interior del div que contiene esta clase, se declara un metatag del tipo **<span>** junto con el texto por mostrar.
card-tabs	Anidado dentro de Padre	Permite definir una serie de tabuladores o pestañas dentro de la tarjeta creada. Estos **tabs** se visualizan como hipervínculos y, al hacer clic sobre ellos, en el apartado inferior contiguo, se muestra información en formato texto relacionada a dicho **tab**.

Finalmente, junto a las clases **card** y **card-horizontal**, podemos agregar una segunda clase que las acompañe, para definir el tamaño de la tarjeta que se va a crear. Las medidas posibles de tarjeta son: **small**, **medium** y **large**.

Card panel

Utilizando la clase **card-panel**, creamos un panel del tipo tarjeta simple, que acepta solo texto, y es posible aprovechar las clases que definen los colores para darle un color de fondo a la tarjeta y otro al texto que se debe visualizar. También, al igual que en la infografía anterior, si utilizamos la clase **z-depth-N**, podremos darle un sombreado que denote más su formato de tarjeta.

En los archivos que acompañan a esta obra, se encuentran los ejemplos aquí explicados, bajo el proyecto denominado **Ejemplo-Cards.zip**.



Carrusel de imágenes

Materialize CSS también pone a nuestra disposición una galería de imágenes en formato carrusel. Veamos a continuación cómo crear un efecto slide sobre un conjunto de fotografías, y profesionalizar aún más nuestros desarrollos móviles.

Este componente nos permite definir dos tipos de galerías de imágenes. Una de ellas simula un formato 3D o del tipo solapamiento:

Y la otra es en formato del tipo deslizamiento, desde el lateral derecho hacia el izquierdo.:

La implementación del componente **Carousel** se debe realizar en dos partes. Primero, en un documento HTML agregamos un div, e incorporamos las clases **carousel** y **carousel-slider**. Luego, dentro de este div, agrupamos un metatag del tipo **<a>** con la clase **carousel-item** y, dentro de este, referenciamos un tag **<img>** con el path hacia el archivo gráfico en cuestión. Veamos un ejemplo a continuación:

```
<div class="carousel carousel-slider yellow lighten-1">
    <a class="carousel-item" href="#one!"><img src="images/ika1.jpg"></
a>
    <a class="carousel-item" href="#two!"><img src="images/ika2.jpg"></
a>
    <a class="carousel-item" href="#three!"><img src="images/ika3.
jpg"></a>
    <a class="carousel-item" href="#four!"><img src="images/ika4.
jpg"></a>
    <a class="carousel-item" href="#five!"><img src="images/ika5.jpg"></
a>
</div>
```

La otra parte que compone al carrusel corresponde al código JavaScript, siendo este último quien le da vida al efecto de transición entre un archivo gráfico y el siguiente. Desde JavaScript inicializamos el componente en cuestión, y le especificamos un comportamiento a través de una serie de propiedades y valores.

Propiedades y valores de inicialización de Carousel

Veamos a continuación qué elementos JavaScript pueden utilizarse para inicializar y aplicar un efecto de transición a Carousel.

TABLA 6

Nombre	Valor	Descripción
duration	Numérico	Especifica en milisegundos el tiempo de duración entre una imagen y la siguiente.
dist	-100 a 0	Indica una distancia en píxeles entre una imagen y las otras.
numVisible	Numérico (5 por defecto)	Especifica el número de ítem visibles, principalmente, para aplicar en el carrusel tipo 3D.
indicators	Booleano	Muestra u oculta los indicadores de imágenes.

96% 23:23

Existen más propiedades, como **fullWidth**, **noWrap**, **onCycleTo**, **shift** y **padding**. Es posible encontrar más información de ellas en la página oficial del componente: https://materializecss.com/carousel.html.

Métodos

La clase Carousel incluye también una serie de métodos JS que nos permitirán actuar sobre el carrusel de imágenes creado. Estos métodos son:

.next(): avanza a la siguiente imagen.
.prev(): retrocede a la imagen anterior.
.set(n): mueve el carrusel a un número de slide especificado dentro de los paréntesis.
.destroy(): elimina el efecto del carrusel de imágenes, si está en modo automático.

Ejercicio práctico: compra-venta

Descargamos el archivo **Ejemplo Carrousel - Base.zip** que acompaña a esta obra, descomprimimos el contenido e iniciamos un nuevo proyecto con ella en Visual Studio Code. Si ejecutamos el proyecto como está, veremos una simple página que simula el apartado detalle de un sitio de compra-venta.

Este ejemplo tiene un botón en la barra superior, con el icono de una cámara fotográfica. Ya tiene cargado el componente Carousel en el documento HTML, pero no está inicializado. Lo que haremos será agregarle un código JS para que el carrusel tenga funcionalidad. Para hacerlo, en el apartado <head>, ubicamos la declaración del archivo externo JS y hacemos Ctrl + clic sobre él.

```
<script src="js/carousel.js"></script>
```

VS Code nos ofrecerá crearlo ya que no existe.

Objetivos de la práctica

- Setearemos el componente Carousel para que funcione como galería.
- Agregaremos una función al botón cámara que active la galería automáticamente.
- Usaremos el método JS **carousel.next()** para pasar las imágenes en forma automática.

Controlar la carga del documento HTML

En principio, agregamos la sentencia que controla la carga total del documento HTML antes de ejecutar cualquier código JS:

```
document.addEventListener('DOMContentLoaded', function() {

});
```

En el documento HTML, utilizamos un atributo ID en el botón cámara, para llamarlo **btnfotos**; y otro ID en el componente Carousel, para llamarlo **imagenesCarousel**.

Inicializamos a continuación el componente Carousel, con el siguiente código:

```
var el = document.getElementById("imagenesCarousel");
var opciones = "duration: 200, indicators: true, dist:
-100";
var instancia = M.Carousel.init(el, opciones);
```

Como opciones del componente, especificamos la distancia de 100 píxeles entre una imagen y otra, lo que le dará un efecto tipo 3D. La duración de transición entre imágenes será de 200 milisegundos. Luego, creamos una función que utilizará el método **.next()** para transicionar las imágenes:

```
function encenderCarrousel() {
    instancia.next();
}
```

Por último, agregamos el evento clic en el botón superior, el cual simplemente ejecuta la función JS **setInterval** (intervalo) cada 3 segundos, e invoca la función de transición anterior:

```
document.getElementById("btnfotos").addEventListener("click", function() {
    setInterval(encenderCarrousel, 3000)
})
```

La estructura del código debe quedar como muestra la siguiente figura:

```
index.html    carousel.js ×
js > carousel.js > ...
1   document.addEventListener('DOMContentLoaded', function() {
2
3       var el = document.getElementById("imagenesCarousel");
4       var opciones = "duration: 200, indicators: true, dist: -100";
5       var instancia = M.Carousel.init(el, opciones);
6
7       function encenderCarrousel() {
8           instancia.next();
9       }
10
11      document.getElementById("btnfotos").addEventListener("click", function() {
12          setInterval(encenderCarrousel, 3000)
13      })
14  });
```

Resultado del ejercicio

Finalmente, podemos probar nuestro desarrollo, habiendo guardado previamente los archivos editados. El resultado debe ser una transición automática entre las cinco imágenes que componen este ejemplo.

En el repositorio de material que acompaña a esta obra es posible encontrar este proyecto terminado, bajo el nombre **Ejemplo Carrousel - Resuelto.zip**. Es conveniente tenerlo a mano, ya que el próximo componente que veremos enriquecerá aún más este ejercicio.

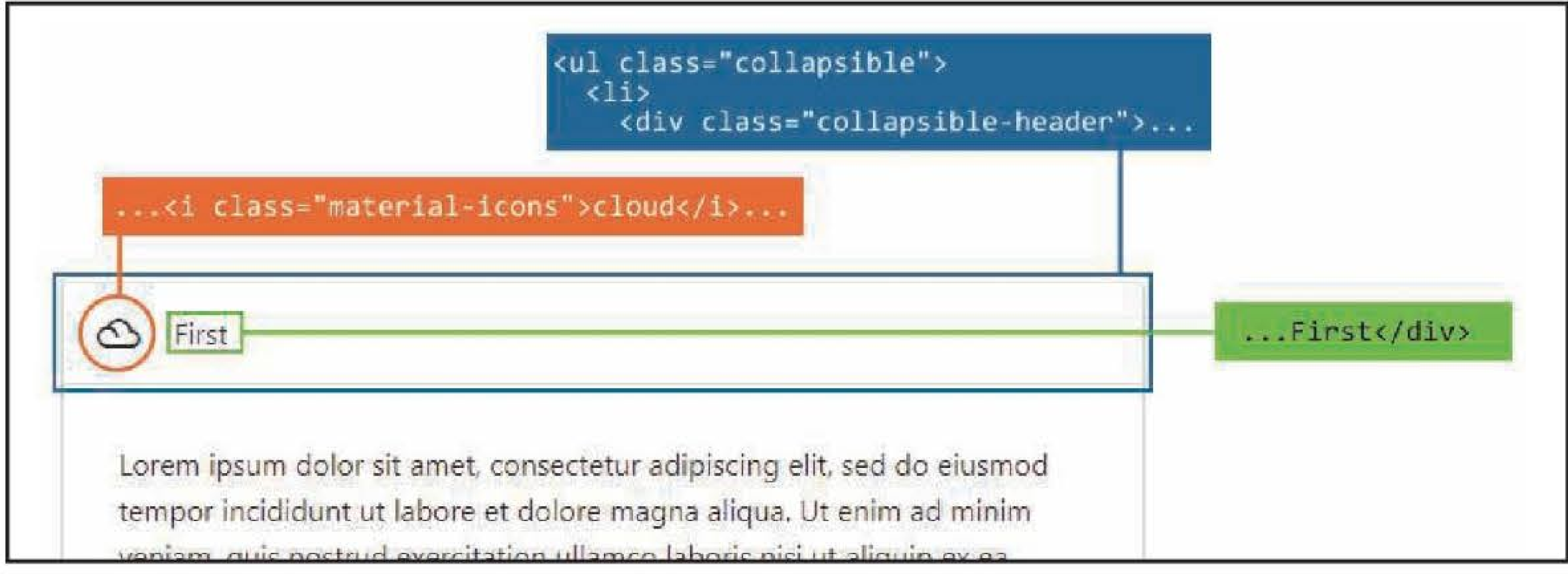
Elementos expansibles

En las pantallas móviles, es crucial poder simplificar contenido, anidando el exceso de textos dentro de componentes dedicados para tal fin. **Collapsible** es la propuesta de Materialize que nos permite llevar a cabo esta tarea, dejando que el usuario sea quien decida qué información ver y cuál no.

Veamos a continuación cómo implementar esta fabulosa opción.

Componentes HTML

Collapsible se conforma de dos partes: la declaración de los componentes HTML y la inicialización desde JS. Para implementarlo, inicializamos un elemento HTML **<ul>** con la clase **Collapsible**. Dentro de este iniciamos un elemento **<li>** seguido de un **<div>** con la clase **collapsible-header**, y eventualmente, un icono y título asociado a él.



96% 23:23

Ahora nos queda definir el cuerpo de cada ítem que estará inicialmente oculto. Para hacerlo, a continuación del <div> anterior agregamos un nuevo div, que tendrá en su atributo **class**, la clase **collapsible-body**:

```
   <div class="collapsible-body"><span>Lorem ipsum dolor sit amet...</
span></div>
</li>
```

Repetimos esto hasta finalizar el último elemento que conformará nuestro componente Collapsible. Veamos un ejemplo de código completo:

```
<ul class="collapsible">
  <li>
    <div class="collapsible-header"><i class="material-icons">filter_
drama</i>First</div>
    <div class="collapsible-body"><span>Lorem ipsum dolor sit amet.</
span></div>
  </li>
  <li>
    <div class="collapsible-header"><i class="material-icons">place</
i>Second</div>
    <div class="collapsible-body"><span>Lorem ipsum dolor sit amet.</
span></div>
  </li>
  <li>
    <div class="collapsible-header"><i class="material-icons">whatshot</
i>Third</div>
    <div class="collapsible-body"><span>Lorem ipsum dolor sit amet.</
span></div>
  </li>
</ul>
```

El bloque de código anterior dará un resultado similar al de la siguiente imagen:

First

Second

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

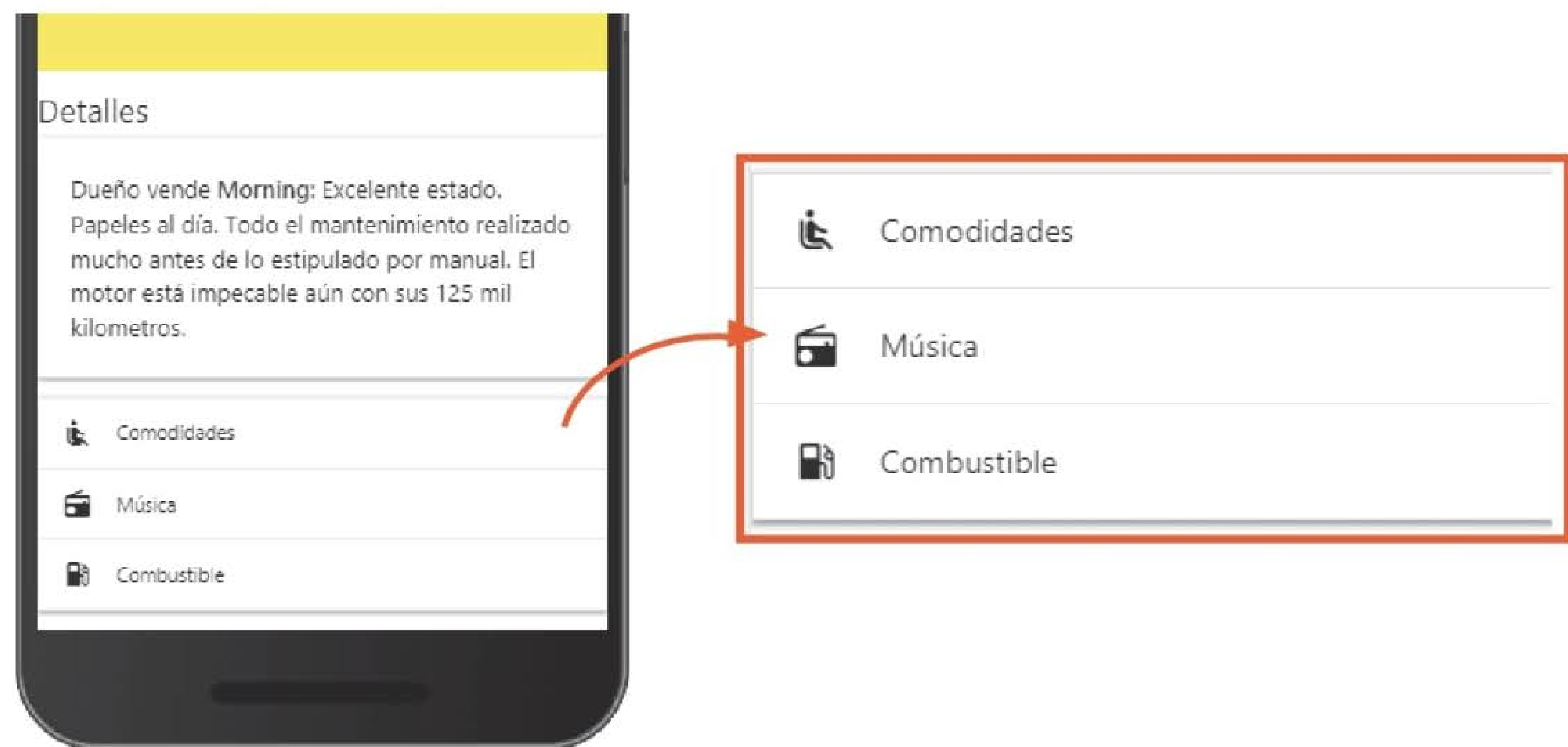
Third

96% 23:23

Inicialización del componente

Si probamos este último código en un documento HTML, veremos simplemente tres elementos y no podremos acceder a su contenido. Deberemos inicializarlo desde JS. La inicialización requiere establecer una serie de parámetros sobre Collapsible para definir su comportamiento. *(Ver páginas 46 y 48, Capítulo 3)*

Veamos cómo inicializarlo directamente sobre un ejercicio práctico. Del material que acompaña a esta obra, recuperemos el archivo **Ejemplo Collapsible - Base.zip**. Este parte del código base del ejercicio anterior, resuelto, y le agrega un componente **Collapsible**, con tres ítem de contenido, que corresponden a un detalle más profundo en la descripción de las características de un vehículo que se encuentra a la venta.



Vamos a ocuparnos entonces de darles vida a estos componentes HTML desde el lado de JS. Para hacerlo, editamos el archivo JavaScript vinculado a este proyecto, y vamos a la línea final, justo antes del cierre del **eventListener**. Allí agregamos lo siguiente:

```
var coll = document.querySelectorAll('.collapsible');
```

En esta línea declaramos una variable **coll**, la cual queda vinculada a todos los elementos **Collapsible** que tenga el proyecto, a través del método **querySelectorAll()** del objeto JS **document**. A continuación, debemos declarar otra variable, donde les asignamos un valor específico a tres propiedades del componente: **accordion**, **inDuration** y **outDuration**:

```
var efectos = "accordion: true, inDuration: 400, outDuration: 400";
```

96% 23:23

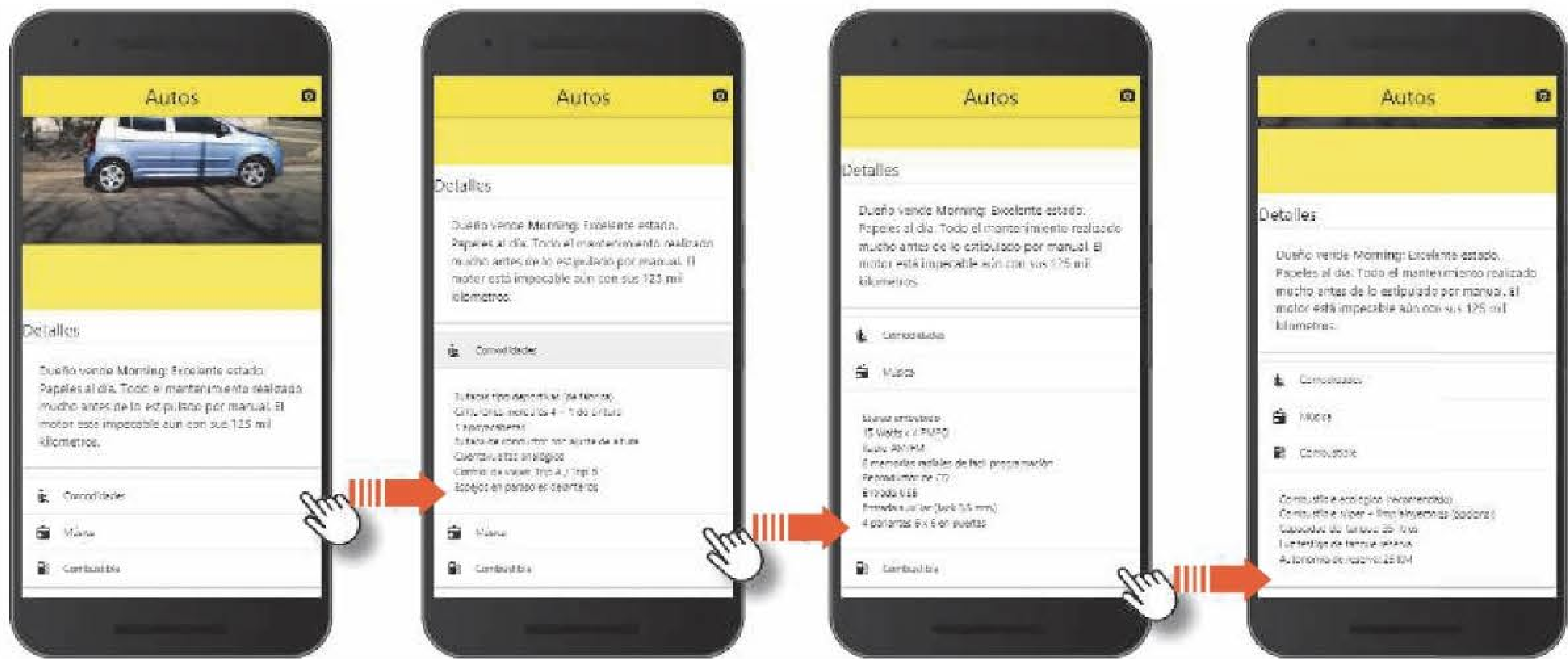
La variable **efectos** se ocupa de establecer que el componente HTML generará un efecto acordeón; la duración de transición de este efecto, tanto de inicio como de fin, será de 400 milisegundos cada una. Por último, iniciamos el componente con la siguiente línea de código:

```
var inst = M.Collapsible.init(coll, efectos);
```

El código debe quedar estructurado como muestra la siguiente imagen:

```
index.html    carousel.js
js > carousel.js > ...
 9        }
10
11        document.getElementById("btnfotos").addEventListener("click", function() {
12            setInterval(encenderCarrousel, 3000)
13        })
14
15        var coll = document.querySelectorAll('.collapsible');
16        var efectos = "accordion: true, inDuration: 400, outDuration: 400";
17        var inst = M.Collapsible.init(coll, efectos);
18    });
```

Con esto, estamos en condiciones de probar el funcionamiento de nuestra solución. Cargamos la misma en el simulador mobile de Google Chrome o en nuestro teléfono móvil para ver su comportamiento.



96% 23:23

Propiedades y valores del componente Collapsible

En el apartado **Options** de la ayuda oficial de Materialize CSS se encuentran todas las propiedades, valores y explicaciones que es posible aplicarle a este componente: https://materializecss.com/collapsible.html.

Métodos del componente Collapsible

Al igual que Carousel, Collapsible cuenta con una serie de métodos que nos permiten controlar la apertura, el cierre y el desplazamiento de cada ítem. Estos métodos son:

open(): abre una sección determinada, que se identifica con un ID, el cual se indica como parámetro entre los paréntesis.
close(): cierra la sección determinada entre paréntesis.

Tipos de efectos en Collapsible

Acompañando la clase Collapsible, encontramos una serie de efectos específicos que permiten cambiar la animación de la transición entre un ítem y otro. Ellos son:

- **Accordion**: es el efecto predeterminado que acompaña a Collapsible. Podemos escribirlo o no como clase asociada a Collapsible. Si no lo incluimos, Materialize lo asume como predeterminado.
- **Expandable**: funciona como Accordion, pero el efecto que produce entre la transición de un ítem y otro es similar a un deslizamiento.
- **Popout**: genera una animación del tipo expansión, dándole mayor protagonismo al ítem seleccionado y destacándolo un relieve superior, por sobre el resto. Veamos el resultado de este efecto en la siguiente imagen:

El archivo **Ejemplo Collapsible - Resuelto.zip** contiene la resolución funcional de este último ejercicio práctico.

Preselected

Si al cargar un documento HTML que contiene un apartado con varios elementos Collapsible, podemos hacer mediante código, que uno de ellos aparezca pre-seleccionado de forma automática, esto hará que dicho elemento aparezca abierto cuando dicho documento es cargado en un navegador o motor web.

Para que esto ocurra, debemos agregar en el elemento **<li>** correspondiente a la sección que queremos que aparezca desplegada, la clase active:

```
<li class="active">
...
```

Tengamos presente que sólo podremos utilizar la clase active una vez.

04 Mensajes de información e interacción

La interacción y la información son elementos importantes, más aún, cuando de pantallas diminutas se trata. Veamos qué nos propone Materialize para optimizar y agrupar contenido, e interactuar con los usuarios a través de mensajes informativos.

Las pantallas móviles, ya sea de tablets y/o smartphones, usualmente nos limitan mucho la capacidad de explayarnos en cuanto a contenido de texto y multimedia. Y , bajo la premisa de ser lo más concisos posible, Materialize cuenta con varias opciones que nos hacen el desarrollo más flexible y práctico.

Colecciones

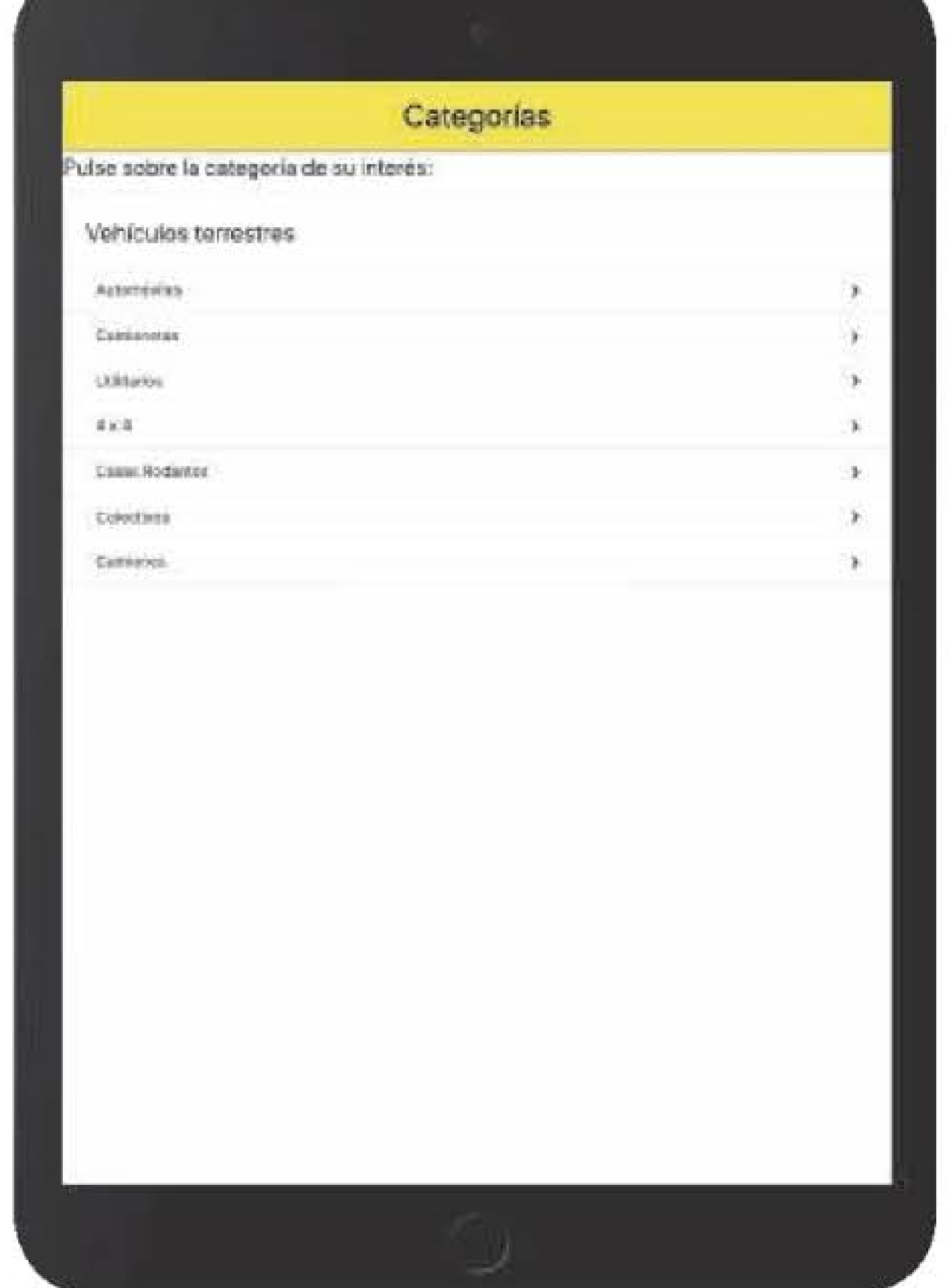
Con el nacimiento y la masividad de los dispositivos móviles basados en pantallas táctiles, el uso de colecciones para listar elementos en pantalla es algo más que común. Hasta el momento, la forma más práctica de ordenar mucho contenido es utilizando estas colecciones y, a su vez, anidando y estableciendo diferentes niveles de acceso, por ejemplo, a subcategorías de productos.

Materialize CSS pone a nuestra disposición la clase **Collection**, la cual, utilizando un sistema anidado de elementos HTML **<ul>** y **<li>**, permite listar contenido de forma clara y ordenada. Veamos a continuación un ejemplo de código:

```
<ul class="collection with-header">
    <li class="collection-header"><h5>Vehículos terrestres</h5></li>
    <li class="collection-item"><div>Automóviles<a href="automoviles.
html" target="_self" class="secondary-content"><i class="material-icons
black-text">chevron_right</i></a></div></li>
    <li class="collection-item"><div>Camionetas<a href="#!"
class="secondary-content"><i class="material-icons black-text">chevron_
right</i></a></div></li>
    …
</ul>
```

La lista se compone del elemento HTML **<ul>** con la clase **collection** y la clase **with-header**. Esta última se ocupa de indicarle al documento HTML que esta colección de elementos contendrá un encabezado. El elemento siguiente, **<li>**, contiene la clase **collection-header**, la cual anida un tag del tipo título, con la descripción de este.

A partir del segundo elemento **<li>**, utilizamos la clase **collection-item** para empezar a listar los elementos. De forma anidada a este elemento, agregamos un metatag **<i>** con la clase **material-icons**, para poder generar un icono. En nuestro ejemplo, utilizamos el icono **chevron_right** para crear una flecha hacia la derecha que indica el avance. Finalmente, encontramos anidado el elemento **<a>**, el cual se ocupa de generar un hipervínculo hacia otra URL o documento HTML.

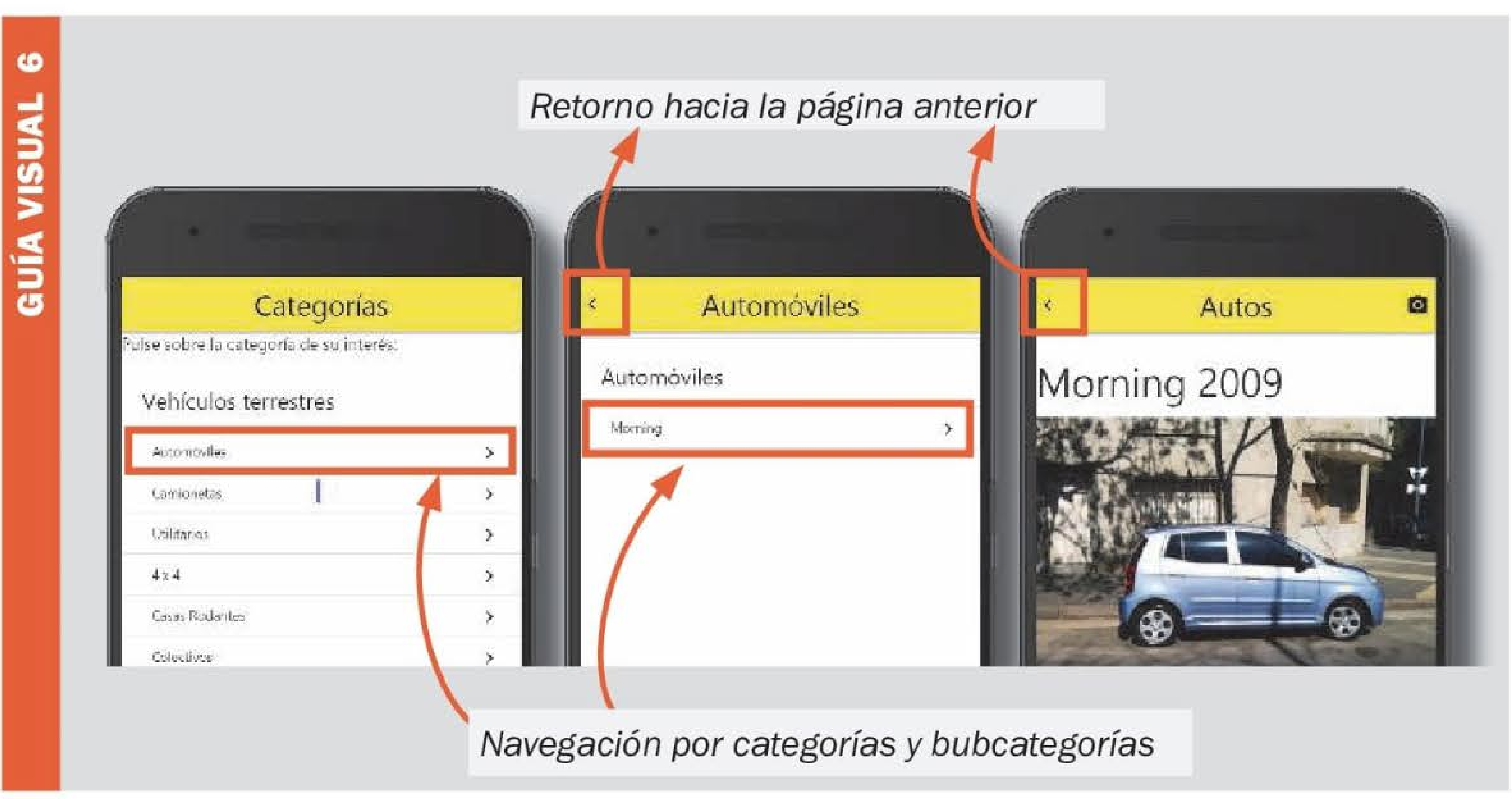


96% 23:23

Ejemplo práctico: navegación por categorías

Descargamos del material adjunto a esta obra el archivo **Ejercicio Collections - Base.zip**. Lo descomprimimos y creamos un nuevo proyecto en VS Code a partir de la carpeta de este ejercicio. Este ejemplo está compuesto por tres documentos HTML; uno de ellos, relacionado al ejercicio anterior de la página de venta de un automotor.

En esta adaptación del ejercicio previo, agregamos una página inicial de categorías, una segunda página con subcategoría y finalmente la página de descripción del vehículo. A su vez, cuando iniciamos la inmersión dentro de las subcategorías, en la barra de navegación superior tenemos el icono **chevron_left**, que nos permite regresar a la página anterior.



Navegación por categorías

El ejercicio descargado ya tiene resuelta la estructura básica de las colecciones, por lo que solo nos ocuparemos de darle funcionalidad de navegación. Si bien cada elemento **Collection** tiene una etiqueta **<a>** y su atributo **href** para agregarle un hipervínculo, realizaremos la navegación funcional integramente desde JavaScript.

A continuación editaremos **index.html**, **automoviles.html** y **morning.html**. En el apartado **<head>** de cualquiera de estos documentos HTML ubicamos la referencia **<script>** hacia un archivo denominado **navegar.js**.

Presionemos **Ctrl + clic** sobre el archivo para crearlo. Ya dentro de este, agregaremos dos funciones JS, que nos permitirán navegar fácilmente por las categorías y subcategorías de este sitio, y retornar también hacia el nivel anterior.

96% 23:23


```
index.html    automoviles.html ×    morning.html
automoviles.html > html
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <meta charset="utf-8">
5           <title>Compra-Venta de vehículos</title>
6           <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=n
7           <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
8           <link href="https://fonts.googleapis.com/css?family=Roboto&display=swap&subset=cyrillic,latin-ext" rel="
9           <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.mi
10          <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>
11          <script defer src="js/navegar.js"></script>
12      </head>
13      <body>
14          <div class="navbar-fixed">
15              <nav>
16                  <div class="nav-wrapper yellow">
17                      <a href="#!" class="brand-logo black-text">Automóviles</a>
```

History.back

```
function retornar() {
    history.back(-1);
}
```

En este bloque de código, creamos una función llamada **retornar()**, la cual contiene una simple línea de código. En ella encontramos el elemento JS **history**, que se ocupa de llevar el historial de navegación y de permitirnos navegar por él. History incluye un método denominado **back()** que inicia la navegación hacia las páginas o sitios anteriormente navegados. Este último método recibe un parámetro numérico, donde podemos especificar cuántos niveles de navegación hacia atrás, deseamos. Aquí solo especificamos que retorne al punto inmediato anterior de donde nos encontramos.

Location.href

```
function navegarCategorias(d) {
    location.href = d + '.html';
}
```

Aquí vemos que la nueva función se llama **navegarCategorias()** y recibe un parámetro denominado **d**. Dentro de la función también nos encontramos con una simple línea de código que contiene el elemento **location**, perteneciente al objeto JS **window**. Location permite obtener y realizar operaciones sobre la navegación actual. La propiedad **href** del elemento location permite obtener la URL actual o establecerla. Esto último nos da la posibilidad de ejecutar una navegación hacia un sitio o documento HTML.

En este ejemplo estamos realizando esta última acción, asignándole como parámetro lo que llega al método y concatenándole el texto **.html**. Entonces, cuando utilicemos esta función, simplemente pasaremos como parámetro el nombre del documento HTML al cual deseamos navegar, y **location.href** se ocupará de llevarnos hacia esa página. Apliquemos esto mismo en los documentos HTML.

Abrimos **index.html y ubicamos el elemento ** con la clase **collection-item** con **id = "automoviles"**. A continuación del atributo id, incluimos el siguiente código:

```
onclick="navegarCategorias('automoviles');"
```

La línea completa de código debe quedar conformada del siguiente modo:

```
<li class="collection-item" id="automoviles" onclick="navegarCategor
ias('automoviles');"><div>Automóviles<a href="#!" class="secondary-
content"><i class="material-icons black-text">chevron_right</i></a></
div></li>
```

Onclick es otro atributo de los elementos HTML, en donde podemos especificar un bloque de código JavaScript o, como en este caso, una función JS. De esta manera, agregamos la función JS que permite navegar hasta una categoría. En este caso, la categoría especificada por parámetro en dicha función hace referencia directamente al documento HTML donde deseamos dirigir la aplicación. Esto le otorgará un poco de dinamismo a nuestro desarrollo.

Navegación hacia atrás

Abrimos a continuación el documento **automoviles.html**, donde haremos lo propio agregándole el código JS para aplicarle funcionalidad. Ubicamos el apartado **<nav>** y, dentro de él, ubicamos el tag **<li>** con **id="categorias"**. A continuación del id, añadimos el atributo **onclick**, tal como lo muestra el siguiente código:

96% 23:23


```
onclick="retornar();"
```

La línea completa debe quedar conformada como se muestra a continuación:

```
<li id="btncategorias" onclick="retornar();"><a href="#"></a><i
class="material-icons black-text">chevron_left</i></a></li>
```

El botón ya cuenta con la funcionalidad de volver hacia la página anterior, gracias a la función **retornar()**, sin importarnos cuál sea el nombre de la página. Esta función es completamente autónoma y dinámica, por lo cual podremos aprovecharla en cada rincón de nuestros proyectos donde sea necesario establecer una navegación hacia el nivel anterior.

Hipervínculo hacia el producto

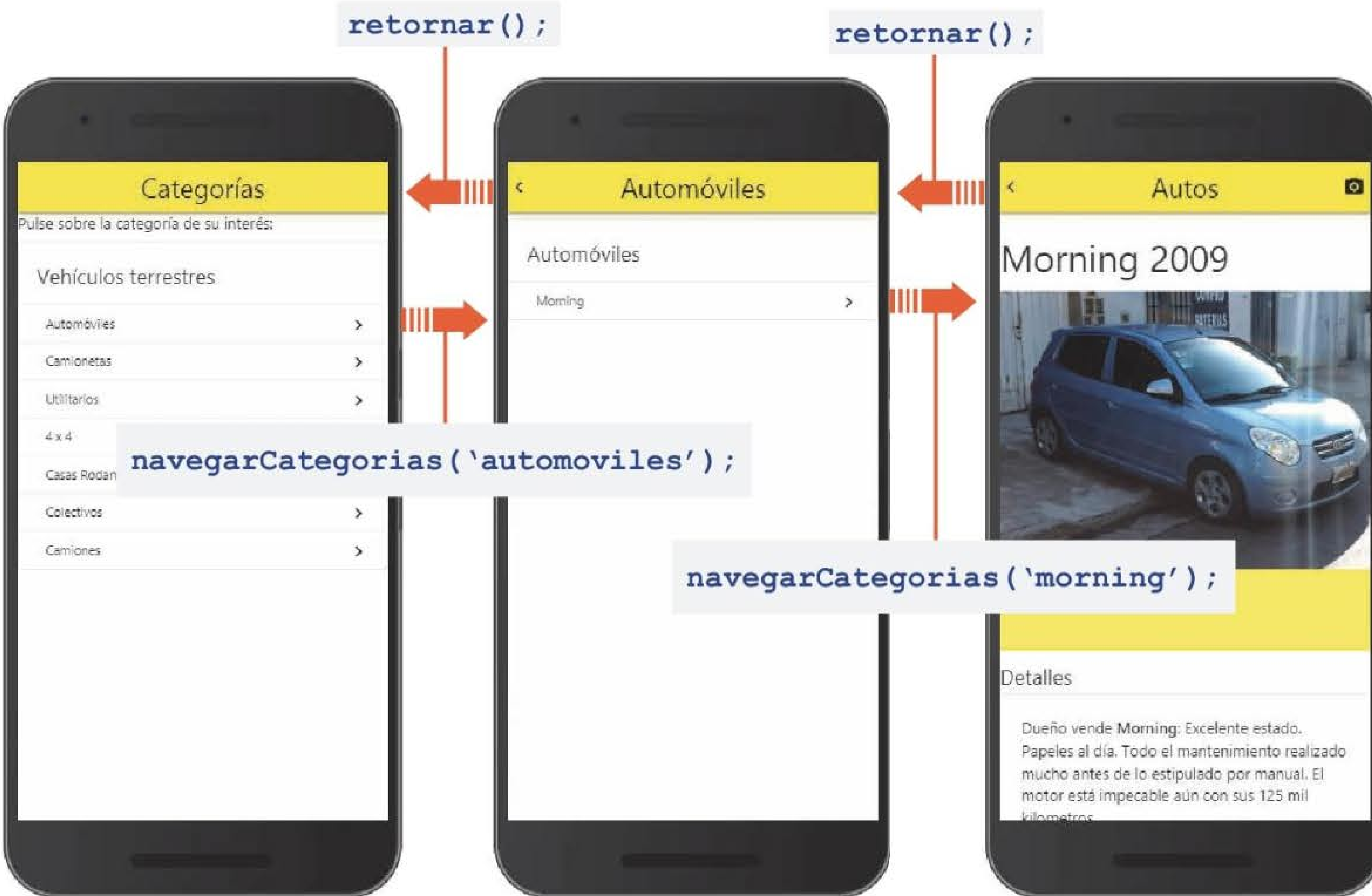
Solo nos resta agregar el hipervínculo hacia el producto final. Dentro de este mismo documento ubicamos el tag **<li>** de la colección, llamado **Morning**, y le agregamos el atributo **onclick**, como muestra el siguiente código:

```
onclick="navegarCategorias('morning');"
```

La conformación final de la línea debe quedar de la siguiente manera:

```
<li class="collection-item" onclick="navegarCategorias('morning');"><d
iv>Morning<a class="secondary-content"><i class="material-icons black-
text">chevron_right</i></a></div></li>
```

Por último, solo nos resta agregar la función **retornar()** en **morning.html**. Con esto conseguiremos la navegación funcional del sitio entre estas tres páginas. En el archivo **Ejercicio Collections - Resuelto.zip** del material que acompaña a esta obra, se encuentra el ejercicio completamente funcional.



Colección activa

Si agregamos colecciones con hipervínculos y estas deben mostrarse en una misma página, podemos establecer como activo el elemento **collection-item** sobre el cual hicimos clic o tap. Para esto, simplemente le agregamos la clase **active**, que resaltará su color por sobre el resto:

```
<li class="collection-item active"…>
```

Avatares

Contamos también con colecciones que permiten incorporar imágenes ilustrativas agregando la clase **avatar**. Veamos un ejemplo de código:

```
<li class="collection-item avatar" id="automoviles" onclick="navegarCat
egorias('automoviles');">
    <img src="images/car.png" class="circle">
    <span class="title">Automóviles</span>
    <p>1 categoría disponible</p>
    <a href="#!" class="secondary-content"><i class="material-icons
black-text">chevron_right</i></a></div>
</li>
```

Expandimos nuestro **collection-item ** agregando un tag **<img>** con la ruta a la imagen, un tag **<span>** que oficia como título utilizando la clase **title**, y un tag **<p>** para añadir una mínima descripción.

Badges

La clase **badge** es utilizada comúnmente dentro de las colecciones para distinguir información relacionada al ítem específico donde se aplica. En el caso de las colecciones, podemos aplicar un badge en una categoría específica para destacar, por ejemplo, cuántos elementos nuevos aparecieron en dicha categoría.

Vamos a modificar nuestro ejercicio anterior eliminando el icono **chevron_right** del **collection-item automóviles**. A continuación agregamos la siguiente línea de código:

96% 23:23


```
<span class="badge ">3</span>
```

Utilizando un tag **<span>**, esto reemplaza el icono anteriormente asignado a la categoría, por un número que señala una hipotética cantidad. Agregándole a <span> la clase **badge**, conseguimos este resultado, pero como dicho número queda poco claro, agregamos dos clases más junto a badge: **green y black-text**. Dicha modificación debe quedar como muestra la siguiente línea de código:

```
<li class="collection-item" id="automoviles" onclick="navegarCate
gorias('automoviles');"><span class="badge green black-text">3</
span><div>Automóviles</div></li>
```

La clase **green** genera un color de fondo verde sobre el badge agregado, y la clase **black-text** cambia el color del número de este badge. De esta forma, logramos un mejor contraste para su correcta visualización. Por supuesto que podemos buscar el mejor contraste de colores que se ajuste a nuestro proyecto, o crearlos nosotros mismos desde una hoja de estilos CSS.

Por último, agregando la clase **new** dentro del atributo **class** de la etiqueta **span**, el badge mostrará la palabra **new** (nuevo, en inglés), junto al número o texto indicado al badge. Esto es opcional por supuesto. Si realizamos proyectos solo en español, podremos crear nuestro propio CSS para reflejar dicha palabra directamente en nuestra propia lengua.

96% 23:23

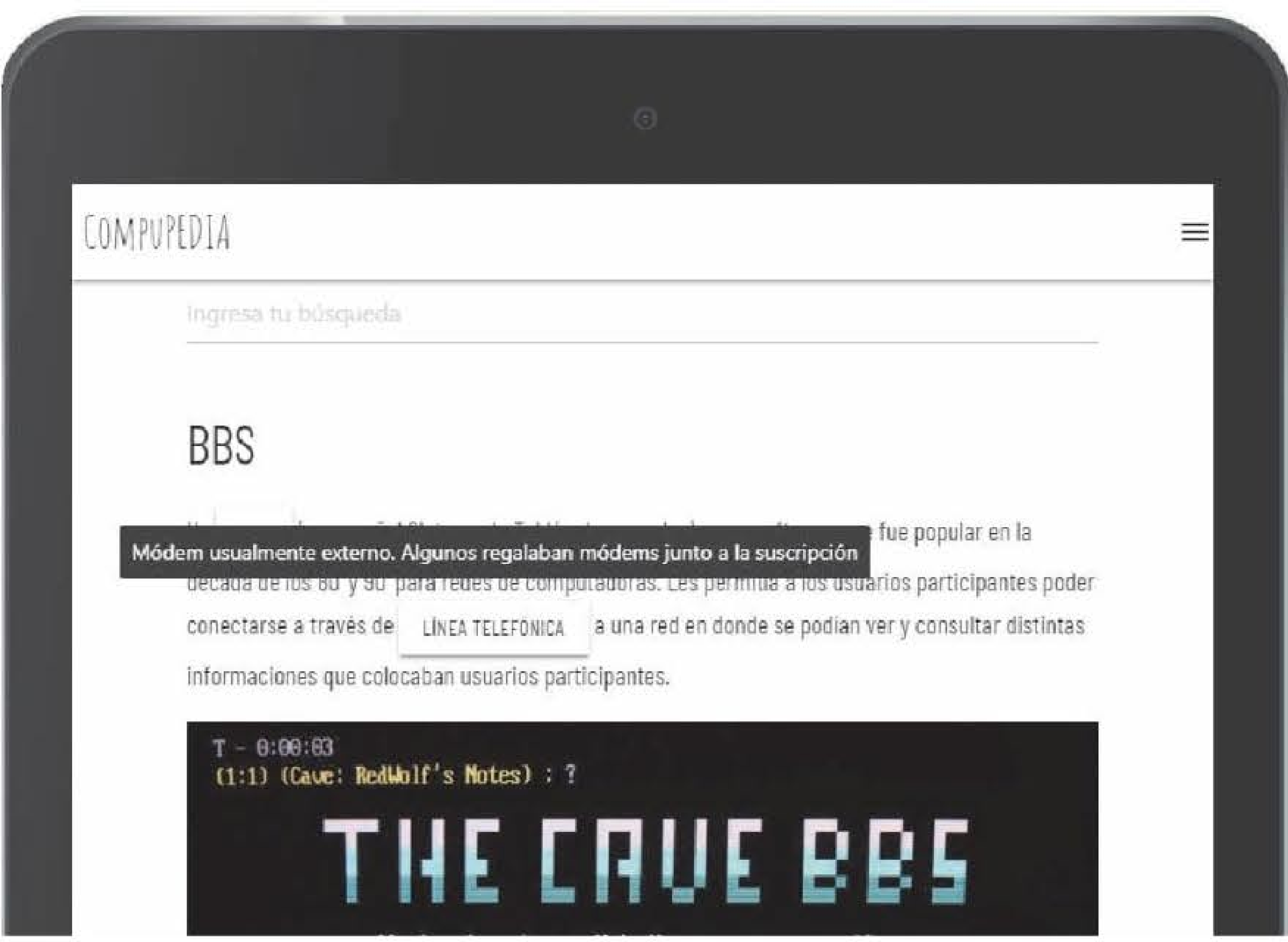
Ejemplo del uso de la clase badge

En el material que acompaña a esta obra, es posible consultar el archivo **Ejemplo Badges.zip**, que contiene una muestra de cómo se implementa y de qué manera funciona la clase **badge**.

Tooltip

Veamos a continuación cómo aprovechar la clase/componente **Tooltip**, ideada originalmente para el terreno de desarrollo desktop pero, aun así, muy fácil de aprovechar en el campo móvil.

En Materialize, una plataforma ideada para el ecosistema mobile pero también muy utilizada en el ámbito de escritorio, se incluye un componente denominado **Tooltip**, que permite desplegar un mensaje asociado a un botón, icono o texto dentro de una aplicación web/web mobile.



Si bien podemos pensar que Tooltip se utiliza en el ambiente de escritorio para mostrar un mensaje determinado al posicionar el mouse sobre un componente HTML o una palabra, se adapta muy bien al escritorio, donde podemos aprovecharlo para explayar información referente a algo que mencionamos, simplemente haciendo tap sobre un componente HTML.

Su uso es muy simple, y se divide en dos partes:

- Un **tag HTML** con una serie de atributos básicos.
- La **inicialización** de este componente desde **JavaScript**.

96% 23:23

Tag HTML

```
<a class=”btn tooltipped white black-text” data-position=”top” data-
tooltip=”Información a mostrar en el tooltip”>Palabra o frase a resal-
tar</a>
```

Inicialización JS

La inicialización JavaScript tiene asociada la declaración de determinadas propiedades que darán vida a su comportamiento. Las más importantes son:

TABLA 7

Propiedad	Valor	Descripción
enterDelay	**Número**	Tiempo de retardo antes de que el Tooltip aparezca.
exitDelay	**Número**	Tiempo de retardo antes de que el Tooltip desaparezca.
html	**Cadena de texto**	Formato de tags de este lenguaje para mostrar dentro del mensaje incluido en el Tooltip.
position	**Cadena de texto**	Posición en pantalla donde aparecerá el Tooltip: **‘top’, ‘right’, ‘left’, ‘bottom’**.

Estos parámetros se deben especificar del lado de JavaScript, para inicializar dicho componente con los efectos y las características necesarias que deseemos mostrar en pantalla cuando el elemento Tooltip es invocado.

Manos a la obra

Veamos con un ejercicio práctico la forma más fácil de implementar el componente Tooltip. Para esto, abrimos el archivo **Ejercicio Tooltip – Base.zip** alojado en el repositorio de archivos de esta obra.

Descomprimimos este archivo y abrimos un nuevo proyecto en **Visual Studio Code**. Si ejecutamos este ejemplo, veremos una página dedicada a un sistema imaginario símil **Wikipedia**, enfocado al mundo computacional.

Los sistemas basados en Wiki utilizan siempre hipervínculos para expandir el significado de una palabra o frase, y es lo que tomaremos como base para darle mayor interactividad utilizando el elemento Tooltip.

Lo primero que hacemos es agregar los tags HTML dentro del documento homónimo. Trabajamos el contenido de estos tags modificando una serie de palabras, que detallamos a continuación:

- **BBS**: la segunda palabra del primer párrafo será reemplazada por la siguiente línea de código:

```
<a class="btn tooltipped white black-text" data-position="top" data-tooltip="Bulletin Board System">BBS</a>
```

- **línea telefónica**: en el mismo primer párrafo, encontramos esta frase, que también será modificada para que despliegue información. La línea de código que la reemplazará es la siguiente:

```
<a class="btn tooltipped white black-text" data-position="top" data-tooltip="Usualmente eran Módems externos. Algunos ISP los regalaban con al pago de la suscripción anual de sus servicios.">línea telefónica</a>
```

- **telnet**: en el segundo párrafo figura esta palabra, que cambiamos por el siguiente código:

```
<a class="btn tooltipped white black-text" data-position="top" data-tooltip="Referencia al comando telnet; no al protocolo.">telnet</a>
```

- **correo**: por último, ubicamos esta palabra en este mismo segundo párrafo, y la reemplazamos por el siguiente código:

```
<a class="btn tooltipped white black-text" data-position="top" data-tooltip="Pocos BBSs ofrecían el servicio de e-mail, abierto al envío internacional de correo electrónico.">correo</a>
```

Inicialización JavaScript

Veamos ahora en el archivo JS el código necesario para inicializar el componente Tooltip con una serie de atributos que le darán su funcionalidad. Para esto, abrimos el archivo JS identificado en el apartado **<head>**, haciendo **Ctrl + clic** en él:

```
<script src="js/tooltip.js" defer></script>
```

En el código del archivo JS, encontramos la sentencia **document.addEventListener ("DOMContentLoaded"...)**, que hace que su contenido se ejecute una vez cargado el documento HTML. A continuación, encontramos la variable **elementos**, que se ocupa de seleccionar todos los componentes HTML con la clase **tooltiped**. La siguiente variable, **opciones**, se ocupa de almacenar las opciones que le dan los efectos de inicio y cierre del elemento Tooltip, y la ubicación de este en pantalla cuando es invocado:

```
var opciones = 'enterDelay: 200, exitDelay: 100; position: top';
```

Por último, la línea correspondiente a la variable **instancia** se ocupará de dejar al componente funcional:

```
var instancia = M.Tooltip.init(elementos, opciones);
```

Con esto, ya podemos cargar el proyecto en el simulador de Google Chrome, comenzar a navegar y probarlo en las diferentes opciones de simuladores, para ver cómo se comporta en cada uno, mostrando la información a través del componente Tooltip.

Si no queremos darle un marco tan profundo en cada hipervínculo Tooltip, podemos eliminar la clase **btn** del atributo **class** de cada Tooltip utilizado pero, a su vez, agregamos alguna clase CSS relacionada al color para que se destaque la palabra indicando que hay información adicional, como **yellow**.

Toast: mensajes temporales

Los mensajes del tipo Toast abundan en las aplicaciones móviles y web mobile, ya que son concisos y casi no ocupan espacio en pantalla. Veamos lo que nos propone Materialize para integrar este tipo de notificaciones en nuestros desarrollos móviles.

Esta clase de alertas se basan en la interacción con el usuario a través de mensajes del tipo "no invasivos", ya que son muy reducidos en cuanto al espacio en pantalla y, a su vez, son temporales (aparecen y desaparecen sin intervención del usuario). A diferencia de otros componentes Materialize, Toast no es un elemento gráfico que debemos crear desde HTML, sino que nació directamente como un componente JS que se ocupa de recrear, a través de un método, la funcionalidad y el comportamiento de este.

Propiedades personalizables

Toast cuenta con una serie de propiedades que podemos manipular para que su comportamiento se ajuste a nuestras necesidades. Veamos a continuación cuáles son:

TABLA 8

Nombre	Tipo de dato	Descripción
html	**Cadena de texto**	Permite especificar una cadena de texto HTML para desplegar dentro del mensaje.
displayLenght	**Número**	Tiempo en milisegundos de la existencia de Toast en pantalla, antes de desvanecerse.
inDuration	**Número**	Tiempo en milisegundos que demora la transición de Toast en aparecer en pantalla.
outDuration	**Número**	Tiempo en milisegundos que demora la transición de Toast en desaparecer de la pantalla.
classes	**Cadena de texto**	Permite agregar una o más clases existentes en Materialize o de nuestro CSS, para estilizar el mensaje a mostrar.

Modo de invocarlo

Desde un **<script>** o un documento JS, debemos escribir el apartado correspondiente para que Toast sea desplegado en pantalla. Su sintaxis es la siguiente:

```
M.toast(mensaje, opciones);
```

96% 23:23

Invocamos el objeto **M** correspondiente a Materialize. De este, el método **toast()**. **mensaje** corresponde a una variable previamente creada, la cual engloba el mensaje de texto que se mostrará dentro de Toast; y **opciones** corresponde a todas las propiedades junto con sus valores, que darán forma a la animación, tiempos y estilo que Toast tendrá cuando sea invocado. Para entender mejor su implementación, vayamos a un ejemplo práctico.

Ejemplo práctico: menú de Compupedia

Abramos el último proyecto realizado, **Compupedia**, en el cual integramos los mensajes **Tooltip**. En él tenemos un menú ubicado a la derecha de la barra superior.

Como dicho menú no tiene interacción, agregaremos un mensaje tipo Toast que indique que no está disponible. Si miramos el código HTML del elemento HTML menu, veremos que el botón tiene como atributo **ID** el nombre **nav-mobile**.

Editemos a continuación el documento JS de este proyecto. Dentro del apartado **AddEventListener**, justo al final del código correspondiente a Tooltip, agregamos lo siguiente:

```
var bm = document.getElementById('nav-mobile');
```

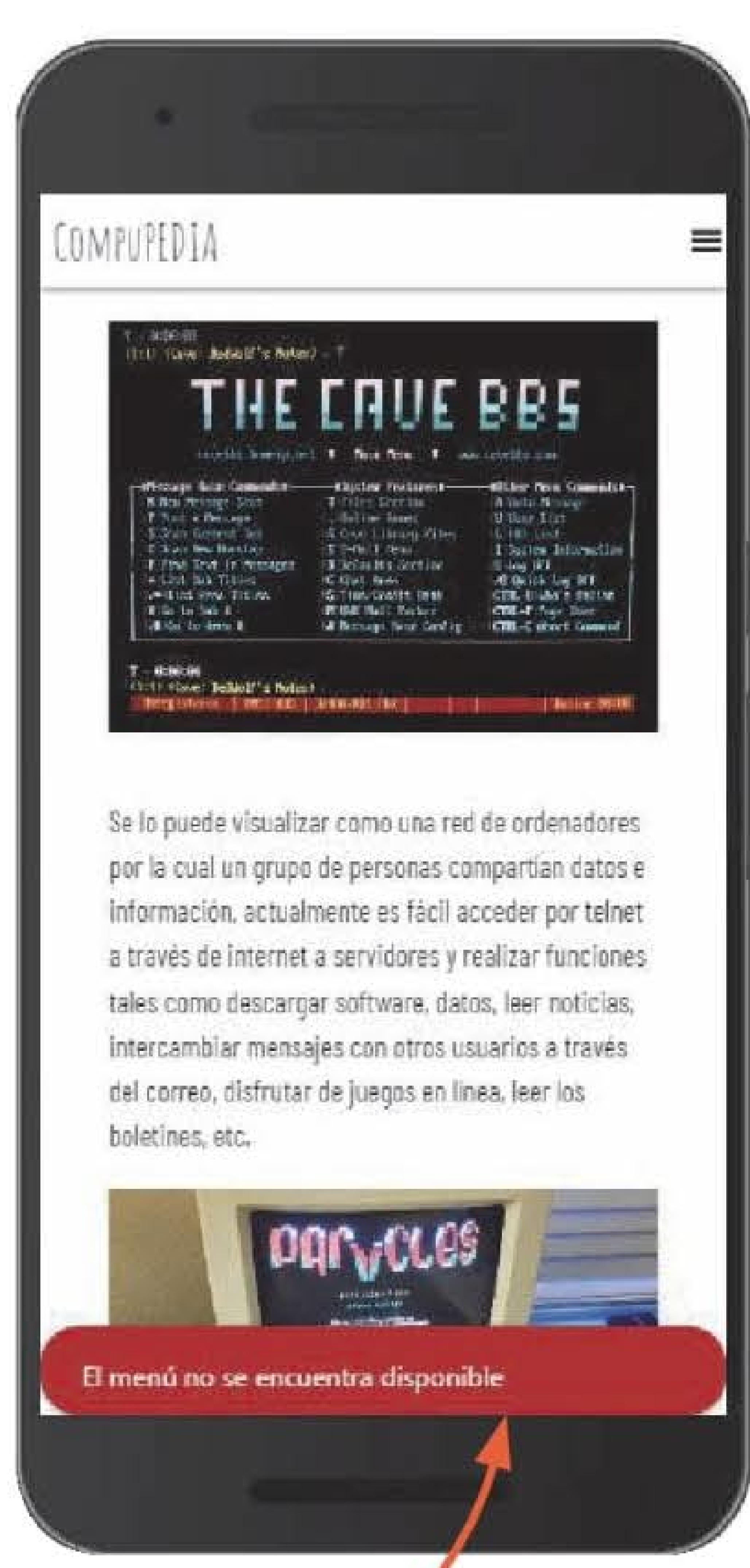
Relacionamos en la variable **bm** el elemento HTML correspondiente al botón Menú. Ahora vamos a ponerle funcionalidad al evento **clic** o **tap** de dicho botón:

```
        bm.addEventListener('click', function() {

        })
```

Dentro de este **addEventListener**, agregamos la funcionalidad de Toast:

```
var msj = '<h6>El menú no se encuentra disponible</h6>';
M.toast({html: msj, displayLength: 2000, outDuration: 1000, classes:
'rounded red darken-4'});
```

Creamos la variable **msj**, a la cual le asignamos el texto a mostrar por Toast, con tags HTML incluidos. Luego invocamos el método **toast()** del objeto **M**, pasándole como parámetros el HTML del texto, que dure 2 segundos en pantalla, y que cuando desaparezca tenga una transición de 1 segundo. Finalmente, le indicamos bordes redondeados y un color cuasi bordó.



Como vemos en la figura anterior, el mensaje Toast aparece al pie de página del dispositivo móvil. Si queremos cambiar su ubicación en pantalla, con un poco de CSS podremos relocalizarlo donde deseemos:

```
#toast-container {
    min-width: 10%;
    top: 50%;
    right: 50%;
    transform: translateX(50%) translateY(50%);
}
```

En el repositorio de archivos de esta obra, bajo el nombre **Ejercicio Tooltip y Toast - Resuelto.zip**, está el proyecto con el elemento Toast integrado, para su consulta y pruebas adicionales. En el apartado dedicado a este elemento, en la web oficial de Materialize, encontraremos otras funcionalidades para controlarlo y agregarle otras opciones y comportamientos que lo potencien aún más.

Ventanas modales

Las ventanas modales son el elemento esencial para no tener que salir de un documento HTML hacia otro, ni abrir una nueva pestaña para interacción del usuario. Veremos a continuación qué nos propone Materialize para este terreno.

El uso de **Modal Dialogs** nos permite, por ejemplo, crear mensajes personalizados de diálogos, confirmación e ingreso de texto, evitando así las limitaciones de las funciones JS **alert()**, **confirm()** y **prompt()**, respectivamente. También su uso se puede extender a la creación de formularios de contacto o, como en el caso que realizaremos a continuación, la apertura de una ventana modal para el ingreso de datos a modo de comentarios.

Para comenzar la práctica, abrimos el proyecto **Modal Dialog – base.zip**, que se encuentra junto al material que acompaña a esta obra. Con este ya descomprimido en nuestro equipo, abrimos dicha carpeta como proyecto de VS Code. Nos encontraremos con el proyecto creado en el apartado **Colecciones** de este mismo capítulo, donde incluimos unas ligeras modificaciones.

Ejercicio práctico: ventana de comentarios

Para entender Modal Dialog a partir de la práctica, abrimos una ventana modal que permitirá al usuario escribir una pregunta, tal como si estuviésemos consultando al vendedor de dicho producto.

El objetivo de esto será entender cómo se comporta Modal Dialog y, a su vez, extender la funcionalidad de dicha ventana controlando, por ejemplo, la cantidad de caracteres escritos por el usuario, y habilitando y deshabilitando el botón de Enviar, si la extensión de lo escrito excede el límite establecido.

De esta manera, no solo comprenderemos cómo se comporta este componente Materialize, sino que también afianzaremos aún más nuestra base de conocimiento de JavaScript.

Estructura HTML de Modal Dialog

Veamos en una primera instancia cómo se utiliza la parte HTML de Modal Dialog. Editamos el archivo **morning.html** y nos posicionamos al final de él. Luego de las etiquetas **<ul>** que le daban vida a los elementos **Collection**, encontraremos una etiqueta **<div>** que contiene un botón del tipo **Floating**. En este hay un icono de mensaje, que será el disparador de la ventana modal. Dentro del código HTML del botón, encontraremos el atributo **data-target**, el cual relaciona directamente al botón como disparador de la ventana Modal.

```
<div class="center">
    <a class="btn-floating z-depth-4 waves-effect waves-light yellow black-text
    modal-trigger" href="#" id="btnPreguntar" data-target="modalMsj"><i
    class="material-icons black-text">message</i></a>
</div>
<br>
```

Modal Dialog tiene su componente HTML que, a pesar de estar escrito dentro de este documento de marcado, en un principio no se ve. Analicemos a continuación una infografía para entender dicho apartado de código:

Estructura JS de Modal Dialog

Ahora trabajaremos con el apartado JavaScript que le da la funcionalidad a nuestro proyecto. Para esto, en la sección **<head>** del documento HTML, ubicamos la siguiente línea de código:

96% 23:23


```
<script defer src="js/preguntas.js"></script>
```

Esta línea corresponde al archivo **preguntas.js** que no existe, y es donde escribiremos el código JS funcional. Pulsamos **Ctrl + clic** sobre él, para crearlo. Ya dentro de este, definimos una serie de variables con las cuales controlaremos el comportamiento de los botones y de la ventana modal.

```
var bp = document.getElementById("btnPreguntar");
var tc = document.getElementById("textoConsulta");
var ep = document.getElementById("btnEnviarPregunta");
var msj = document.getElementById("mensajeExcedido");
var caracteres;
```

La primera variable, **bp**, se vincula directamente con el componente HTML **btnPreguntar**; es la que abre el cuadro de diálogo modal. La segunda variable, **tc**, se vincula con el componente **textArea**, cuyo ID es **textoConsulta**, que es donde se ingresará la pregunta. La tercera variable, **ep**, se relaciona con el componente HTML **btnEnviarPregunta**, botón que cerrará el formulario modal.

La cuarta variable, **msj**, se relaciona directamente con el componente HTML **<span>**, donde se visualizará un mensaje por el exceso de caracteres escritos en el componente **textArea**. Finalmente, la variable **caracteres** registrará el total de caracteres escritos en **textArea**, para determinar si se muestra o no el mensaje de exceso de caracteres en el tag **<span>**.

Abrir la ventana Modal

Como dijimos antes, el componente HTML modal existe ya dentro del documento, pero no se visualiza por defecto. Para poder verlo, lo inicializaremos desde JavaScript, escuchando cuándo ocurre el evento **click** sobre el botón **btnPreguntar**, para mostrar el Modal Dialog:

```
bp.addEventListener("click", function() {
    ...
})
```

Dentro de él escribimos el código de inicialización del componente Modal Dialog. La inicialización se realiza de la misma manera en que vimos cómo inicializar otros componentes Materialize, por ejemplo, **Collections**. Modal Dialog cuenta con una serie de propiedades que se deben pasar a

dicho componente HTML para que este se inicialice y aplique los efectos determinados, como así también para que maneje los tiempos de transición entre que se muestra y se oculta. Dichas propiedades son:

TABLA 9

Nombre	Tipo de dato	Descripción
preventScrolling	**Verdadero/falso**	Previene que el formulario modal pierda su visualización principal por un evento del tipo Scroll, no controlado.
inDuration	**Numérico**	Especifica en milisegundos lo que tardará el formulario modal en mostrarse en pantalla.
outDuration	**Numérico**	Especifica en milisegundos lo que tardará el formulario modal en salir de pantalla.
startingTop	**Porcentaje**	Permite establecer la posición inicial de dicho formulario en pantalla, pudiendo especificar si debe aparecer pegado al borde superior, medio o inferior de la misma, mediante un porcentaje.

Veamos a continuación cómo utilizar dichas propiedades. Creamos primero una variable denominada **elemModal**, que se relacionará con todos los elementos Modal Dialog existentes en este documento HTML, mediante el método **querySelectorAll()**. Acto seguido, declaramos la variable **opciones**, donde agregaremos las propiedades de inicialización de la ventana modal:

```
var elemModal = document.querySelectorAll('.modal');
var opciones = 'preventScrolling: true, inDuration: 500, outDuration:
300, startingTop: 50%';
```

Inicializamos la variable a través de la siguiente sentencia:

```
M.Modal.init(elemModal, opciones);
```

Y finalmente, le damos el foco al elemento HTML **textArea**, para que el usuario pueda comenzar a escribir:

```
tc.focus();
```

Veamos en la siguiente imagen, cómo debe quedar estructurado el código:

```
js > JS preguntas.js > ...
1    var bp = document.getElementById("btnPreguntar");
2    var tc = document.getElementById("textoConsulta");
3    var ep = document.getElementById("btnEnviarPregunta");
4    var msj = document.getElementById("mensajeExcedido");
5    var caracteres;
6
7    bp.addEventListener("click", function() {
8        var elemModal = document.querySelectorAll('.modal');
9        var opciones = 'preventScrolling: true, inDuration: 500, outDuration: 300, startingTop:
10       M.Modal.init(elemModal, opciones);
11       tc.focus();
12   })
```

Nuestro formulario modal ya debe funcionar, tanto en su apertura como cierre. Ejecutemos el proyecto hasta aquí escrito en el simulador de Google Chrome, para probarlo. Probemos a abrir el formulario para escribir contenido en **textArea**.

Control de caracteres

Añadiremos ahora un contador de caracteres, que controlará cada carácter tipiado en la caja de texto **textArea**. A través de la propiedad JS **textLenght** de dicho componente, guardaremos en la variable **caracteres** el total escrito. Luego, a través de un **if()** comparamos si la variable caracteres superó el valor 200.

96% 23:24

En este caso, en el componente **<span>** creado junto con Modal Dialog, escribiremos el mensaje de exceso de caracteres, y deshabilitamos el botón de enviar mensaje. En caso contrario, borramos el mensaje de exceso de caracteres y volvemos a habilitar el botón:

```
tc.addEventListener("input", function() {
    caracteres = tc.textLength;
    if (caracteres > 200) {
        msj.innerHTML = 'Excede caracteres: <strong>' + caracteres +
'<strong>';
        ep.classList.add("disabled");
    } else {
        msj.textContent = '';
        ep.classList.remove("disabled");
    }
})
```

La estructura de este código debe ser la siguiente:

```
morning.html    JS preguntas.js ×
js > JS preguntas.js > tc.addEventListener("input") callback
20
21  tc.addEventListener("input", function() {
22      caracteres = tc.textLength;
23      if (caracteres > 200) {
24          msj.innerHTML = 'Excede caracteres: <strong>' + caracteres + '<strong>';
25          ep.classList.add("disabled");
26      } else {
27          msj.textContent = '';
28          ep.classList.remove("disabled");
29      }
```

Y el resultado es el que muestra la imagen de la próxima página.

La propiedad **innerHTML** de la variable **msj** nos permite escribir el contenido HTML formateado con negrita. La propiedad **classList** y su método **add()** permiten agregar una clase a determinado componente HTML, y el método **remove()**, eliminarla.

Finalmente, el evento **input** dispara la función que contiene todo este código cuando se ingresa o elimina uno o más caracteres, del elemento **textArea**.

En el archivo **Modal Dialog – resuelto.zip**, se encuentra este ejercicio finalizado, para poder comparar el código.

05 Integración de datos remotos

JSoN es la opción más adecuada para obtener y consumir datos remotos, ya que evita impactar de manera directa contra una base de datos. Veamos cuáles son sus características y cómo podemos integrarlo fácilmente dentro de nuestros desarrollos móviles.

Desde el nacimiento de las aplicaciones que consumen datos remotos, a lo largo de la historia de Internet se propuso una diversidad importante de consumo de información vía web, así como también desde aplicaciones móviles. Lo cierto es que **JSoN**, junto a **XML**, son las dos opciones viables que nos permiten obtener y grabar datos de manera remota, con una interfaz sencilla y la seguridad que otras alternativas web o móviles no brindan de modo directo.

La técnica Ajax

Este nombre proviene de **Asynchronous JavaScript and XML**, y se trata de una técnica con la que podemos crear aplicaciones interactivas de diversa índole. Ajax se ejecuta del lado del cliente, y consume datos remotos a los que se accede mediante una comunicación asincrónica realizada contra el servidor que los aloja.

Ejemplo de código JSoN

Veamos un ejemplo de código JSoN que nos permite generar un menú anidado:

```
var JSON = {
    menu: [
        {name: 'Croacia', link: '0', sub: null
        },
        {name: 'Inglaterra', link: '1', sub: [
            {name: 'Arsenal',link: '0-0', sub: null},
            {name: 'Liverpool',link: '0-1', sub: null},
            {name: 'Manchester United',link: '0-2', sub: null}
            ]},
```

```
            {name: 'España', link: '2', sub: [
                {name: 'Barcelona',link: '2-0', sub: null},
                {name: 'Real Madrid',link: '2-1', sub: null}
            ]},
            {name: 'Alemania', link: '3',sub: [
                {name: 'Bayern Munich',link: '3-1', sub: null},
                {name: 'Borrusia Dortmund',link: '3-2', sub: null}
                ]}
        ]
    }
```

```
1  var JSON = {
2      menu: [
3          {name: 'Croacia', link: '0', sub: null
4          },
5          {name: 'Inglaterra', link: '1', sub: [
6              {name: 'Arsenal',link: '0-0', sub: null},
7              {name: 'Liverpool',link: '0-1', sub: null},
8              {name: 'Manchester United',link: '0-2', sub: null}
9              ]},
10         {name: 'España', link: '2', sub: [
11             {name: 'Barcelona',link: '2-0', sub: null},
12             {name: 'Real Madrid',link: '2-1', sub: null}
13         ]},
14         {name: 'Alemania', link: '3',sub: [
15             {name: 'Bayern Munich',link: '3-1', sub: null},
16             {name: 'Borrusia Dortmund',link: '3-2', sub: null}
17             ]}
18     ]
19 }
```

- Croacia
- Inglaterra
  - Arsenal
  - Liverpool
  - Manchester United
- España
  - Barcelona
  - Real Madrid
- Alemania
  - Bayern Munich
  - Borrusia Dortmund

Segmentos de aplicación

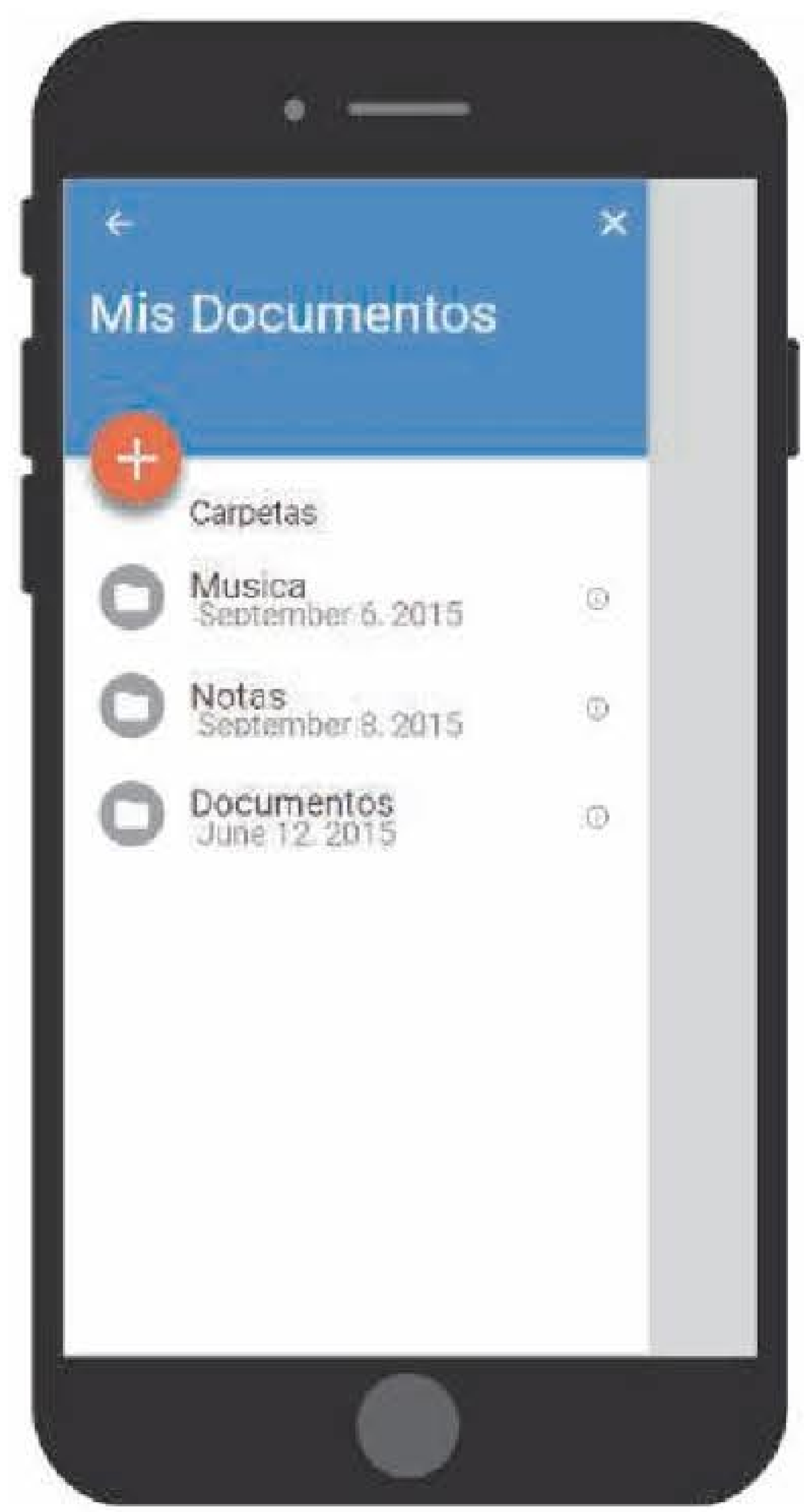
El uso de JSoN está destinado no solo a implementar datos remotos provenientes de bases de datos, sino también a armar la estructura de cualquier sitio basado en la Web o aplicación móvil, de manera simple y rápida.

Más allá de que nos permite obtener información de una base de datos, también podemos encontrar opciones para la creación de un menú dinámico, o para actualizar noticias y datos en pantalla recurriendo al cambio de ítems e hipervínculos dentro del código JSoN, sin necesidad de modificar el código de la aplicación. En los casos de apps que se instalan desde un store, podremos actualizar los datos que visualiza nuestra aplicación, cuando el usuario cierra la misma y vuelve a ingresar.

Servicios web

Esta técnica de acceso a datos remotos es también conocida como **web services**. Mientras que Ajax permite acceder a datos solo de manera sincrónica, dentro de un mismo servidor web de donde proviene la aplicación, el uso de web services nos da la posibilidad de ir más allá en un ecosistema dinámico, al traer datos remotos a nuestro dispositivo móvil, actualizados de forma asincrónica.

```
sidebar-menu.json
{

    menu: [
        {name: 'Carpetas'},
        {name: 'Musica', icon-folder,sub:
                [icon-info, '20150906']},
        {name: 'Notas', icon-folder,sub:
                [icon-info, '20150908']},
        {name: 'Documentos', icon-folder,sub:
                [icon-info, '20150612']}
         ]
}
```



Para que el servicio web remoto devuelva datos en formato JSoN, debe tener soporte para la tecnología **CORS** (acrónimo de **Cross Origin Resource Sharing**). Los datos que vuelven a nuestra aplicación móvil o sitio web llegan en formato texto, y estructurados de tal modo que puedan ser leídos e interpretados por JavaScript.

La web **jsonapi.org** es uno de los mejores puntos de partida para comenzar a conocer JSoN a fondo, y sacar así el máximo partido de dicho lenguaje asincrónico.

JSON API Specification Extensions Recommendations Examples Implementations FAQ About v1.0 STABLE

{ json:api }

A SPECIFICATION FOR BUILDING APIS IN JSON

View the specification Contribute on GitHub

If you've ever argued with your team about the way your JSON responses should be formatted, JSON:API can be your anti-bikeshedding tool.

By following shared conventions, you can increase productivity, take advantage of generalized tooling, and focus on what matters: your application.

Clients built around JSON:API are able to take advantage of its features around efficiently caching responses, sometimes eliminating network requests entirely.

Here's an example response from a blog that implements JSON:API:

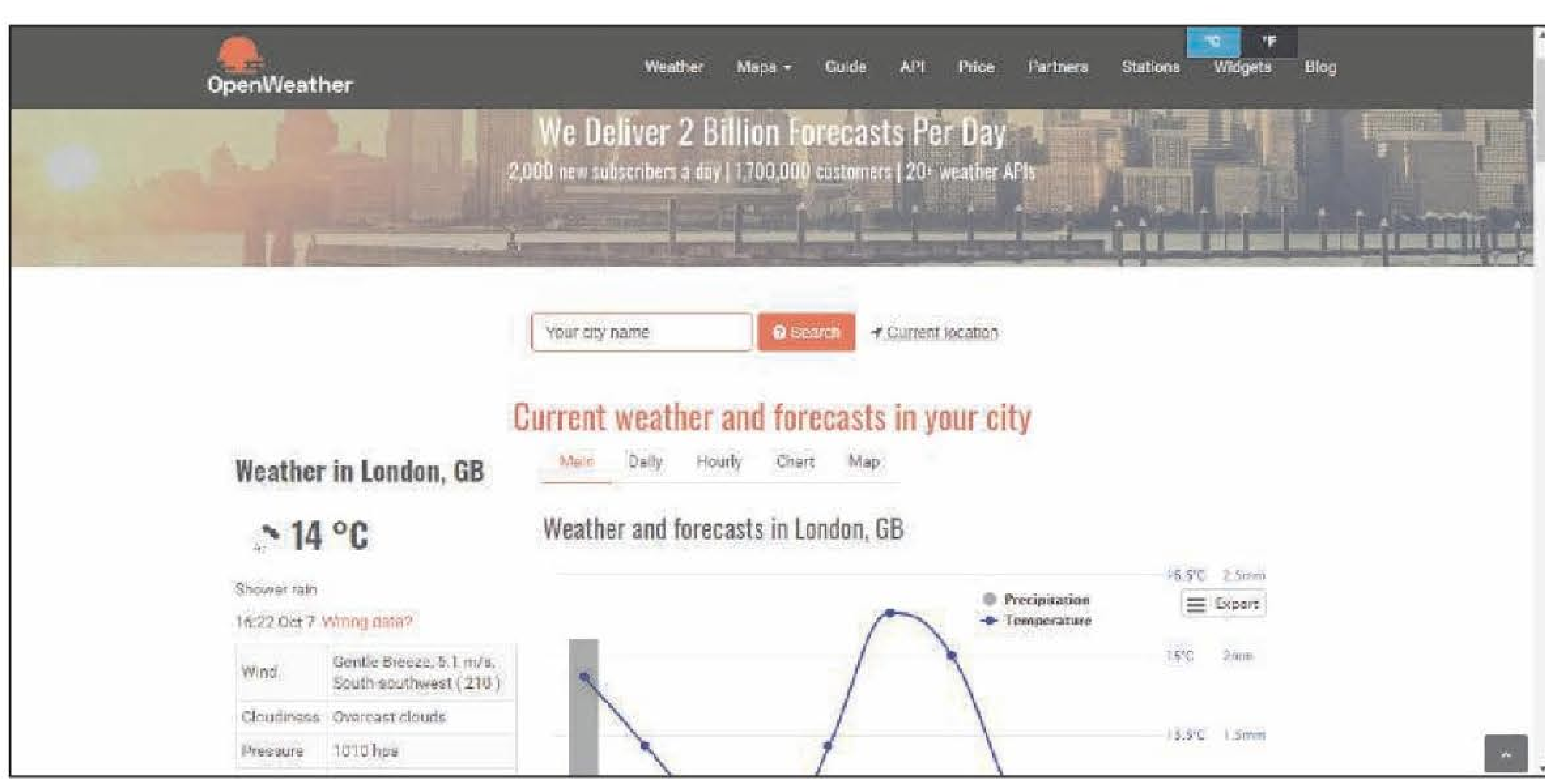
Diseño de una aplicación para el clima

La mejor forma de comprender el uso y la implementación de un sistema de datos remotos del tipo JSoN es mediante la práctica. Por eso, a continuación, vamos a desarrollar una aplicación para el clima, que obtendrá datos remotos reales y permitirá mostrar en pantalla la información en tiempo real de una determinada ciudad. Descarguemos para esto el proyecto **app-del-clima-base.zip** del repositorio de archivos que acompaña a esta obra.

Si cargamos el proyecto en el simulador de Google Chrome, nos encontraremos con un documento HTML estructurado tal como muestra la figura. A través de tags HTML, este documento tiene establecidos determinados párrafos que completaremos cuando obtengamos y decodifiquemos los datos de la API JSoN de un servicio de clima remoto.

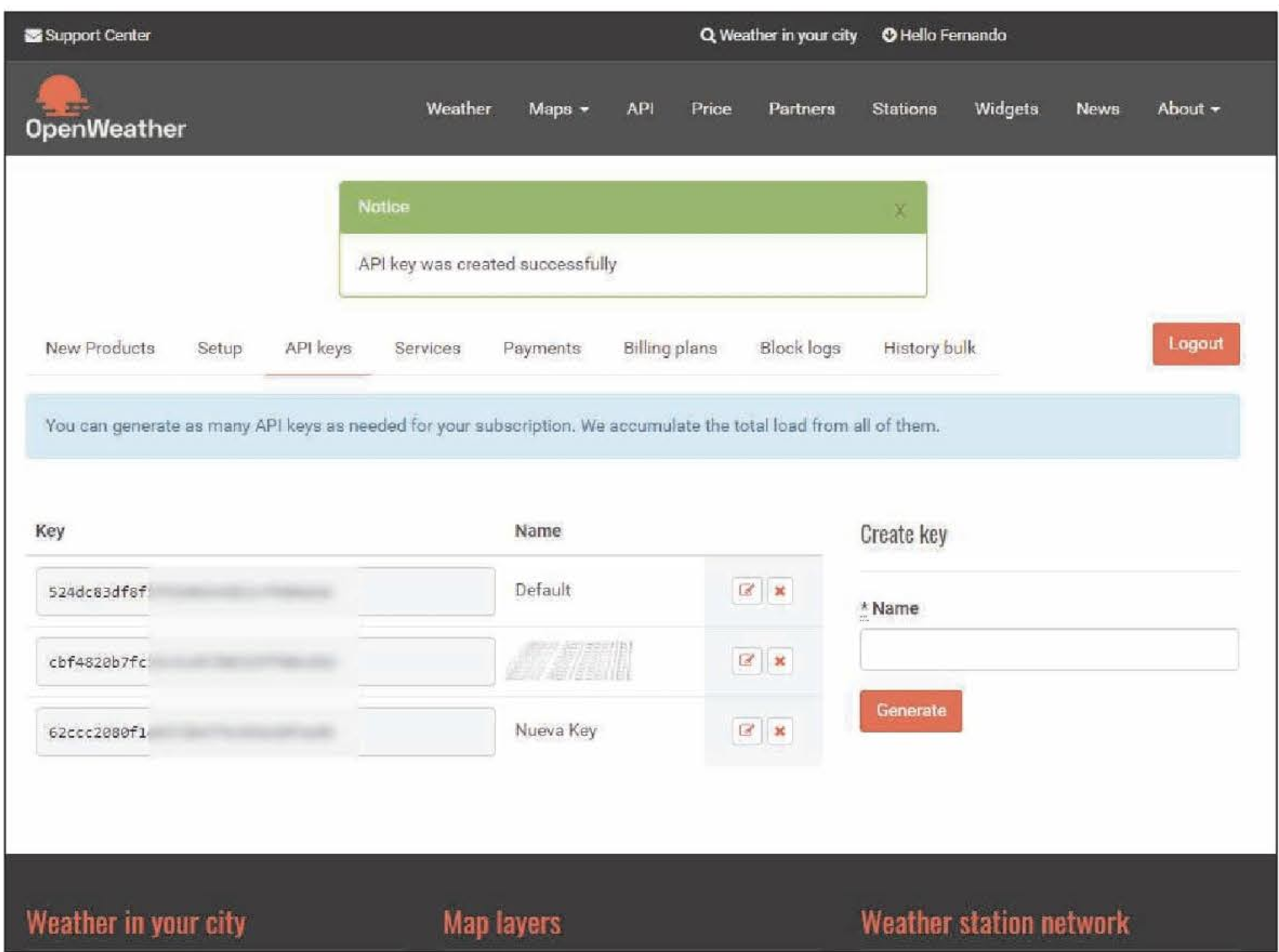
Open Weather Map

Para poder consumir datos remotos vía JSoN, debemos suscribirnos al servicio de **openweathermap.org**. Es uno de los más completos en Internet, y cuenta con una suscripción gratuita, que es la cual utilizaremos en este ejemplo. Luego de ingresar en la página, pulsamos en la opción **sign-in**, ubicada en el extremo superior.



Generar una API Key

Una vez suscriptos, iniciamos sesión y ubicamos el apartado de nuestro perfil de usuario. Acto seguido, presionamos en el apartado **API keys** y, luego, sobre el botón **Generate Key**, habiendo ingresado previamente un nombre que describa la API que estamos por generar. Creada la API, la encontraremos en el listado principal.



Abrimos el proyecto descargado en **Visual Studio Code**, ubicamos el apartado **<head>** del documento HTML y presionamos **Ctrl + clic** sobre el archivo **clima.js**. Como no existe, nos ofrecerá crearlo desde cero:

```
<script defer src=”js/clima.js”></script>
```

Agregaremos ahora el código inicial JS, que comenzará a darle vida a este proyecto:

```
const appKey = //”INGRESA_AQUI_TU_API_KEY”;
var ciudad = “Buenos Aires, AR”;
```

96% 23:24

AppKey es una constante JS; una vez que la creamos con un valor asignado, este no podrá modificarse desde ninguna otra parte del programa. Ingresamos en ella el código de API que nos proveyó **Open Weather Map** cuando la creamos. Debe ser similar al siguiente código: **cbf2480b2fc19c9c89008112ff00c65d**. La siguiente variable, **ciudad**, es la que almacena la ciudad desde donde obtendremos los datos climáticos; esta información será enviada al **web service** del clima como parámetro de filtro.

A continuación, incluyamos otras variables útiles en nuestra aplicación.:

```
var btnRecargar = document.getElementById("refrescar");
var pImagen = document.getElementById("imagen");
var lCiudad = document.getElementById("ciudad");
```

La variable **btnRecargar** se enlaza directamente con el botón refrescar, creado con el **ID refrescar** en el documento HTML; **pImagen** almacena el nombre de la imagen que ilustrará la información climática por consultar; y **lCiudad** guarda el nombre de la ciudad devuelto por la **API JSoN**. Esta es la forma inicial en que debe ir quedando estructurado el código dentro del documento JS:

```
JS clima.js ×    <> index.html
js > JS clima.js > ...
1   const appKey = "cbf4820b7fc"; //"INGRESA_AQUI_TU_API_KEY";
2   var ciudad = "Buenos Aires, AR";
3   var btnRecargar = document.getElementById("refrescar");
4   var pImagen = document.getElementById("imagen");
5   var lCiudad = document.getElementById("ciudad");
6
```

Mostrar información del clima en pantalla

A continuación, debemos definir unas variables que nos permitirán luego conectarnos con los diferentes tags HTML, para mostrar la información que nos devuelve la API del clima:

```
// ETIQUETAS PARA VISUALIZAR EL ESTADO DEL TIEMPO
var lTemp = document.getElementById("temperatura");
var lHume = document.getElementById("humedad");
var lPres = document.getElementById("presion");
var lVien = document.getElementById("viento");
```

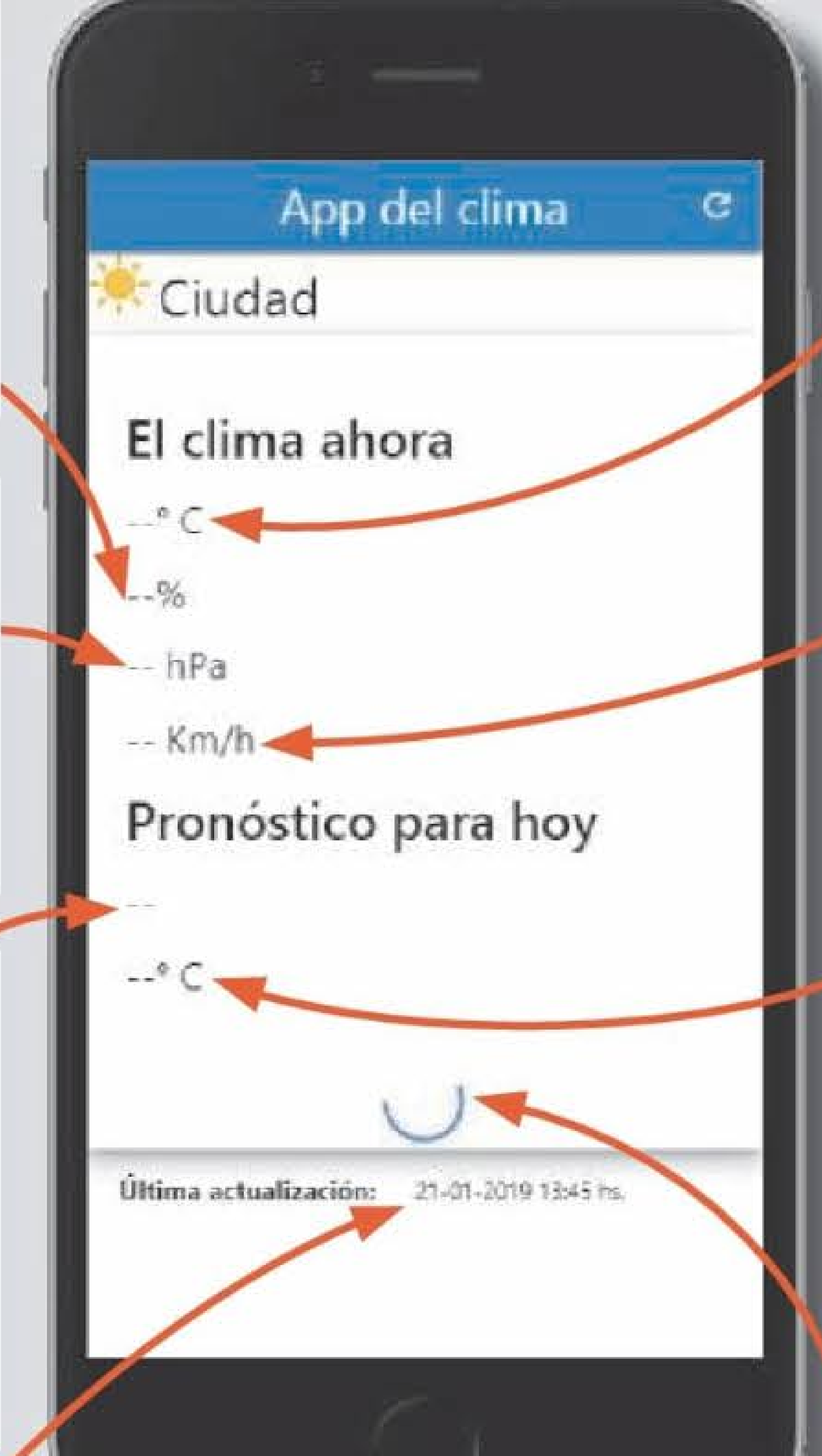
```
var lPron = document.getElementById("descripcion");
var lMima = document.getElementById("minimax");
var lUact = document.getElementById("actualizacion");
var Rueda = document.getElementById("rueda");
```

GUÍA VISUAL 8

lHume *es la variable JavaScript que mostrará información de humedad en el componente HTML* ***humedad****.*

lPres *es la variable JavaScript que mostrará información de presión en el componente HTML* ***presion****.*

lPron *es la variable JavaScript que mostrará una descripción general del pronóstico en el componente HTML* ***descripcion****.*



lTemp *es la variable JavaScript que mostrará información en el componente HTML* ***temperatura****.*

lVien *es la variable JavaScript que mostrará información del viento en el componente HTML* ***viento****.*

lMima *es la variable JavaScript que mostrará información general de las temperaturas mínimas y máximas en el componente HTML* ***minmax****.*

lUact *es la variable JavaScript que mostrará información sobre la última fecha y hora en que se actualizaron los datos del clima en el componente HTML* ***actualizacion****.*

Rueda *es la variable JavaScript que controlará la visualización o el ocultamiento de la rueda de progreso en el componente HTML* ***rueda****.*

Círculo progresivo de carga

Aprovecharemos el elemento Materialize CSS **Preloader**, que muestra un círculo de carga de datos, comúnmente visto en aplicaciones basadas en Material Design, para presentar dicho componente mientras se esperan los datos de

96% 23:24

parte del servidor. Dado que el componente **preloader** ya está creado en el documento HTML, vamos a controlarlo desde JavaScript, mostrándolo u ocultándolo a demanda. Para hacerlo, creamos la siguiente función:

```
function cargando(s) {
    switch(s) {
        case "a": Rueda.classList.add("active"); break;
        case "i": Rueda.classList.remove("active"); break;
    }
}
```

Esta función simple contiene un parámetro de ingreso determinado por la letra **s**. Cuando este parámetro tiene el valor **'a'**, JavaScript agrega en el componente **preloader** la clase **active**, que permitirá mostrarlo en pantalla. Por el contrario, si se llama a esta función con el valor del parámetro establecido en **'i'**, JavaScript quitará la clase active del componente Preloader para ocultarlo en pantalla. Esto último ocurrirá cuando los datos del clima hayan llegado a la app y se estén visualizando en pantalla.

```
function obtenerDatosDelClima() {
    cargando("a");
    if (ciudad.value === "") {
        alert("Debe ingresar una búsqueda válida.");
    } else {
        var URLowm = "https://api.openweathermap.org/data/2.5/
weather?q=" + encodeURI(ciudad) + "&appid=" + appKey +
"&units=metric&lang=ES";
        peticionHTTPAsincronica(URLowm, respuesta);
    }
}
```

La función **obtenerDatosDelClima()** ejecuta la API Open Weather Map enviando la ciudad que deseamos consultar, y la **AppKey** obtenida cuando nos registramos en el servicio.

La primera línea de código de esta función verifica que la variable JS **ciudad** tenga un valor ingresado. Si el valor es vacío (**''**), entonces mostrará una alerta para que se ingrese una búsqueda válida en dicha variable. Si no es vacío, entonces invoca la URL de consulta parseando en ella los valores de ciudad y AppKey correspondientes.

GUÍA VISUAL 9

*Concatenamos la variable **ciudad** dentro de la URL de consulta, utilizando el método JS **encodeURI()**. Dicho método se ocupa de convertir los espacios que pueda tener el valor almacenado en esta variable, en un formato apto para ser leído por JSoN.*

```
"https://api.openweathermap.org/data/2.5/weather?q="
                    + encodeURI(ciudad) + "&appid="
                    + appKey + "&units=metric&lang=ES";
```

*La variable **appKey**, definida como constante, nos identifica ante el servicio API de **Open Weather Map**, para finalmente brindarnos los datos solicitados.*

Por último, agregamos datos adicionales al string de consulta, de modo que la información devuelta por la API venga procesada bajo el sistema métrico y en idioma español.

Por último, invocamos a la función **peticionHTTPAsincronica()**, que recibe dos parámetros: **URLowm**, correspondiente a la variable de la API; y el método **respuesta()**, que se ocupa de armar la estructura HTML por mostrar en pantalla, una vez que los datos hayan llegado al dispositivo que los invocó. Vamos a crear a continuación la función de petición mencionada:

```
function peticionHTTPAsincronica(url, callback) {
    var req = new XMLHttpRequest();
    req.onreadystatechange = () => {
        if (req.readyState == 4 && req.status == 200) callback(req.res-
ponseText);
    }
    req.open("GET", url, true); //método asincrónico
    req.send();
}
```

Esta función crea la variable **req** del tipo objeto, la cual referencia un nuevo objeto **XMLHttpRequest()**. Cuando se ejecuta el evento **onreadystatechange**, JavaScript valida que el resultado del servidor sea igual a **4** y que su status sea igual a **200**. Esto indica que es óptimo para haber conseguido una respuesta del servidor del tipo satisfactoria.

Con estas condiciones validadas, ejecutamos el método **open()** parseando en él la **URL** y el método HTTP **Get**, que permitirá obtener los datos. Finalmente, invocamos el método **send()**, que enviará los datos al servidor. Esta función contiene un método de vuelta denominado **callback**.

Posibles estados de servidor

Veamos a continuación una tabla con los posibles estados de servidor que pueden volver cuando realizamos una consulta del tipo XMLHTTPrequest, a través del evento **readyState**.

TABLA 10

readyState		
Valor	**Estado**	**Descripción**
0	UNINITIALIZED	Todavía no se llamó a **open()**.
1	LOADING	Todavía no se llamó a **send()**.
2	LOADED	**send()** fue invocado. Encabezados y Estado se encuentran disponibles.
3	INTERACTIVE	Descargando; **responseText** contiene información parcial.
4	COMPLETED	La operación está terminada.

```
XMLHttpRequest.open(método, url, ¿asincrónico?)
XMLHttpRequest.send()
```

TABLA 11

Parámetros de open(...)	
Valor	**Descripción**
Método	GET o PUT
URL	Correspondiente al web service por consultar.
ASYNC	TRUE para indicar que la consulta se ejecutará de forma asincrónica.

TABLA 12

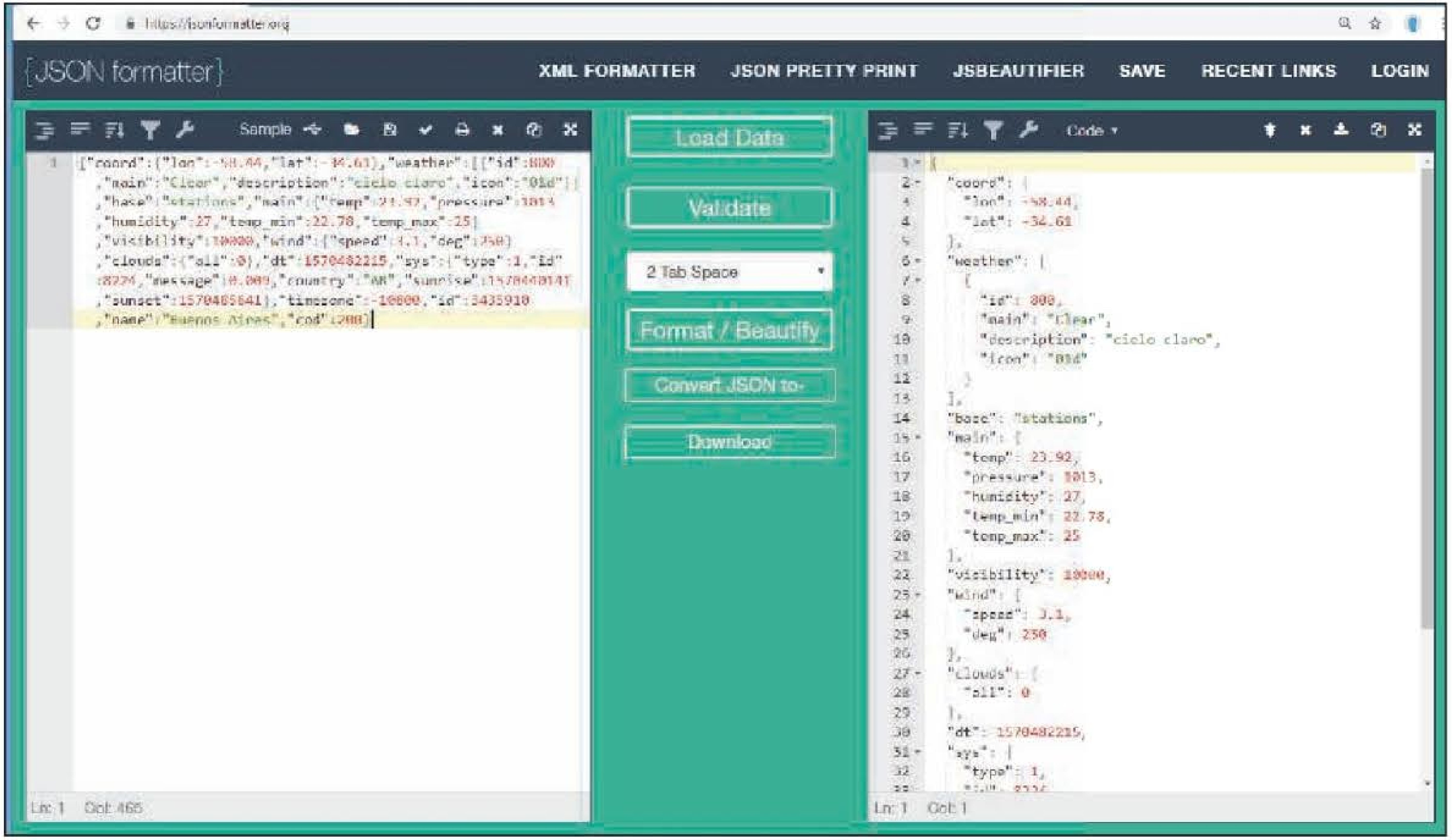
Status		
Valor	**Estado**	**Descripción**
200>	READY	responseText exitoso.
400>	ERROR	Error en la obtención de datos.

Cuando todo ha sido ejecutado exitosamente, **responseText** nos devuelve una cadena de datos **JSoN** con la información solicitada. Si pegamos la URL concatenada en una pestaña del navegador web, podremos ver el tipo de datos que obtenemos a través de **responseText**:


```
{"coord":{"lon":-58.44,"lat":-34.61},"weather":
[{"id":800,"main":"Clear","description":"cielo
claro","icon":"01d"}],"base":"stations","main":
{"temp":23.92,"pressure":1013,"humidity":27,"temp_min":22.78,"temp_max":25
},"visibility":10000,"wind":{"speed":3.1,"deg":250},"clouds":
{"all":0},"dt":1570482215,"sys":
{"type":1,"id":8224,"message":0.009,"country":"AR","sunrise":1570440141,"s
unset":1570485641},"timezone":-10800,"id":3435910,"name":"Buenos
Aires","cod":200}
```


JSoN Formatter

Como los datos devueltos a través del objeto **responseText** no son usualmente legibles, podemos copiarlos y pegarlos en un servicio como **JSON Formatter**, para así poder leerlos de manera más amigable: www.jsonformatter.org.



Como podemos apreciar, **JSON Formatter** estructura de modo vertical y anidado los datos devueltos por JSoN, los cuales se vuelven mucho más fáciles de leer, para luego poder armar el bloque de código JS que termine por mostrarlos en pantalla. Veamos el ejemplo del siguiente bloque de código:

96% 23:24


```
  "main": {
    "temp": 23.92,
    "pressure": 1013,
    "humidity": 27,
    "temp_min": 22.78,
    "temp_max": 25
  }
```

Los datos aquí presentes corresponden a la información del clima: temperatura, presión, humedad, y temperaturas mínima y máxima del día. Más abajo, en el apartado **wind**, obtendremos los valores del viento (velocidad y orientación) y, dentro del apartado **weather.description**, veremos la información general del día de la fecha.

Procesar la respuesta JSoN

Veamos a continuación el método que se ocupa de leer la información recibida por la cadena de datos JSoN y la mostrará en pantalla, seteando cada valor obtenido con su correspondiente tag HTML:

```
function respuesta(r) {
    let clima = JSON.parse(r);
    lCiudad.innerHTML  = clima.name + ", " + clima.sys.country;
    lTemp.innerHTML = "Temperatura: " + clima.main.temp.toFixed(1) + "°
C";
    lHume.innerHTML = "Humedad: " + parseInt(clima.main.humidity) + "%"
    lPres.innerHTML = "Presión: " + parseInt(clima.main.pressure) + "
hPa";
    lVien.innerHTML = "Viento: " + parseInt(clima.wind.speed * 3.6) + "
Km/h";
    lPron.innerHTML = "Se espera: " + clima.weather[0].description;
    lMima.innerHTML = "Mínima: " + parseInt(clima.main.temp_min) +"° /
Máxima: " + parseInt(clima.main.temp_max) +"°";
    pImagen.src = "http://openweathermap.org/img/w/" + clima.weather[0].
icon + ".png";
    pImagen.setAttribute("width", "90px");
    lUact.innerHTML = ultimaActualizacion();
    cargando("i");
}
```

Utilizando los métodos de concatenación, escribimos, a través de la propiedad **innerHTML**, cada uno de los tags HTML con el respectivo valor obtenido mediante JSoN. Esta función recibe como respuesta (**r**) el objeto **responseText** recibido durante la petición anterior.

Registrar la última hora

El dato de la última hora de consulta del clima se obtiene del dispositivo del usuario. Para lograrlo, utilizamos el objeto JS **Date()**, que nos permite conocer la fecha y hora del sistema local. Veamos la función:

```
function ultimaActualizacion() {
    d = new Date();
    f = d.getDate(); if (f < 10) {f = "0" + f};
    m =  (d.getMonth() + 1); if (m < 10) {m = "0" + m};
    actualizacionFecha = f + "/" + m + "/" + d.getFullYear();
    m = d.getMinutes();
    if (m.length == 1) { m = m + 0 + m}
    h = d.getHours();
    if (h.length == 1) { h = h + 0 + h}
    actualizacionHora = h + ":" + m + " hs.";
    return actualizacionFecha + " " + actualizacionHora;
}
```

Finalmente, cuando se presione el botón actualizar de la barra superior, se invocará a la función **obtenerDatosDelClima()** para actualizar la información de pantalla:

```
btnRecargar.addEventListener("click", obtenerDatosDelClima);
```

Si todo ha salido bien, ya podemos probar nuestra aplicación del clima:

12:29
App del clima
Ciudad
El clima ahora
--° C
--%
-- hPa
-- Km/h
Pronóstico para hoy
--
--° C
Última actualización:
21-01-2019 13:45 hs.

12:29
App del clima
Buenos Aires, AR
El clima ahora
Temperatura: 23.6° C
Humedad: 22%
Presión: 1013 hPa
Viento: 12 Km/h
Pronóstico para hoy
Se espera: cielo claro
Mínima: 22° / Máxima: 24°

96% 23:24

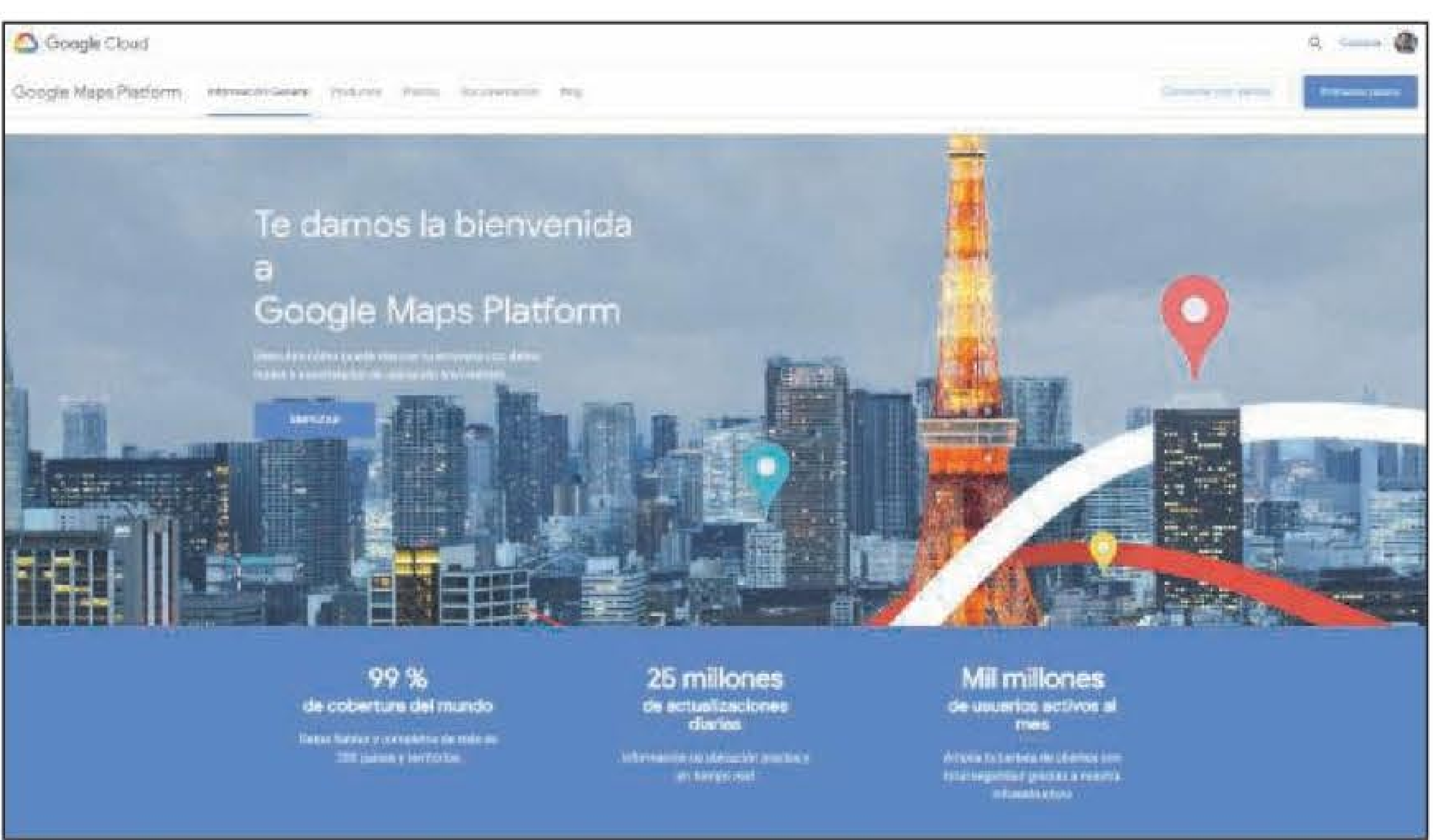
06 Geolocalización y mapas dinámicos

Este nuevo proyecto nos permitirá ahondar en el manejo de mapas dinámicos, una tarea muy habitual en la mayoría de las aplicaciones web y móviles. Veamos a continuación quiénes son los principales jugadores y cómo integrar mapas en nuestros desarrollos.

Ya sea para mostrar una ubicación corporativa en un formulario de contacto, para geolocalizar lugares de interés o, simplemente, para marcar nuestra posición, el uso de los mapas dinámicos en la actualidad es de gran importancia en la mayoría de los lugares que necesitan interactuar de una u otra forma con una solución que ayude a sus usuarios. Y dentro del mundo de los mapas, encontramos varios jugadores con algunos años de experiencia en el desarrollo de cartografía, que ponen a nuestra disposición herramientas de programación basadas en APIs para facilitarnos la tarea de desarrollar. Veamos a continuación quiénes están en el mercado.

Google Maps

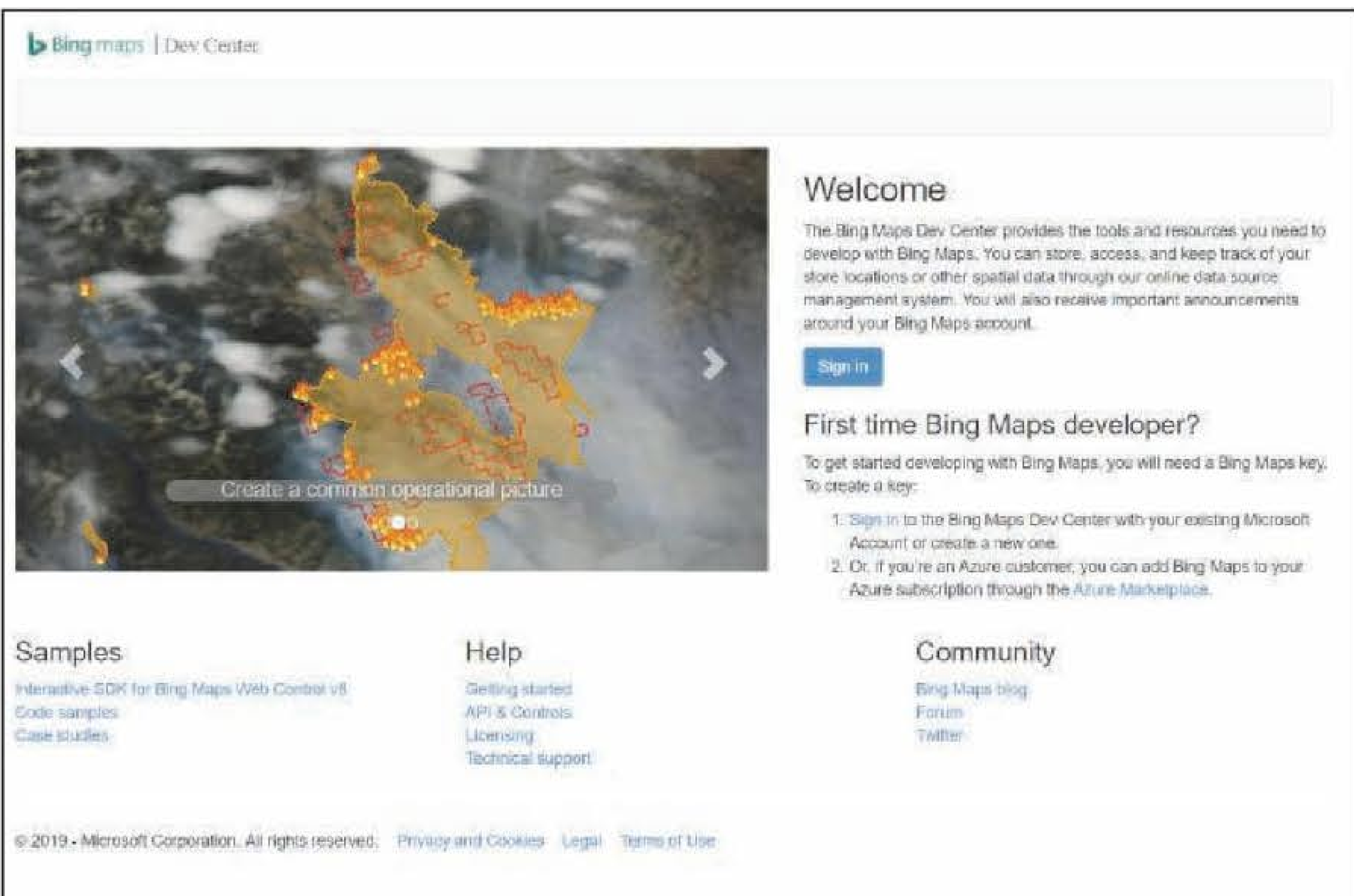
Podemos decir que, hoy en día, **Google Maps** es la plataforma de cartografía más popular y más utilizada a nivel mundial, que brinda una batería de herramientas indispensables para mostrar productos, empresas o servicios en un mapa dinámico: https://developers.google.com/maps/documentation/?authuser=2&hl=es



96% 23:24

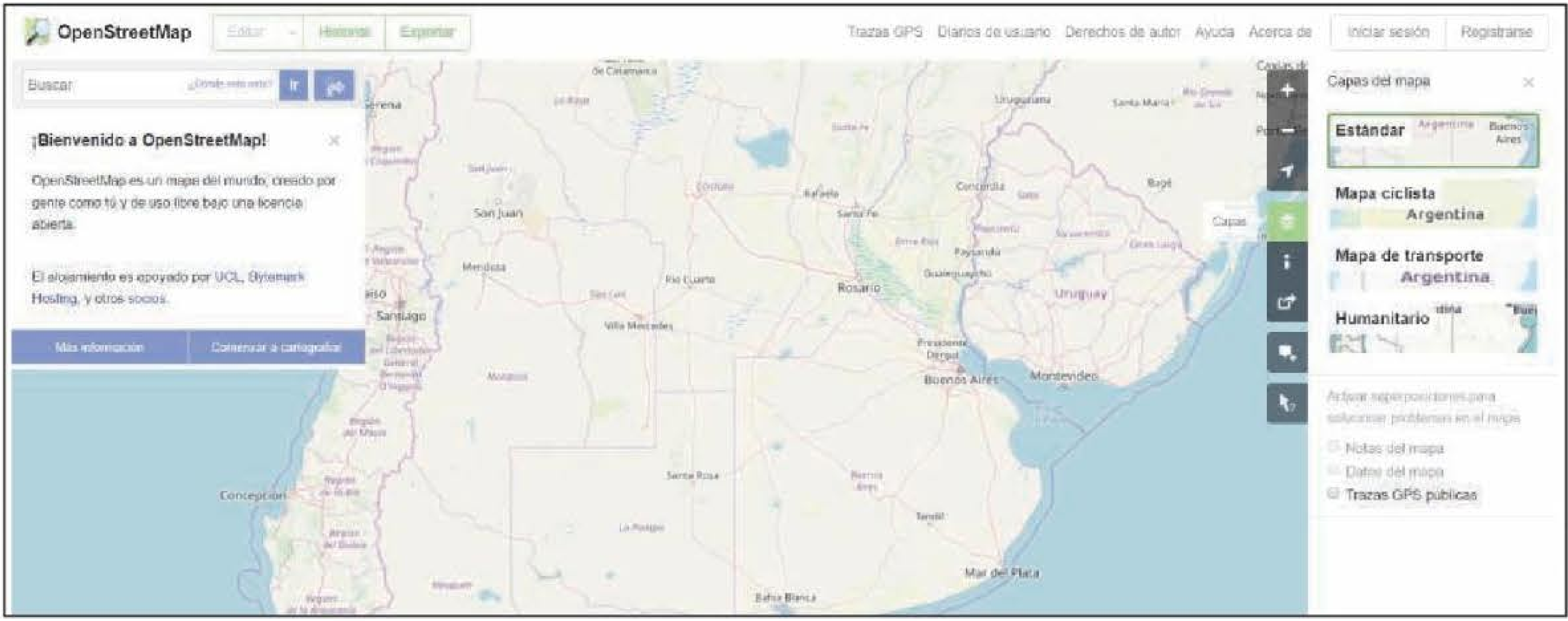
Bing Maps

Microsoft también tiene varios años dentro del ecosistema de mapas y servicios de geolocalización. **Bing Maps Portal** es la solución que la empresa de Redmond ofrece al mercado para el desarrollo de productos que integren mapas dinámicos. A través de su sitio web oficial, www.bingmapsportal.com, podemos acceder a la suscripción y uso de sus servicios.



Open Street Map

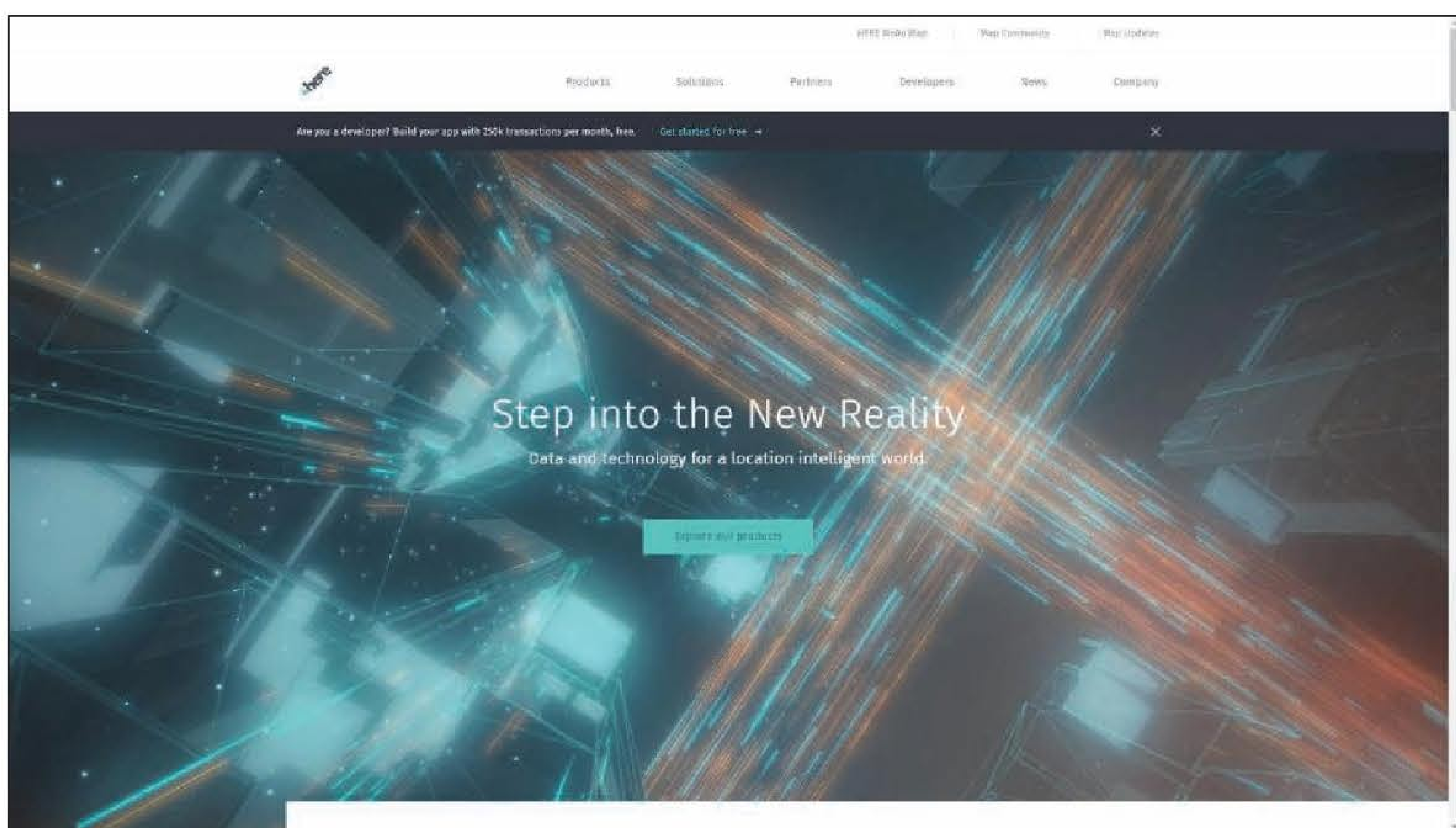
Basado en una economía colaborativa mundial, **Open Street Map** ofrece la mayoría de los servicios de cartografía y geolocalización, de manera gratuita. Es una gran opción para proyectos pequeños y medianos, pero hay que tener en cuenta que, para cualquier solución que requiera de efectividad y precisión, Open Street Map no será la mejor alternativa, ya que contiene muchos más errores que cualquiera de las otras plataformas, y estos solo se solucionan a través del reporte fundamentado de usuarios de su comunidad: www.openstreetmap.org.



96% 23:24

HERE Maps

Antes conocido como Navteq, Nokia Here o Nokia Maps, HERE es una de las empresas más especializadas en el terreno cartográfico digital. Fundada por Nokia en 1985, es la que más llegada tiene en los dispositivos móviles a nivel mundial, ya que se inició comercialmente en estos equipos con el lanzamiento de los primeros dispositivos GPS para barcos y automóviles, hace ya varias décadas. HERE Maps será la API elegida para desplegar mapas desde nuestras aplicaciones web: www.here.com.



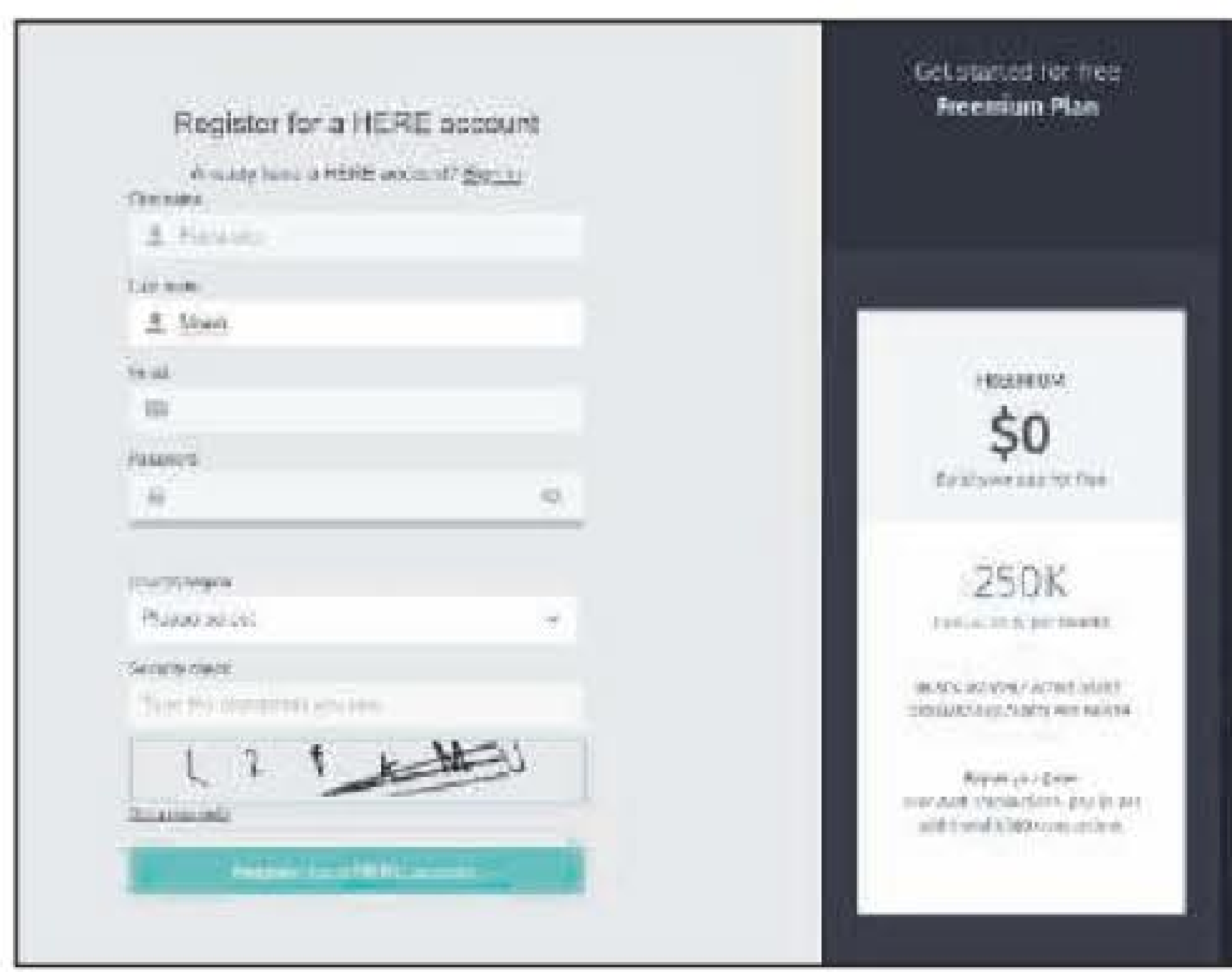
Las ventajas de HERE Maps

Lo más destacable de esta plataforma es que, en la suscripción gratuita, cuenta con más de 250.000 transacciones mensuales disponibles, frente a las 5000 mensuales que nos ofrecen Bing Maps o Google Maps. Sus APIs son de las más fáciles de implementar, lo cual llevará a que nuestros desarrollos sean realizados de una manera mucho más ágil que si usáramos otras plataformas similares. Finalmente, contamos con la posibilidad de desactivar el uso obligatorio de HTTPS para nuestros desarrollos; por lo tanto, podemos probarlos de modo local, sin necesidad de tener un servidor web acorde a este requerimiento.

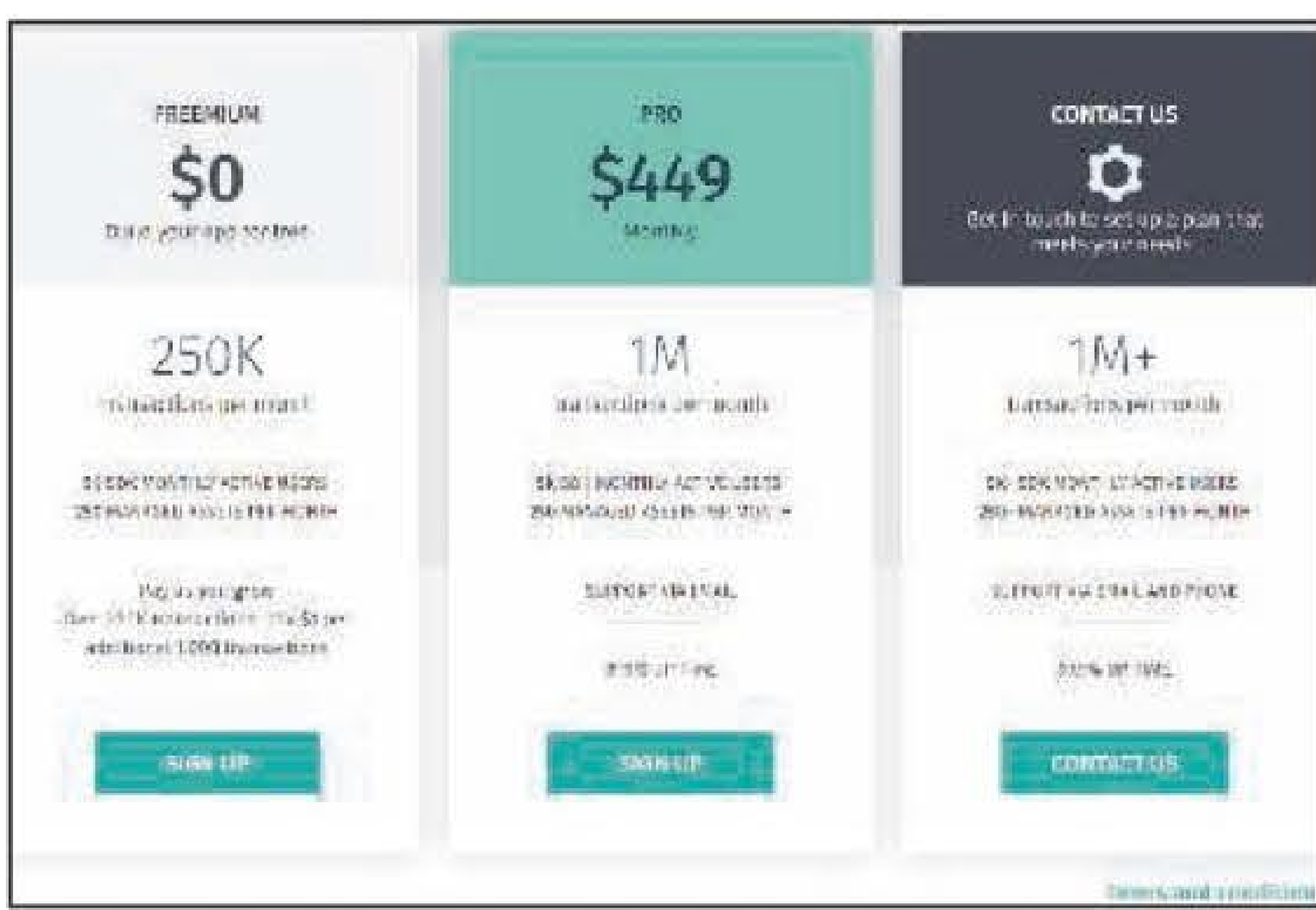
Suscripción y gestión de claves

Veamos a continuación cuáles son los pasos necesarios que debemos realizar para comenzar a utilizar la API de la plataforma HERE Maps. En primer lugar, ingresamos a la URL https://developer.here.com y pulsamos sobre la opción **GET STARTED FOR FREE**.

Paso a paso: Suscripción y gestión de claves

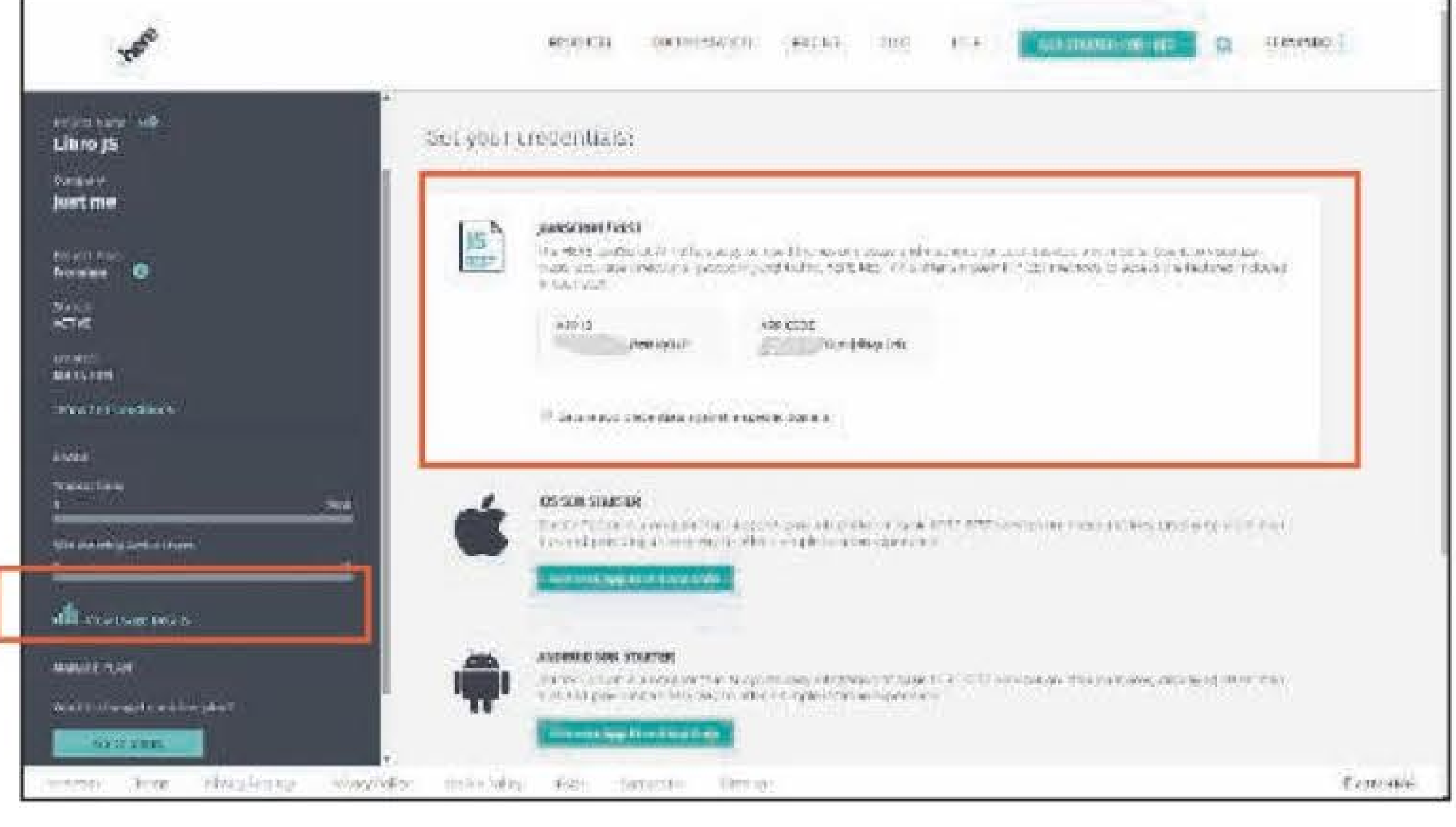


1 Se abrirá el formulario de suscripción, donde nos piden los datos básicos necesarios para gestionar nuestra cuenta. Finalizado el ingreso de estos datos, pulsamos el botón **REGISTER**, que terminará el proceso para poder comenzar a utilizar los servicios de HERE Maps.

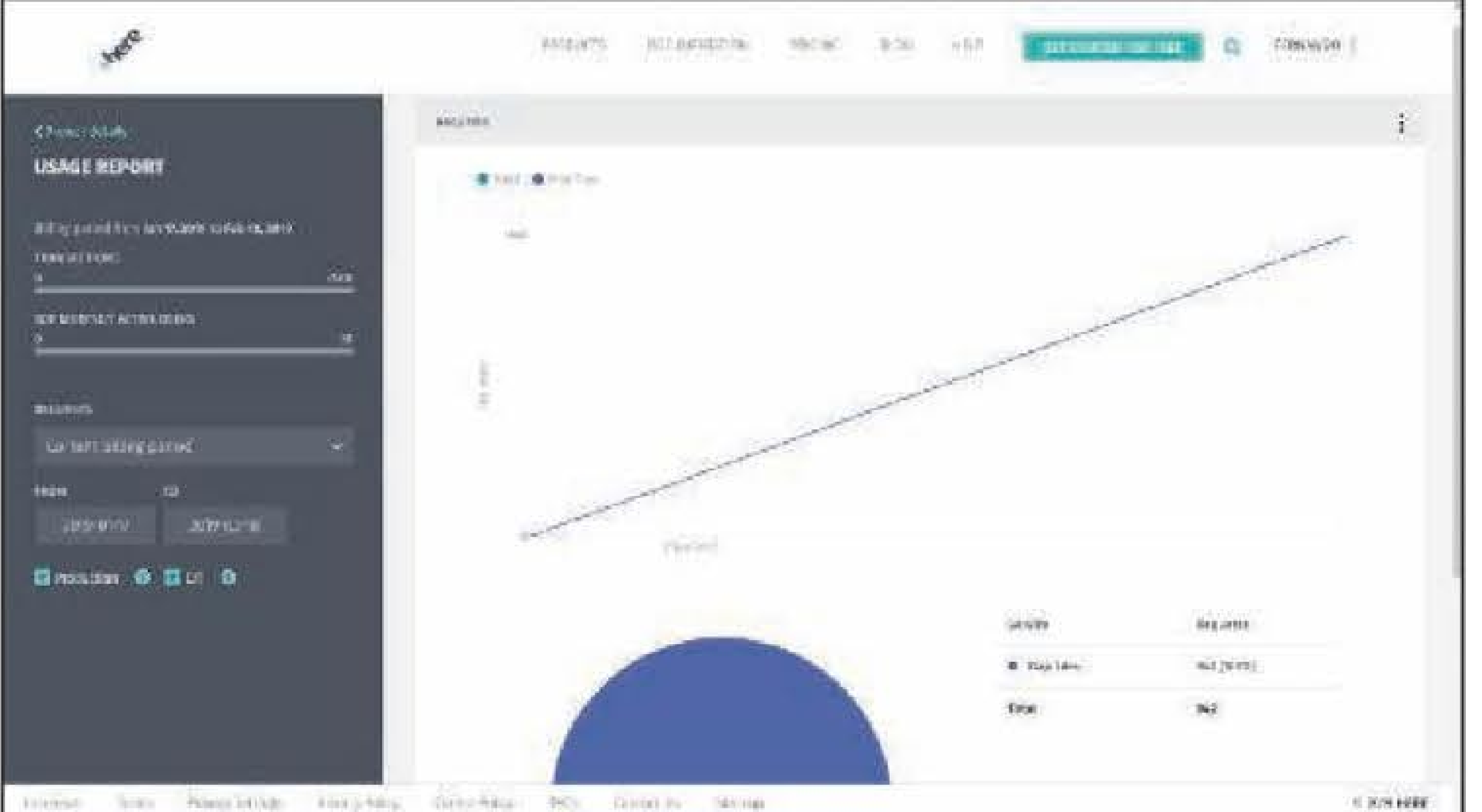


2 Finalizado el proceso de suscripción e iniciada la sesión en esta plataforma, accedemos a un panel de control donde podemos elegir el tipo de suscripción que queremos iniciar. Entre las opciones, podemos elegir la gratuita (**fremium**) o directamente una suscripción paga.

3 A continuación, indicamos para qué plataforma queremos desarrollar. En nuestro caso, elegimos **REST/JavaScript**. Se crearán las credenciales que utilizaremos dentro de nuestros desarrollos; podemos crear tantas como aplicaciones desarrollemos.

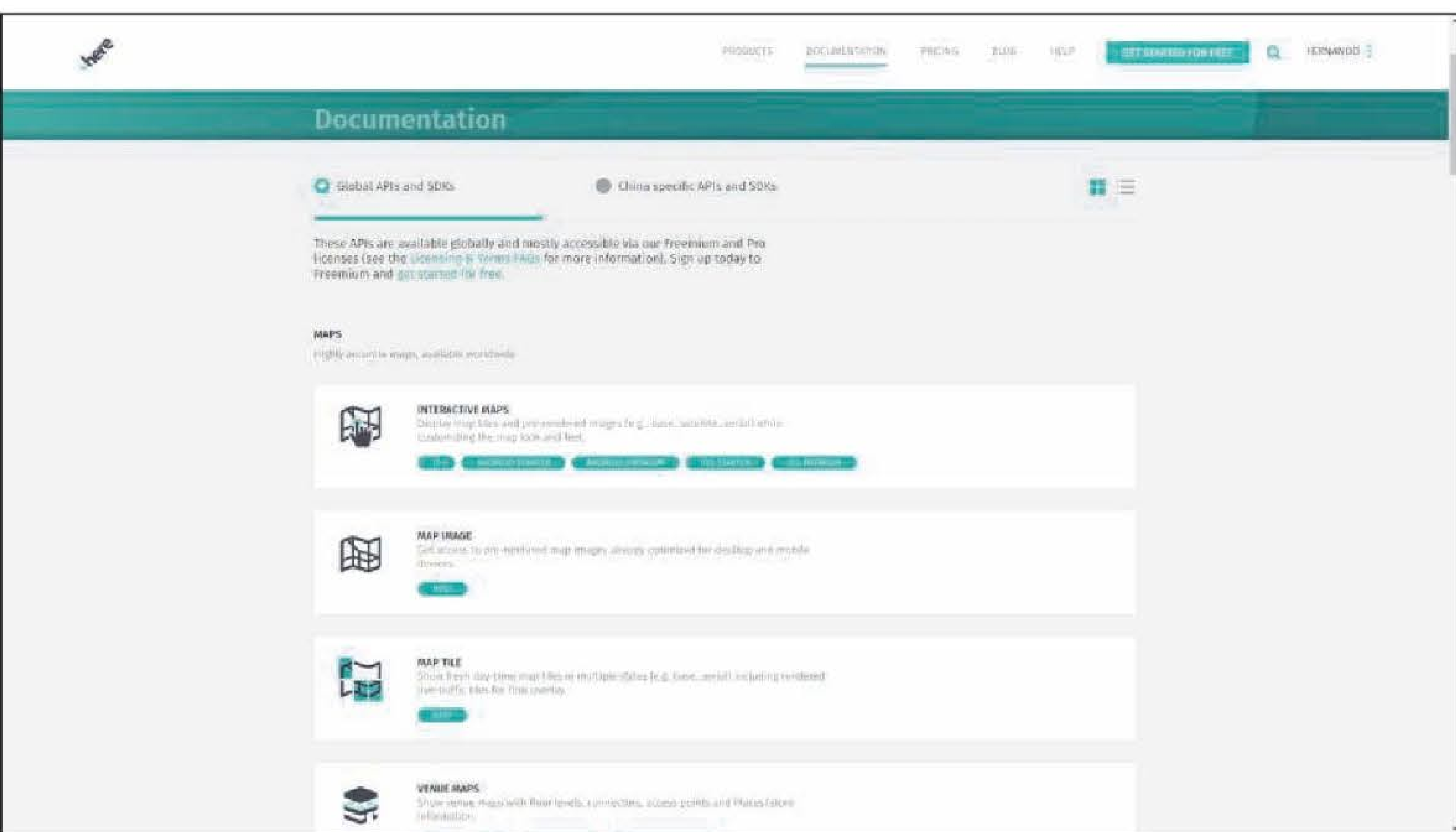


4 Sobre el panel izquierdo, encontramos los detalles de nuestro proyecto y el consumo de eventos realizados sobre él. Pulsando sobre la opción **View Usage Details**, accedemos a un panel gráfico con un detalle más claro sobre el uso de la plataforma; desde allí podemos descargar los datos en formato CSV.



Sistema de documentación y ayuda

Si ingresamos en la URL https://developer.here.com/documentation, accederemos al sistema de documentación y ayuda de esta plataforma. Allí encontraremos estructurados todos los servicios de HERE Maps, desde cómo mostrar un mapa, hasta el manejo de datos geoespaciales complejos, y la descarga de mapas y funciones para trabajar en modo offline (esto último, solo para suscripciones Premium).



Estructura de la API

Exploremos a continuación la estructura que propone HERE Maps para desarrollar una aplicación web con sus mapas integrados. Para hacerlo, vamos a descargar el proyecto **Maps-Ejemplo.zip** del repositorio de archivos de esta obra. En él encontraremos un simple archivo HTML con las referencias e integración del código JS incluido. Como podemos ver, en el apartado **<head>** hay dos líneas agregadas que referencian a los archivos JS necesarios para que el mapa funcione.

```
<script src=http://js.api.here.com/v3/3.0/mapsjs-core.js></script>
<script src="http://js.api.here.com/v3/3.0/mapsjs-service.js"></script>
```

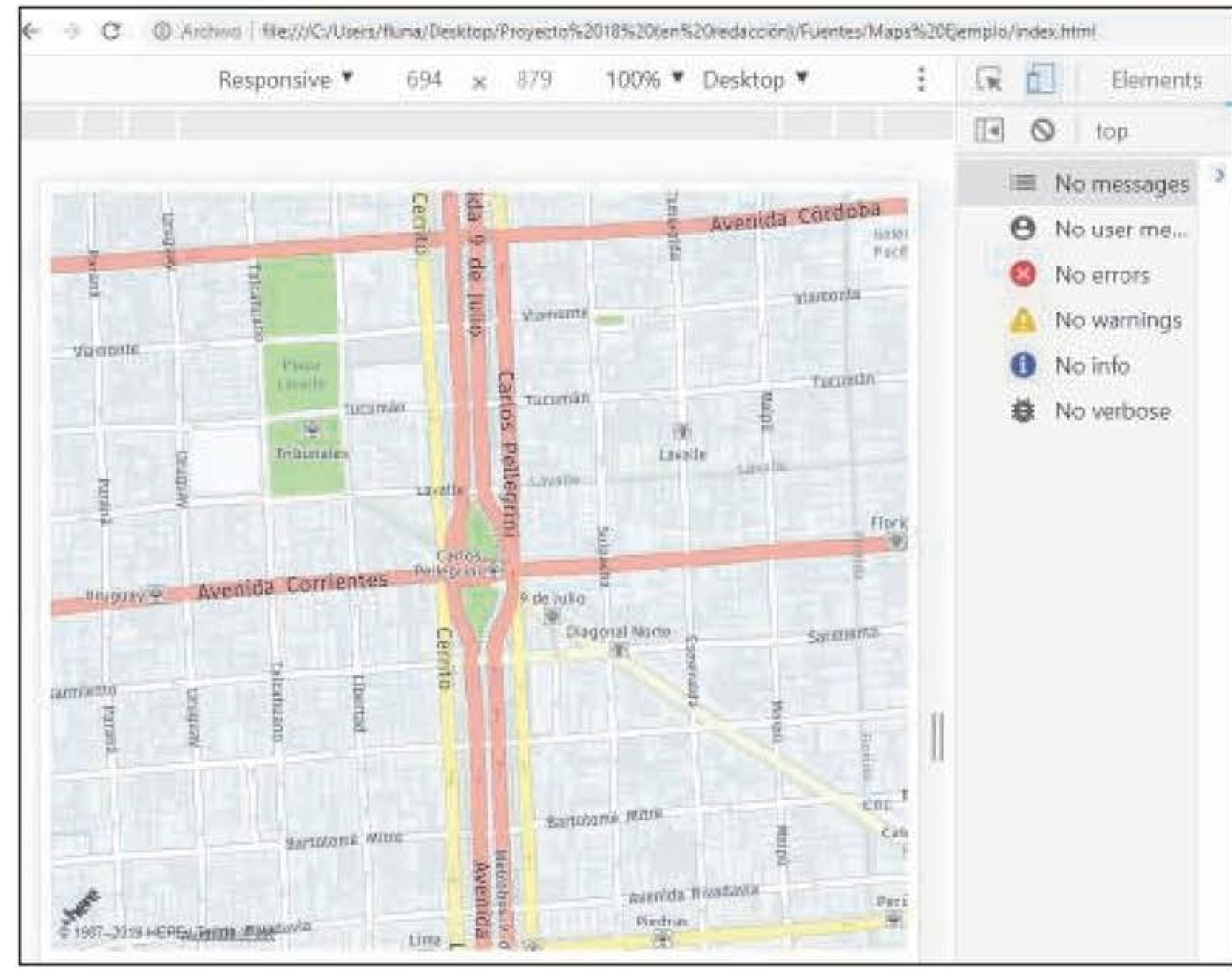
El primero de ellos hace referencia al **Core**, o base necesaria del mapa, mientras que el segundo hace referencia al servicio del mapa en sí. Veamos ahora el apartado **<body>**:

```
<div style="width: 100%; height: 600px;" id="mapContainer"></div>
```

Aquí tenemos un elemento **<div>** declarado, cuyo ID es **mapContainer**. Este ID lo utilizaremos desde JS para referenciar el mapa y que se muestre en pantalla cuando se cargue el documento HTML. Por último, encontramos un **<script>** donde se aloja el código necesario para mostrar el mapa dentro del elemento **div**.

```
index.html
<head>
    <title>Mapa dinámico</title>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1.0, width=device-width, user-scalable=no" />
    <script src="http://js.api.here.com/v3/3.0/mapsjs-core.js"></script>
    <script src="http://js.api.here.com/v3/3.0/mapsjs-service.js"></script>
</head>
<body>
    <div style="width: 100%; height: 600px;" id="mapContainer"></div>
    <script>
        var platform = new H.service.Platform({
        'app_id': 'INSERTA_TU_APP_ID',
        'app_code': 'INSERTA_TU_APP_CODE'
        });

        var maptypes = platform.createDefaultLayers();

        var map = new H.Map(
        document.getElementById('mapContainer'),
        maptypes.normal.map,
        {
        zoom: 16,
        center: { lng: -58.381592, lat: -34.603722 }
        });
    </script>
</body>
Preview Available    Lín. 15, Col. 45   Espacios: 4   UTF-8   CRLF   HTML
```

Dentro del código JS alojado en el script, debemos ubicar **app_id** y **app_code**, y agregar en ellos el ID y el código alfanumérico que nos asignó HERE Maps en la suscripción. Finalmente, podemos ejecutar nuestro documento HTML en un navegador web de manera local. El resultado debe ser similar a la siguiente imagen.



Como podemos apreciar en este mapa, el mismo es dinámico, ya que podemos comenzar a desplazarnos por él sin ningún problema. Ingresamos las coordenadas en formato (LATITUD, LONGITUD) para que visualice, como punto central, el microcentro de la ciudad donde residimos.

96% 23:24

El objeto Map

Si bien en la documentación oficial encontraremos el código necesario para nuestro desarrollo, es bueno que tengamos siempre presentes los pasos que debemos seguir cuando creamos un nuevo mapa. El primero de ellos es iniciar el objeto **Platform**:

```
var platform = new H.service.Platform({
'app_id': 'TU_API_ID', 'app_code': 'TU_APP_CODE'
});
```

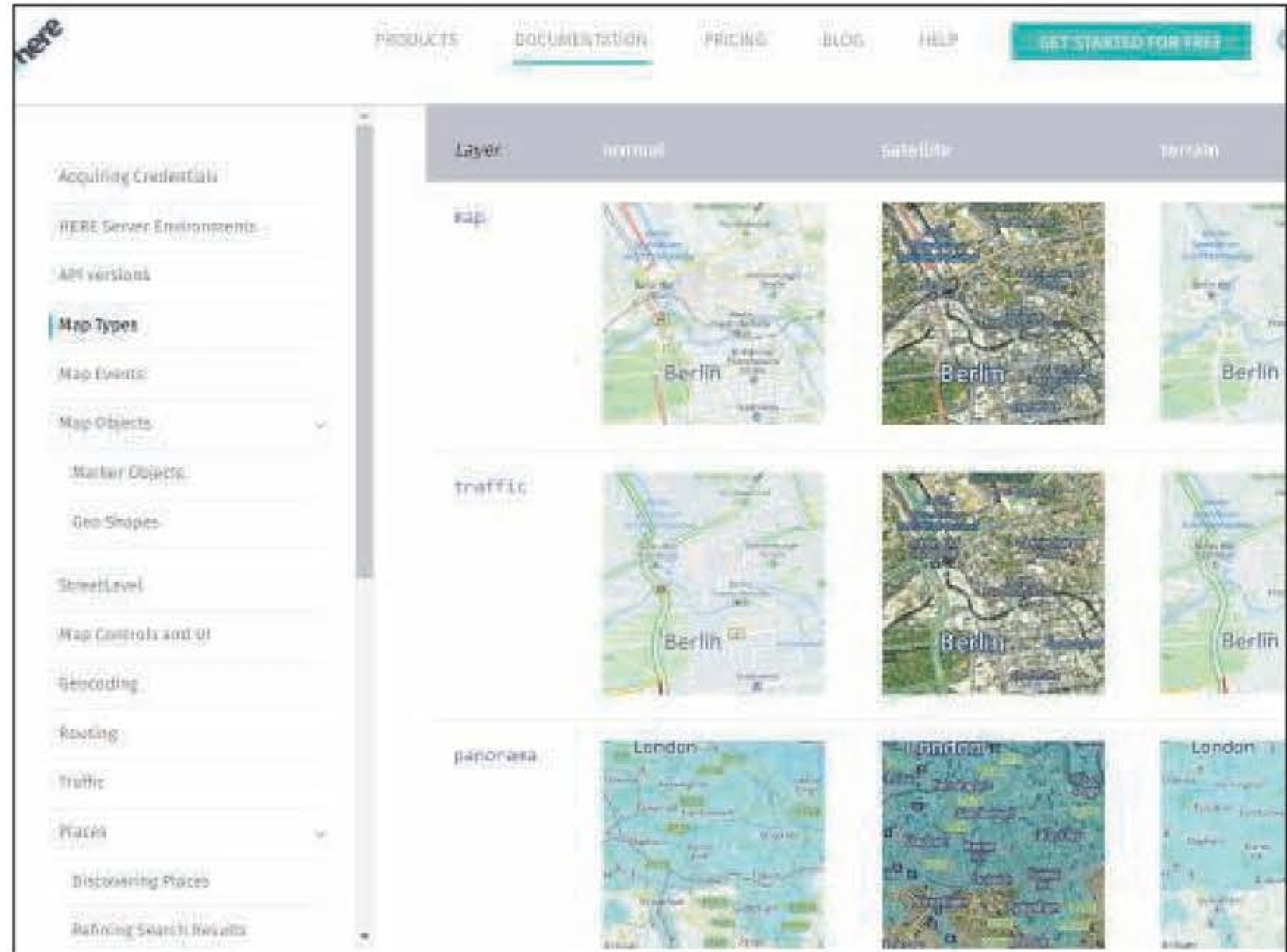
Con el objeto Platform definido, creamos a continuación el tipo de capa que deseamos visualizar en el mapa:

```
var maptypes = platform.createDefaultLayers();
```

Y, por último, iniciamos y visualizamos el objeto **Map** dentro del elemento **<div>** de nuestro documento HTML:

```
var map = new H.Map(document.getElementById('mapContainer'),
            maptypes.normal.map,
            {zoom: 16, center: { lng: -58.381592,
                                 lat: -34.603722 }}
      );
```

Creamos la variable **map** indicando en qué elemento HTML se visualizará. A través de **maptypes**, señalamos que deseamos ver el mapa normal, con **zoom = 16**, y centrándolo en la latitud y longitud indicadas. Podemos visualizar diferentes tipos de mapas, incluyendo terreno, satelital y tráfico, entre otros tantos más.



96% 23:24

Proyecto práctico: localizador de cafeterías

Realizaremos a continuación un proyecto práctico denominado **Cofilokeitor**, que nos permitirá integrar una funcionalidad de los mapas dinámicos HERE. Mostraremos el mapa dinámico de una ciudad y, debajo de él, un listado de cafeterías con una breve descripción de cada una. Cada cafetería tendrá una latitud y longitud determinadas, lo cual permitirá que, al hacer clic o tap sobre ella, se visualice en el mapa con un icono personalizado.

Documento HTML base

Para representar visualmente bien nuestro proyecto, descargamos el archivo **Cofilokeitor-Base.zip** del repositorio de esta obra. Descomprimimos su contenido en una carpeta y luego creamos un nuevo proyecto en Visual Studio. Al abrir el documento HTML, nos encontramos con la estructura representada en el siguiente gráfico:

Qué haremos desde JS

El documento HTML no tiene que preocuparnos, ya que la interacción en general la haremos desde JavaScript. Lo que sí debemos conocer, es que tiene un <div> cuyo ID es **map**, y los cuatro iconos derechos del listado de cafeterías, denominados en sus ID como **i1**, **i2**, **i3** e **i4**, oficiarán de link para ubicar físicamente cada cafetería en el mapa. En la subcarpeta **images**, tenemos un set de iconos iguales, que difieren en su tamaño en píxeles. Utilizaremos el que tenga las dimensiones visuales que mejor se ajusten a nuestro mapa.

Agregar la referencia hacia la API de Mapas

Vamos a ocuparnos ahora de agregar la referencia a los archivos JS correspondientes a la API de HERE Maps. El código que debemos agregar en el documento HTML, justo debajo de la declaración CDN correspondiente a Materialize CSS, es el siguiente:

```
<link rel="stylesheet" href="https://js.api.here.com/v3/3.0/mapsjs-ui.
css?dp-version=1542186754" />
<script src="https://js.api.here.com/v3/3.0/mapsjs-core.js"></script>
<script src="https://js.api.here.com/v3/3.0/mapsjs-service.js"></script>
<script src="https://js.api.here.com/v3/3.0/mapsjs-ui.js"></script>
<script src="https://js.api.here.com/v3/3.0/mapsjs-mapevents.js"></
script>
```

Para no copiar manualmente todo esto, dentro del documento HTML base que descargamos para realizar este proyecto, hay un archivo TXT que incluye toda esta referencia; solo tenemos que pegarlo directamente en el apartado **<head>**. El siguiente paso es agregar la declaración JS hacia nuestro archivo de script, justo debajo de código anterior:

```
<script defer src="cofilokeitor.js"></script>
```

La estructura del documento HTML debe quedar como en el siguiente gráfico:

```
OCTYPE html>
ml>
head>
   <title>Cofi Lokeitor</title>
   <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0"/>
   <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css">
   <script defer src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>
   <link rel="stylesheet" type="text/css" href="https://js.api.here.com/v3/3.0/mapsjs-ui.css?dp-version=1542186754" />
   <script type="text/javascript" src="https://js.api.here.com/v3/3.0/mapsjs-core.js"></script>
   <script type="text/javascript" src="https://js.api.here.com/v3/3.0/mapsjs-service.js"></script>
   <script type="text/javascript" src="https://js.api.here.com/v3/3.0/mapsjs-ui.js"></script>
   <script type="text/javascript" src="https://js.api.here.com/v3/3.0/mapsjs-mapevents.js"></script>
   <script defer src="cofilokeitor.js"></script>
/head>
```

96% 23:24

Archivo de proyecto

Hacemos **Ctrl + clic** sobre el archivo **cofilokeitor.js** para crearlo. Dentro de él agregamos el siguiente código:

```
var icono = new H.map.Icon("images/coffePin128.png");
var i1 = document.getElementById("cafe01");
var i2...
```

Declaramos la variable **icono** donde almacenamos el tipo de marcador (**Marker**) que usaremos como pin en el mapa. Luego declaramos la variable **i1**, vinculándola con el icono de cada una de las cafeterías. Repetimos este último punto con las variables **i2**, **i3** e **i4**. Por último, declaramos el Event Listener que espera a que cargue el documento HTML. Ahora vamos a inicializar el mapa a través de todos los objetos explicados anteriormente, y agregaremos algunas características más, por medio del siguiente código:

```
var platform = new H.service.Platform({
app_id: 'TU_APP_ID:AQUI',
app_code: 'TU_APP_CODE_AQUI'
});
var pixelRatio = window.devicePixelRatio || 1;
var defaultLayers = platform.createDefaultLayers({
    tileSize: pixelRatio === 1 ? 256 : 512, ppi: pixelRatio === 1 ? un-
defined : 1024
});
```

A diferencia del primer ejemplo de mapa que vimos, en este nuevo código de inicialización de la API de mapa, agregamos el soporte para **pixel ratio**. Esto permitirá rediseñar el mapa dinámico, de acuerdo con la capacidad propia de la pantalla de cada dispositivo que cargue nuestra aplicación web. Cuanto mejor sea la calidad de display, mayor será el pixel ratio y, por lo tanto, mejor calidad tendrá la visualización del mapa. Declaramos a continuación la variable **map**, indicándole como parámetro el pixel ratio detectado:

```
var map = new H.Map(document.getElementById('map'),
        defaultLayers.normal.map, {pixelRatio: pixelRatio});
var ui = H.ui.UI.createDefault(map, defaultLayers, 'es-ES');
var behavior = new H.mapevents.Behavior(new H.mapevents.MapEvents(map));
centrarMapa(map);
```

Junto con la declaración de la variable **map**, también creamos la variable **behavior**, la cual hereda el objeto de comportamientos del mapa. Esto permitirá desplazarnos por él y realizar

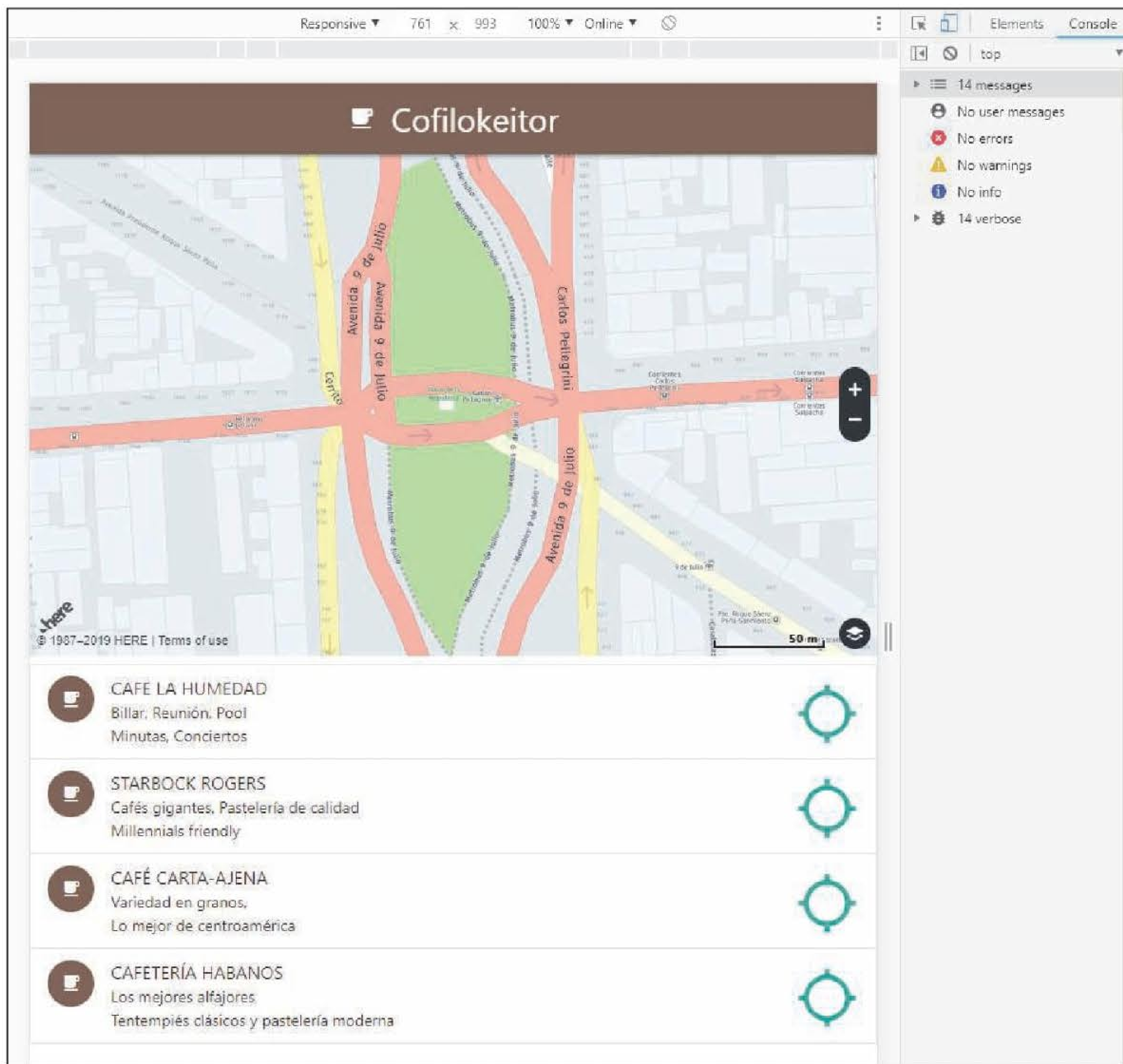
zoom, entre otras opciones. Finalmente, invocamos la función **centrarMapa()**, que crearemos más adelante. Nuestro código, hasta el momento, debe estar estructurado como se observa en la siguiente figura:

```
1   var icono = new H.map.Icon("images/coffePin128.png");
2   var i1 = document.getElementById("cafe01");
3   var i2 = document.getElementById("cafe02");
4   var i3 = document.getElementById("cafe03");
5   var i4 = document.getElementById("cafe04");
6
7   document.addEventListener("DOMContentLoaded", function() {
8
9           //DECLARAR EL OBJETO MAP Y SETEARLO PARA QUE FUNCIONE
10          var platform = new H.service.Platform({
11          app_id: '4XVJohe8e5CJhzMNy3ZP',
12          app_code: 'NZ-XSxY0ZVDJnKjdHquCrQ'
13          });
14          var pixelRatio = window.devicePixelRatio || 1;
15          var defaultLayers = platform.createDefaultLayers({
16              tileSize: pixelRatio === 1 ? 256 : 512, ppi: pixelRatio === 1 ? undefined : 1024});
17
18          var map = new H.Map(document.getElementById('map'),
19                  defaultLayers.normal.map, {pixelRatio: pixelRatio});
20          var ui = H.ui.UI.createDefault(map, defaultLayers, 'es-ES');
21          var behavior = new H.mapevents.Behavior(new H.mapevents.MapEvents(map));
22          centrarMapa(map);
```

Solo nos resta declarar la función **centrarMapa()**, que recibirá el objeto **map**, para poder establecer los puntos de latitud y longitud en él. Estos puntos de referencia serán los que indiquen qué lugar del mundo debe visualizarse en el mapa y, también, cuál es el punto central de la imagen que debemos mostrar. Para esto vamos a utilizar las funciones **setCenter()** junto a la latitud y la longitud, y **setZoom()** para ajustar la distancia de visualización:

```
function centrarMapa(map){
    map.setCenter({lat:-34.603722,
                   lng:-58.381592});
    map.setZoom(18);
    }
```

Si ejecutamos nuestro proyecto ahora, debemos visualizar al menos el mapa principal ya cargado dentro del **<div> map** del documento HTML, similar a la siguiente imagen:



Hardcoding

El **hardcoding**, también conocido en español como “**hardcodeo**”, es la acción de escribir código forzado para que ocurra algo dentro de un programa. A continuación, realizaremos esta acción para ver en el mapa principal las cafeterías listadas en el apartado inferior de esta aplicación web. De esta manera, el proyecto no resultará tan extenso; de otro modo, deberíamos recurrir a un archivo del tipo **JSoN** o, más complejo aún, armar una base de datos exclusiva para que contenga esta información.

Evento del listado de cafeterías

Veamos a continuación el siguiente código:

```
i1.addEventListener("click", function() {
    la = -34.635681;
    lo = -58.424454;
    var pin = new H.map.Marker({ lat:  la , lng:  lo  }, { icon: icono });
    map.addObject(pin);
    map.setCenter({ lat:  la , lng:  lo  });
})
```

Lo que hacemos en este caso es ejecutar el código contenido en **function()** cuando se dispara el evento **click** en la imagen cuyo ID es **i1**. Las variables **la** y **lo** almacenan la latitud y la longitud que deseamos ver en el objeto **map**. Finalmente, declaramos la variable **pin**, que almacena el método **Marker()**, pasándole a este la latitud y la longitud predefinidas, e indicándole que el icono que se ubicará en el mapa es el referenciado en la variable **icono**, la primera línea de código de este archivo JS.

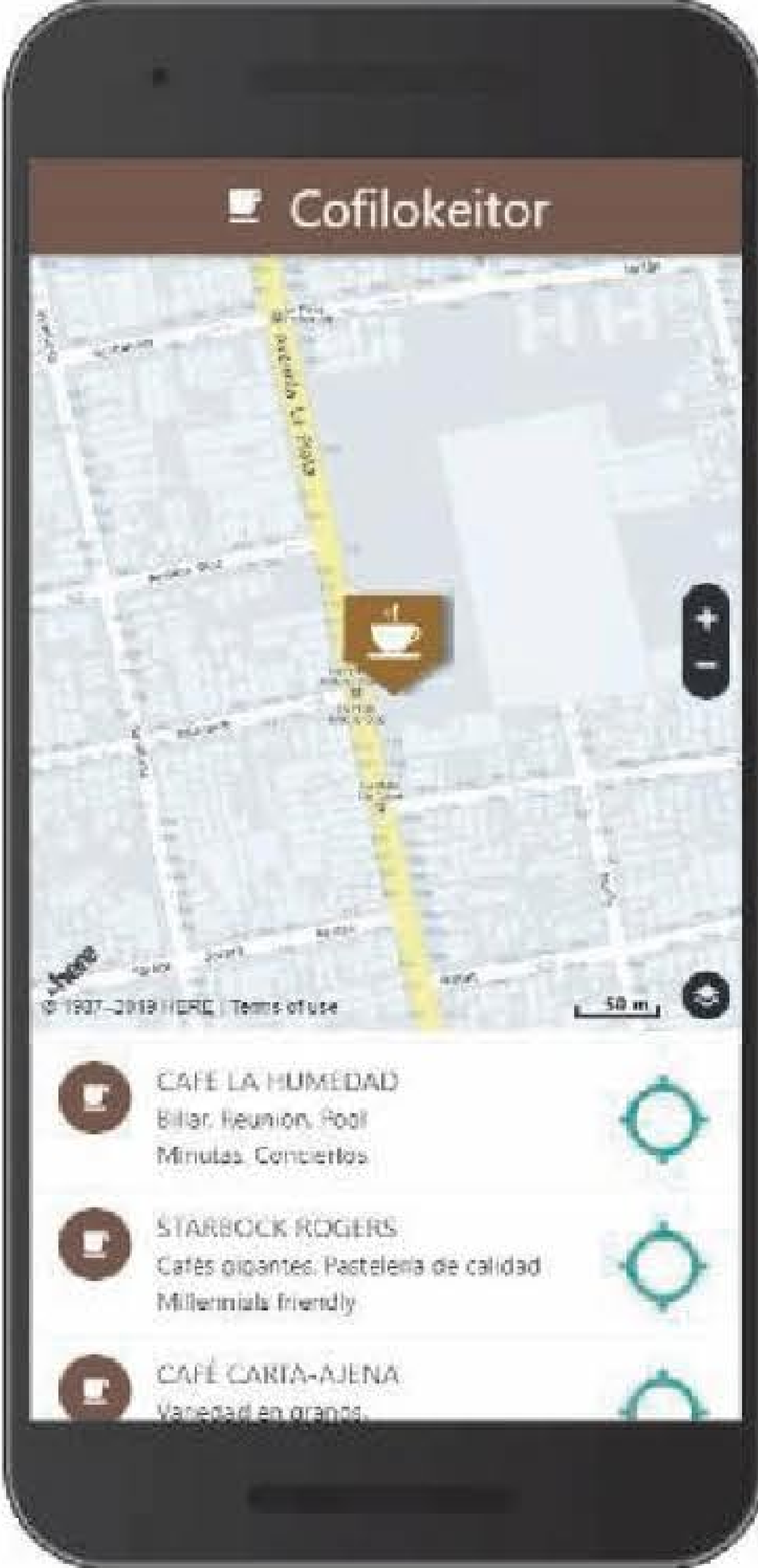
La latitud y la longitud definidas en esta función han sido elegidas al azar. A continuación, detallamos todas las latitudes y longitudes que usaremos en este ejemplo:

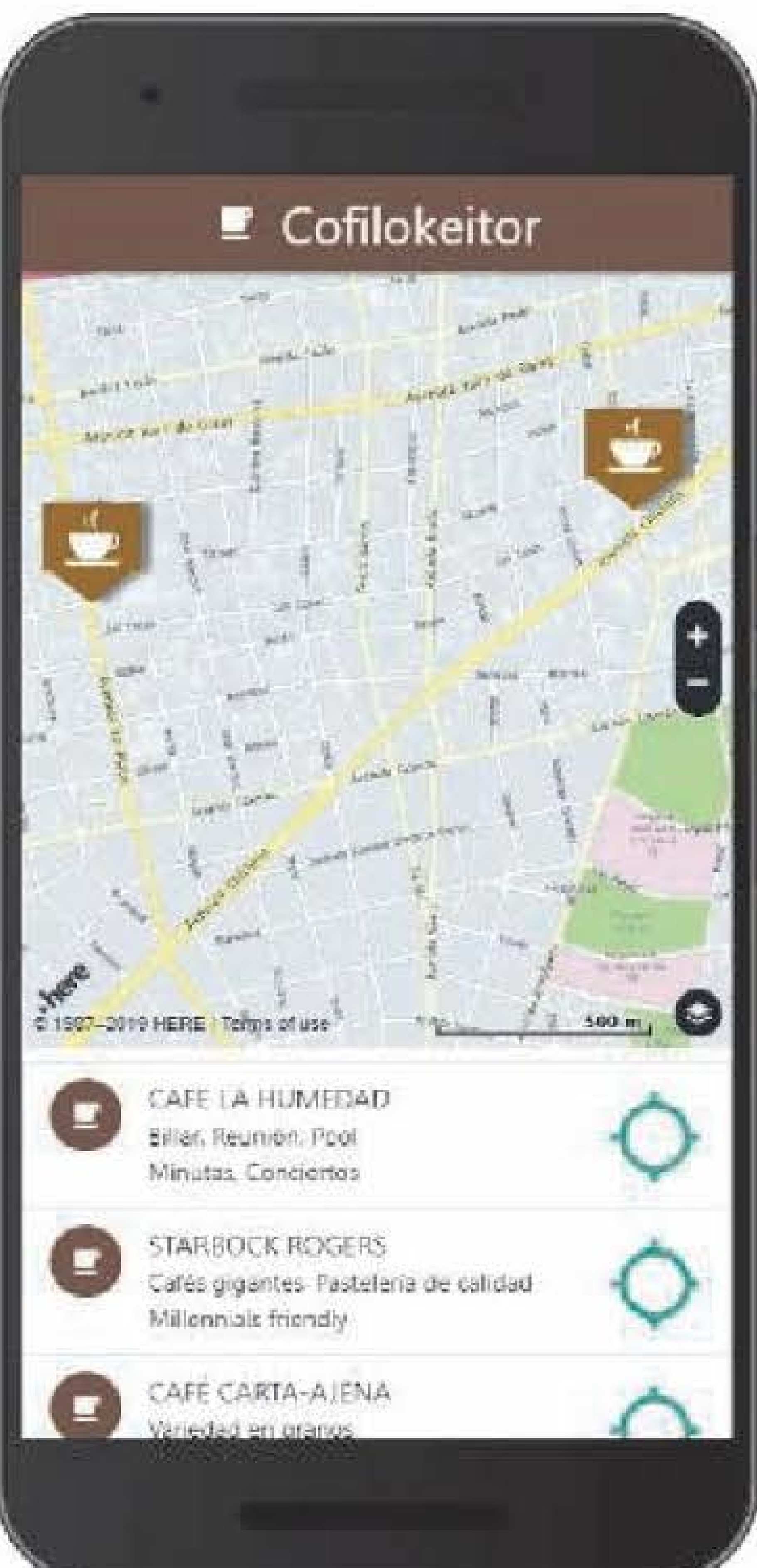
```
//i1              //i3
-34.633787;         -34.631623;
-58.410958;       -58.441554;
//i2                     //i4
-34.635681;       -34.633548;
-58.424454;         -58.460436;
```

Nuestra tarea ahora es repetir el código que detecta cada **evento click**, por cada uno de los iconos que tenemos en el documento HTML. En cada repetición, debemos modificar los valores de latitud y longitud, y también cambiar el nombre de la variable **i1** por el resto de las variables declaradas al inicio de este archivo de script. El resultado debe ser similar al de la siguiente figura:

```
cafilokeitor.js ●
//CAFETERIA 01
i1.addEventListener("click", function() {
        la = -34.633787;
        lo = -58.410958;
        var pin = new H.map.Marker({ lat:  la , lng:  lo  }, { icon: icono });
        map.addObject(pin);
        map.setCenter({ lat:  la , lng:  lo  });
    })

//CAFETERIA 02
i2.addEventListener("click", function() {
    la = -34.635681;
    lo = -58.424454;
    var pin = new H.map.Marker({ lat:  la , lng:  lo  }, { icon: icono });
    map.addObject(pin);
    map.setCenter({ lat:  la , lng:  lo  });
})
//CAFETERIA 03
i3.addEventListener("click", function() {
    la = -34.631623;
    lo = -58.441554;
    var pin = new H.map.Marker({ lat:  la , lng:  lo  }, { icon: icono });
    map.addObject(pin);
    map.setCenter({ lat:  la , lng:  lo  });
})
//CAFETERIA 04
i4.addEventListener("click", function() {
    la = -34.633548;
    lo = -58.460436;
    var pin = new H.map.Marker({ lat:  la , lng:  lo  }, { icon: icono });
    map.addObject(pin);
    map.setCenter({ lat:  la , lng:  lo  });
})
```

Habiendo completado este último paso, solo nos resta ejecutar la aplicación web en el navegador, para finalmente probar su funcionamiento. Si todo está bien, cuando hagamos clic sobre el icono derecho de cada una de las cafeterías mostradas en el listado, se deberá crear un marcador en el mapa, posicionado en la latitud y longitud que corresponda.





Ya tenemos una aplicación web completamente funcional, que accede a mapas dinámicos y muestra los resultados en pantalla. Si investigamos más las diversas opciones que nos da la API de **HERE maps**, podremos potenciar como más nos guste cualquier aplicación que requiera de geolocalización o de otros servicios asociados a un mapa dinámico.

Ante cualquier duda que haya quedado sobre este código, en el repositorio de archivos de esta obra, está la aplicación web lista para descargar bajo el nombre: **Cofilokeitor-Completo.zip**. Es posible usarla tanto en navegadores web de escritorio como en teléfonos móviles o tablets.

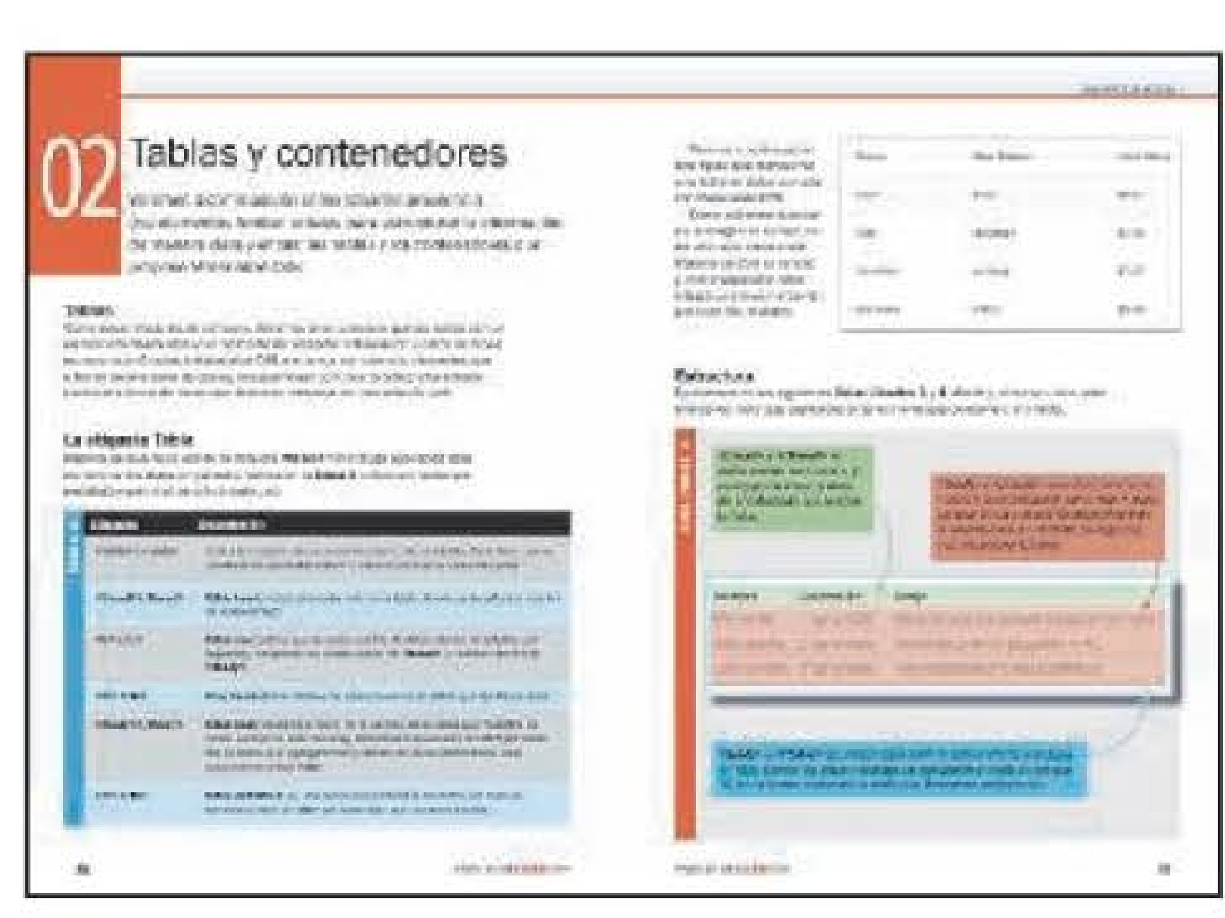
Desarrollo Mobile

ACERCA DE ESTE CURSO

Este curso nos enseñará a programar aplicaciones web bajo la filosofía **Mobile First** utilizando un framework basado en **Material Design**. Abordaremos desde un diseño básico que permite crear una web estática, e iremos escalando en componentes y conocimientos hasta crear una web 100% dinámica. Finalmente aprovecharemos características de los dispositivos móviles y convertiremos nuestra web en una **PWA**, apta para el ecosistema **Android** así como también para **iOS** y **iPadOS**.

EL VOLUMEN II

En el segundo e-book de la colección potenciaremos la barra de navegación, y la integraremos en modo lateral. Aprenderemos las diferentes opciones de botones que podemos aplicar con Materialize, cómo integrar mensajes de tipo tooltip y toast, y también cómo integrar mapas dinámicos.



02 Tablas y contenedores

03 Tarjetas

SOBRE EL AUTOR

Fernando Luna es Analista de Sistemas y Technical Writer. Lleva más de 25 años desarrollando software para diferentes plataformas, y algo más de una década colaborando como autor de publicaciones técnicas orientadas a la programación y las nuevas tecnologías. Su pasión se divide entre la programación, la electrónica y la educación.

REDUSERS.PREMIUM.COM

RedUSERS PREMIUM es la biblioteca digital de USERS, desde donde podrán acceder a cientos de publicaciones: informes, e-books, guías, revistas y cursos. Todo el contenido está disponible online y offline, y para cualquier dispositivo. Publicamos una novedad, al menos, cada siete días.

REDUSERS.COM

En nuestro sitio podrán encontrar noticias relacionadas y participar de la comunidad de tecnología más importante de América Latina.