Desarrollo para entornos multiplataforma

Pau Ballada

PID_00209910

Universitat Oberta de Catalunya

Índice

1. Desarrollo de una aplicación sencilla con jQuery Mobile y PhoneGap 5
1.1. Introducción 5
1.2. Wireframes 5
1.3. Desarrollo 6
1.3.1. Preparación 8
1.3.2. Código inicial 9
1.3.3. Librería jQuery 11
1.3.4. Páginas en jQuery 12

2. Testeo, depuración y compilación de una aplicación sencilla realizada con jQuery Mobile y PhoneGap 37

1. Desarrollo de una aplicación sencilla con jQuery Mobile y PhoneGap

1.1. Introducción

En este apartado aprenderemos todos los pasos necesarios para desarrollar una aplicación multiplataforma utilizando la librería jQuery Mobile y PhoneGap.

Para empezar el proceso, podéis emplear cualquiera de las opciones de instalación planteadas anteriormente.

Para empezar recurriremos a la librería jQuery Mobile, muy útil para hacer una aplicación sencilla; es la más utilizada en el desarrollo de aplicaciones multiplataforma. Además, dispone de un amplio soporte, herramientas en línea para montar los esqueletos, gran cantidad de elementos UI, diferentes temas, etc.

Su problema principal es que, al ser tan completa, ocupa bastante espacio y puede hacer que nuestra aplicación tarde un poco más de tiempo en cargarse.

Cabe comentar que existen una serie de herramientas en línea que nos ofrecen entornos drag&drop para crear nuestro código con jQuery Mobile; algunos ejemplos son:

- **Codiqa**: http://www.codiqa.com
- **Mobjectify**: http://www.mobjectify.com/

Cada vez irán apareciendo más servicios que nos pueden ser muy útiles de cara a ahorrarnos tiempo al desarrollar nuestra aplicación. De todos modos, nosotros aprenderemos a empezar una aplicación desde cero.

1.2. Wireframes

Antes de empezar a programar es muy importante tener claro cuál es el objetivo del desarrollo, lo que supone tener muy definidas las diferentes pantallas y el funcionamiento de la app.

Para hacerlo, se utiliza lo que se denomina wireframes; hay muchas herramientas que podemos utilizar para crearlos; actualmente, existen muchos servicios webs que nos pueden facilitar el trabajo. Podéis encontrar un listado en:

http://mashable.com/2013/04/02/wireframing-tools-mobile/

http://webdesignledger.com/tools/13-super-useful-ui-wireframe-tools

También podemos utilizar programas de diseño como Photoshop o FireWorks para facilitarnos el trabajo. Si escogemos esta opción, hay plantillas GUI que podemos utilizar para ello.

Estas plantillas disponen de todos los elementos de la interfaz de un dispositivo en un fichero PSD.

Algunas de las más conocidas son las de teehan.

- **Android**: http://www.teehanlax.com/blog/android-2-3-4-gui-psd-high-density/
- **iOS**: http://www.teehanlax.com/blog/ios-6-gui-psd-iphone-5/

O incluso podemos hacer un borrador con lápiz y papel.

Lo importante es que, cuanto más definidos tengamos nuestros wireframes, menos problemas y dudas nos encontraremos a la hora del desarrollo.

1.3. Desarrollo

En este apartado desarrollaremos una pequeña aplicación multiplataforma; en nuestro caso queremos que sea compatible tanto para iOS como para Android.

Al ser una aplicación para Android, se han de tener en cuenta las diferentes medidas de pantalla, así que es importante que la interfaz de la aplicación sea capaz de funcionar con diferentes tamaños.

Será una aplicación para estudiantes de la UOC, y tendrá una primera pantalla de login para introducir los datos de acceso. Una vez validado el usuario, se accederá a la aplicación, y esta tendrá varias secciones.

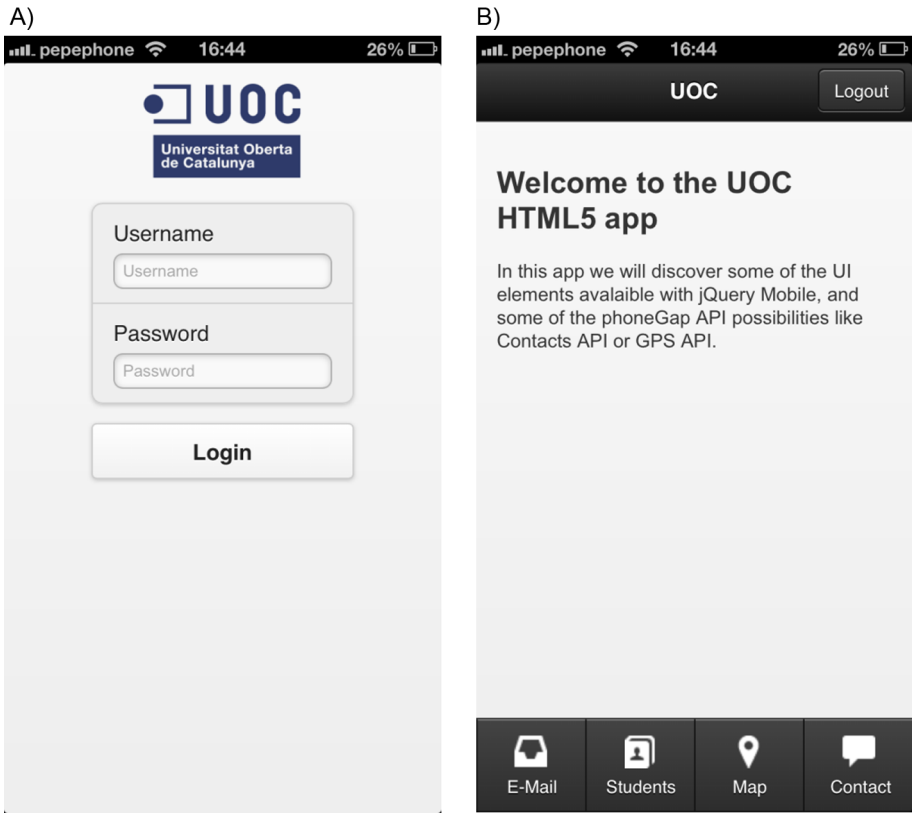
Una primera sección donde simularemos el acceso al correo de un estudiante de la UOC.

Una segunda sección que nos mostrará a los estudiantes que tengamos en nuestra agenda de teléfono; en nuestro caso, mostrará todos los contactos de la agenda.

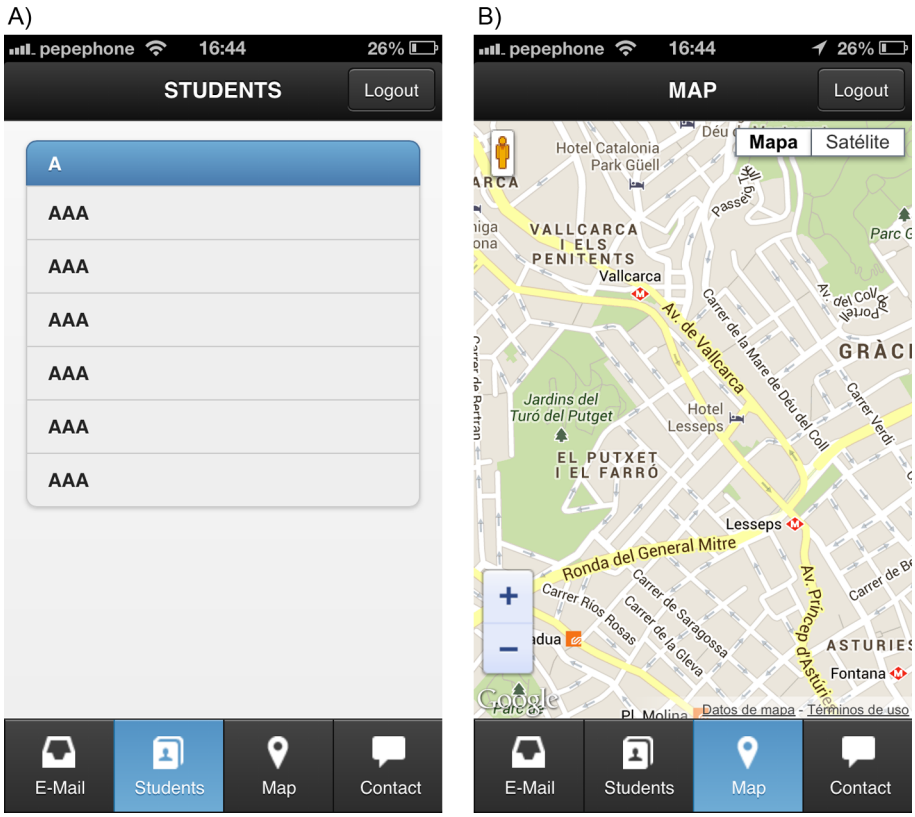
Un tercer apartado donde se nos mostrará un mapa con puntos de interés de la UOC o estudiantes; en la práctica solo mostraremos nuestra localización.

Y finalmente, un último apartado donde poder abrir la aplicación de correo para contactar con los servicios de la UOC.

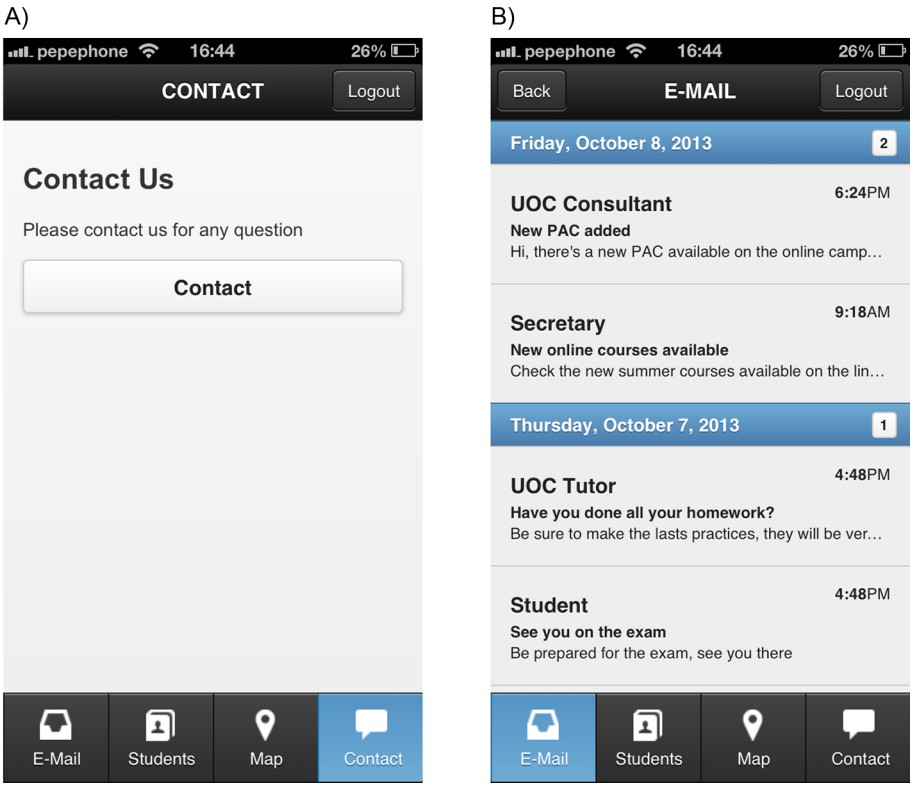
A continuación, veamos algunas pantallas con el resultado final:



A) Pantalla de entrada donde se pedirán los datos de acceso al usuario. **B)** Pantalla principal de presentación desde la cual se podrá acceder a las diferentes secciones.



A) Pantalla con un listado de estudiantes. **B)** Pantalla donde se visualizarán los principales puntos de interés para los estudiantes de la UOC.



A) Pantalla donde el estudiante podrá contactar con los servicios de la UOC. **B)** Pantalla que simulará el buzón del estudiante.

1.3.1. Preparación

A continuación, veremos algunas diferencias que tendremos que tener en cuenta dependiendo del entorno de desarrollo que hayamos escogido.

Opción A

Si hemos escogido la opción A, crearemos un proyecto nuevo en el Aptana y un nuevo archivo HTML que se llame index.html.

En el supuesto de que utilicemos el emulador Ripple, para poder utilizar la versión 2.0 de PhoneGap tendremos que hacer un pequeño arreglo.

Nos descargaremos la última versión de PhoneGap de la web http://phonegap.com/install/. Descomprimiremos el archivo descargado y dentro de la carpeta lib/android/ encontraremos el archivo cordova.js, y lo copiaremos en la raíz de nuestro proyecto.

Tendremos que tener la llamada a la librería cordova.js dentro del código del index.html.

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

Opción B

Si por el contrario habíamos escogido la opción B, aprovecharemos el proyecto creado anteriormente y borraremos todos los archivos de la carpeta assets/www. También crearemos ahora un archivo nuevo llamado index.html.

En este archivo también tendremos que tener siempre la llamada al archivo cordova.js para poder utilizar PhoneGap.

1.3.2. Código inicial

Además del archivo index.html necesitaremos crear en el mismo directorio dos directorios más, uno llamado js y otro llamado css; dentro del directorio css crearemos un documento llamado main.css donde añadiremos los estilos que vayamos creando, y dentro de la carpeta js, crearemos un fichero llamado main.js donde añadiremos las funciones javascript que necesitemos.

En suma, nos tendría que quedar la siguiente estructura:

index.html

/css/main.css

/js/main.js

Copiaremos el siguiente código dentro del archivo index.html:

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="utf-8" />
      <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1,
           minimum-scale=1, width=device-width, height=device-height" />
      <link rel="stylesheet" type="text/css" href="css/main.css" />
      <title>UOC</title>
   </head>
   <body>
      <script src="cordova.js"></script>
      <script type="text/javascript" src="js/main.js"></script>
   </body>
</html>
```

Sobre el código que hemos añadido, cabe comentar algunas cosas:

La primera línea de código es el llamado DocType; este fragmento nos indica qué tipo de documento HTML estamos tratando. En el caso del HTML5, este tiene que ser exactamente como continuación, no hace falta añadir ninguna información más.

```
<!DOCTYPE html>
```

La siguiente línea de código nos indica el comienzo de la página HTML; como parámetro le indicaremos el lenguaje de la página, en este caso dejaremos el inglés.

```
<html lang="en">
```

A continuación encontramos los tags meta; el primero nos indica la codificación de los caracteres del documento; en nuestro caso, indicaremos que se trata de UTF-8.

```
<meta charset="utf-8" />
```

El segundo tag nos interesa más; se trata del tag viewport. Es uno de los tags más importantes en el desarrollo de apps multiplataforma, en parte porque nos permite definir cómo queremos que se visualice la página en el dispositivo.

```
<meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1,
 minimum-scale=1, width=device-width, height=device-height" />
```

En nuestro caso, desactivaremos la opción de que el usuario pueda hacer zoom en la web indicando user-scalable=no, ya que tiene sentido en webs pero no en aplicaciones.

Definiremos como escala inicial el valor por defecto, es decir, initial-scale=1, por lo que los elementos tendrán el tamaño real; como escala máxima también definiremos el valor de tamaño real o maximum-scale=1, a pesar de que no podrán ni ampliar ni reducir los contenidos puesto que hemos desactivado la posibilidad de escalarlos. Y por último, queremos que el viewport ocupe toda la medida de la pantalla del dispositivo, o sea, queremos tener visible toda la web.

Con esto no fijamos ninguna medida de la pantalla sino que dependerá de nuestro código que la aplicación se adapte correctamente a las diferentes medidas de pantalla de los dispositivos que queremos soportar.

Este otro fragmento carga los estilos CSS del fichero main.css que hemos creado anteriormente dentro de la carpeta llamada css.

```
<link rel="stylesheet" type="text/css" href="css/main.css" />
```

A continuación nos encontramos el tag <body>; dentro de este tag es donde añadiremos todo el código HTML de nuestra aplicación.

Después encontramos las llamadas Javascript; justo antes de cerrar el tag </body>, podríamos colocarlos dentro del tag <header>, pero colocarlos en la parte inferior nos asegura que se hayan cargado los elementos del HTML antes de llamar a ningún elemento Javascript.

La primera llamada que nos encontramos, como hemos visto antes, es la llamada necesaria para utilizar PhoneGap; se trata de la librería cordova.js. Por defecto se ubica en el directorio raíz de nuestro proyecto, por eso no la colocaremos dentro del directorio js.

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

Y finalmente, encontramos la llamada en el fichero Javascript main.js, en el cual guardaremos todo el código Javascript de nuestra aplicación.

```
<script type="text/javascript" src="js/main.js"></script>
```

Y con esto ya tendremos la base a partir de la cual construir nuestra aplicación.

1.3.3. Librería jQuery

A continuación, añadiremos la librería jQuery Mobile a nuestra aplicación.

Una de las ventajas de esta librería es que ya dispone de gran cantidad de elementos orientados al desarrollo móvil, muchos de los cuales se adaptan de forma automática a diferentes tamaños de pantalla.

Para añadirla primero accederemos a la web de jQuery Mobile y nos descargaremos la última versión disponible.

Podríamos enlazarla de forma remota, pero esto supondría que nuestra aplicación necesitaría internet para funcionar, así que siempre es mejor si la podemos utilizar de forma local.

Colocaremos el archivo css dentro de la carpeta css y el archivo javascript dentro de la carpeta js.

Hay que destacar que, desgraciadamente, para utilizar jQuery Mobile es necesario también enlazar la librería jQuery de escritorio, lo que hace que necesitemos dos llamadas Javascript y nuestra aplicación pese bastante más.

Tendremos que añadir la siguiente llamada para el css:

```
<link rel="stylesheet" href="css/jquery.mobile-1.3.1.min.css" />
```

Y después los de Javascript:

```
<script src="js/jquery-1.9.1.min.js"></script>
<script src="js/jquery.mobile-1.3.1.min.js"></script>
```

Nota

El número de versión no tiene por qué ser el mismo, ya que pueden haber salido nuevas versiones.

Tendríamos que tener un código como el siguiente:

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="utf-8" />
      <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1,
              minimum-scale=1, width=device-width, height=device-height" />
      <link rel="stylesheet" href="css/jquery.mobile-1.3.1.min.css" />
      <link rel="stylesheet" type="text/css" href="css/main.css" />
      <title>UOC</title>
   </head>
   <body>

      <script src="cordova.js"></script>
      <script src="js/jquery-1.9.1.min.js"></script>
      <script src="js/jquery.mobile-1.3.1.min.js"></script>
      <script type="text/javascript" src="js/main.js"></script>
   </body>
</html>
```

1.3.4. Páginas en jQuery

Programando siempre es recomendable tener a mano la página de documentación; en el caso de jquery Mobile, podemos encontrar demos en:

http://view.jquerymobile.com/1.3.2/dist/demos/

Y documentación en:

http://api.jquerymobile.com/

Para empezar, crearemos las páginas necesarias para nuestra aplicación. Para crear diferentes páginas en jQuery Mobile, se utiliza el atributo:

```
data-role="page"
```

Añadiendo este atributo a nuestro elemento estamos indicando que se trata de una página.

Necesitaremos para empezar 5 páginas, por lo que añadiremos el siguiente código a continuación del tag <body>.

```
<div data-role="page" id="login"></div>
<div data-role="page" id="home"></div>
<div data-role="page" id="mail"></div>
<div data-role="page" id="students"></div>
<div data-role="page" id="map"></div>
<div data-role="page" id="contact"></div>
```

Por defecto jQuery Mobile nos abrirá la primera página que encuentre; en este caso la página de login; por eso la tendremos la primera de todas.

Las páginas de jQuery Mobile se estructuran en tres partes diferenciadas, el header, el content y el footer.

Página de login

Empezaremos con la página de login. En nuestro caso, no funcionará de verdad puesto que requeriría de código por parte del servidor, con lo que no entraremos, solo la mostraremos y haciendo tapón sobre el botón, accederemos a la aplicación.

En la página de login añadiremos una imagen con el logo de la UOC en la parte superior y un pequeño formulario con dos campos para entrar los datos de acceso, cada uno con un label o etiqueta, y finalmente un botón para acceder.

Añadiremos todo dentro de una etiqueta del tipo content, pues esta pantalla no tiene barras ni superior ni inferior.

Empezaremos solo añadiendo la imagen:

```
<div data-role="page" id="login">
   <div data-role="content">
      <div id="logo">
         <img src="img/uoc.png" alt="UOC">
      </div>
   </div>
</div>
```

Seguidamente continuaremos con el formulario; para hacerlo más atractivo visualmente utilizaremos el componente listview, que nos permite hacer agrupaciones de elementos en formato de lista. En nuestro caso solo tendremos dos líneas, una por cada uno de los campos del formulario.

Añadiremos el código del formulario a continuación del contenedor de la imagen.

El código del formulario será el siguiente:

```
<form>
   <ul data-role="listview" data-inset="true">
      <li data-role="fieldcontain">
         <label for="f_username">Username</label>
         <input name="f_username" id="f_username" placeholder="Username" value="" type="text">
      </li>
      <li data-role="fieldcontain">
         <label for="f_password">Password</label>
         <input name="f_password" id="f_password" placeholder="Password" value="" type="password">
      </li>
   </ul>
   <a data-role="button" href="#home">Login</a>
</form>
```

El listview suele ser un tag del tipo <ul> con tags <li> para cada una de las entradas en la lista; en nuestro caso habrá dos <li> anidados.

Estos <li> tienen la propiedad data-role=”fieldcointain”, la cual nos indica que se trata de un campo de formulario; esto hace que se muestre correctamente el <label> al lado del <input>.

Finalmente, añadimos el botón de login; para hacer un botón con jQuery Mobile, solo tenemos que añadir data-role=”button” a un enlace. En este caso, el enlace será #home, que será la siguiente página donde queramos acceder.

Cabe comentar también el parámetro data-inset=”true”; este nos permite que los componentes no ocupen todo el espacio horizontal, sino que se agrupen en el centro y se muestren esquinas redondeadas.

A continuación podemos probar lo que llevamos hasta ahora; utilizando el Aptana podemos visualizar la web en Chrome con la opción de Run (siempre que lo hayamos configurado antes). Si utilizamos otro IDE o programa, es cuestión de abrir la web en el navegador. Si usamos la opción CLI podemos ejecutar la web en el navegador mediante el pedido:

```
cordova serve <PLATFORM>
```

Veremos algo parecido en el caso siguiente:

Captura del emulador Ripple visualizando la pantalla de acceso

Como podemos comprobar, la imagen no se escala según el dispositivo y el formulario ocupa todo el ancho de la pantalla. Añadiremos unos cambios al archivo css para resolver estos problemas.

Para empezar centraremos todo el contenido y ocuparemos solo el 60% del espacio. Con div#login estamos indicando que es un div con id="login" y con [data-role="content"] indicamos que se trata del contenedor content. La instrucción margin: 0 auto nos centra el contenedor ya que establece un margen inferior y superior de 0, pero lo establece como automático por derecha e izquierda.

```
div#login [data-role="content"] {
   width: 60%;
   margin: 0 auto;
}
```

El logo también lo escalaremos según la medida del dispositivo; para hacerlo hay un pequeño truco: indicar que la imagen llene todo el espacio disponible con la instrucción:

```
div#login div#logo img {
   max-width: 100%;
}
```

Y después indicar el tamaño del contenedor de la imagen con un porcentaje del ancho disponible.

En nuestro caso, estableceremos que ocupe el 60%. Este truco CSS es el que se utiliza para escalar las imágenes en páginas responsive o fluidas, para que las imágenes se adapten también al tamaño de los dispositivos.

```
div#login div#logo {
   width: 60%;
   margin: 0 auto;
}
```

Añadiremos un último retoque que nos reduzca el tamaño de la imagen en pantallas más grandes; para hacerlo utilizaremos las media queries; estas son instrucciones css que se ejecutan solo si se cumple el parámetro que le indicamos.

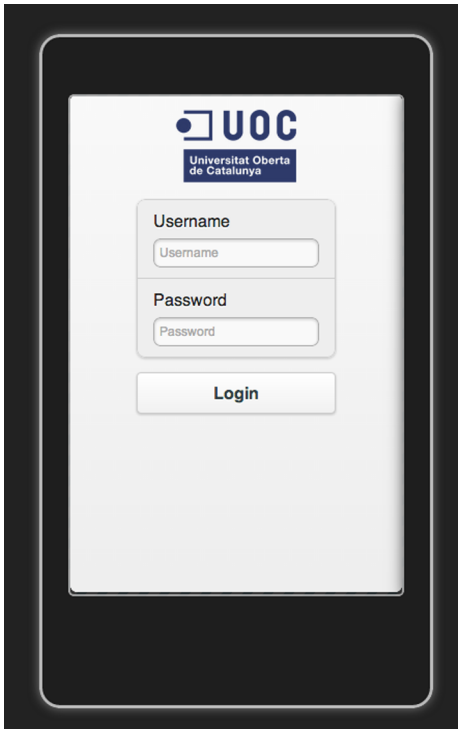
En nuestro caso, el logo solo ocupará el 30% en el supuesto de que la pantalla sea superior de 480px; por lo tanto, en tabletas o pantallas más grandes.

```
@media screen and (min-width: 480px){
   div#login div#logo {width: 30%;}
}
```

Las media queries son las instrucciones CSS que se utilizan con lo que se denomina Responsive Design; con esto, lo que conseguimos es establecer diferentes normas de visualización según la medida del dispositivo. Así, podemos visualizar la aplicación de diferente manera según si es un dispositivo con una pantalla pequeña o una tableta.

Con esto obtendremos el siguiente resultado:

Captura del emulador Ripple visualizando la pantalla de acceso



Página de home

Seguiremos con la siguiente página: la de home.

En esta página añadiremos, además del content, una barra de navegación superior o navbar al header, y una barra inferior para movernos entre las secciones o tabbar.

Antes que nada, añadiremos el header con un botón que nos permita volver a la pantalla de login.

```
<div data-role="header" data-position="fixed" data-id="header1">
   <h1>UOC</h1>
   <a href="#login" class="ui-btn-right">Logout</a>
</div>
```

El parámetro data-position="fixed" nos indica que queremos que la barra superior se mantenga fija en la parte superior. El parámetro data-id identifica con un nombre el elemento; de este modo, jQuery no animará el elemento al cambiar de página.

El tag <h1> nos indicará el título que aparecerá en la barra superior. El botón en la parte derecha lo conseguiremos con la propiedad class="ui-btn-right"; como enlace pondremos el id de la página de login.

A continuación, añadiremos la etiqueta de content; en este añadiremos el texto a mostrar, por ejemplo, el siguiente.

```
<div data-role="content">
      <h2>Welcome to the UOC HTML5 app</h2>
      <p>In this app we will discover some of the UI elements avalaible with jQuery Mobile,
and some of the PhoneGap API possibilities like Contacts API or GPS API.</p>
</div>
```

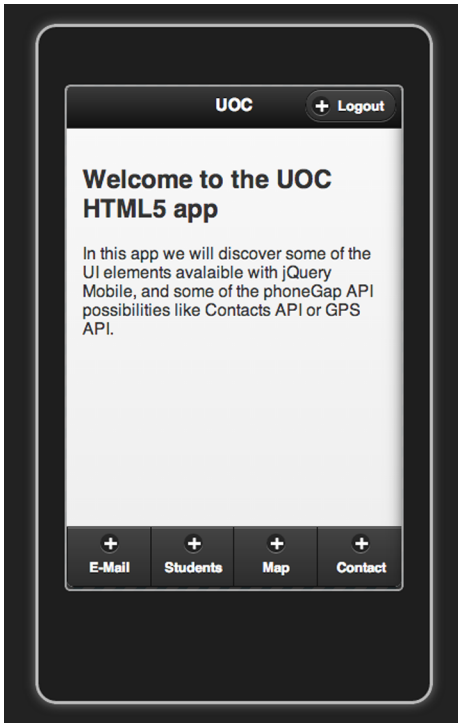
Y finalmente, el footer, donde añadiremos la barra inferior con los diferentes botones. Para crear esta barra tenemos que seguir la siguiente estructura.

```
<div data-id="navbar" data-role="footer" data-position="fixed" data-fullscreen="false">
   <div data-role="navbar">
      <ul>
         <li><a href="#mail" id="tab1" data-icon="custom">EMail</a></li>
         <li><a href="#students" id="tab2" data-icon="custom">Students</a></li>
         <li><a href="#map" id="tab3" data-icon="custom">Map</a></li>
         <li><a href="#contact" id="tab4" data-icon="custom">Contact</a></li>
      </ul>
   </div>
</div>
```

Comentaremos algunas de las propiedades; data-id=”navbar” nos permite darle un id a este elemento; esto nos es útil porque, de este modo, jQuery Mobile no lo animará cuando la siguiente página tenga el mismo elemento también. Al estar la barra de navegación no queremos que desaparezca o se mueva al cambiar de página; el parámetro de data-position=”fixed” nos indica que queremos que esta barra se mantenga siempre pegada a la parte superior. La propiedad data-fullscreen=”false” nos indica que queremos tener siempre visible esta barra; metiéndolo en true, se escondería al tocar el contenido de la app.

La página nos tendría que quedar así:

Captura del emulador Ripple visualizando la pantalla de bienvenida



Página de correo

En esta página seguiremos teniendo la misma barra superior; por lo tanto, el header será idéntico al anterior, solo cambiaremos el título:

```
<div data-role="header" data-position="fixed" data-id="header1">
   <h1>E-MAIL</h1>
   <a href="#login" class="ui-btn-right">Logout</a>
</div>
```

En cuanto al content, tendremos un listado de buzones como si se tratara de una cuenta de correo. Todas las opciones enlazan a una página interior llamada #message que nos mostrará un listado de correos. Con el parámetro data-transition=”slide” conseguimos el efecto de animación lateral, que recalca que se trata de una página interior y no una sección principal.

```
<div data-role="content">
   <ul data-role="listview" data-inset="true">
      <li>
         <a href="#message" data-transition="slide">Inbox <span class="ui-li-count">12</span></a>
      </li>
      <li>
         <a href="#message" data-transition="slide">Outbox <span class="ui-li-count">0</span></a>
      </li>
      <li>
         <a href="#message" data-transition="slide">Drafts <span class="ui-li-count">4</span></a>
      </li>
```

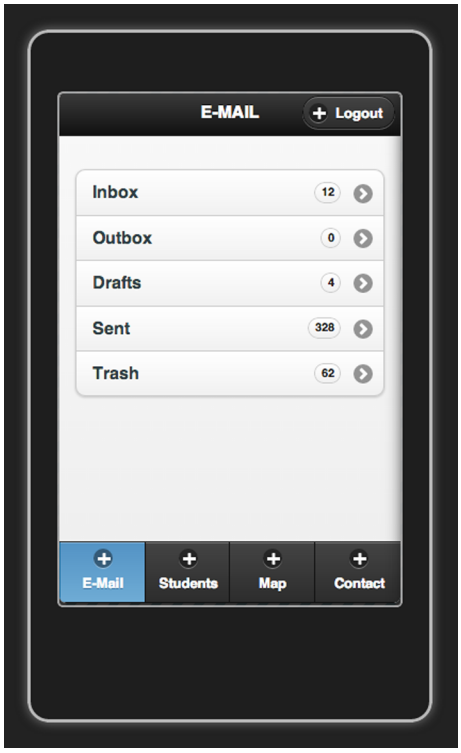
```
        <li>
           <a href="#message" data-transition="slide">Sent <span class="ui-li-count">328</span></a>
        </li>
        <li>
           <a href="#message" data-transition="slide">Trash <span class="ui-li-count">62</span></a>
        </li>
     </ul>
  </div>
```

Y por último, encontramos el footer, que es prácticamente idéntico a la página de #home, menos por el detalle de que queremos que aparezca seleccionado el primer botón de navegación, indicando de este modo la sección donde nos encontramos; para hacerlo, solo tenemos que añadir la clase ui-btn-active. Para mantener el estado, si cambiamos de sección también añadiremos la clase ui-state-persist.

```
<li>
   <a href="#mail" id="tab1"  class="ui-btn-active ui-state-persist" data-icon="custom">E-Mail</a>
</li>
```

El resultado tendría que ser como este:

Captura del emulador Ripple visualizando la sección de acceso al correo



Página de mensaje

Para la página de correo tenemos que hacer un pequeño cambio en el header, puesto que queremos que nos aparezca el botón para volver atrás.

Para hacerlo, tenemos que añadir el parámetro data-add-back-btn="true" al tag de la página message.

```
<div data-role="page" id="message" data-add-back-btn="true">
```

De forma automática se hace la animación inversa al apretar el botón atrás, por lo que, si hemos abierto esta página con un slide a la derecha, al volver atrás hará un slide a la izquierda. Si nos fijamos, no se hace slide de la barra inferior, puesto que le hemos dado la propiedad de data-id="navbar" en cada una de las páginas.

En cuanto al header, dejaremos el mismo header que en la página de email.

```
<div data-role="header" data-position="fixed" data-id="header1">
   <h1>E-MAIL</h1>
   <a href="#login" data-icon="unlink" class="ui-btn-right">Logout</a>
</div>
```

Respecto al contenido de esta página, será otro listview, esta vez sin indentación; por lo tanto, ocupará todo el ancho de la pantalla. Y con un nuevo elemento llamado list-divider que nos permite añadir divisiones a la lista, en nuestro caso para separar los correos por días.

El código resultante será:

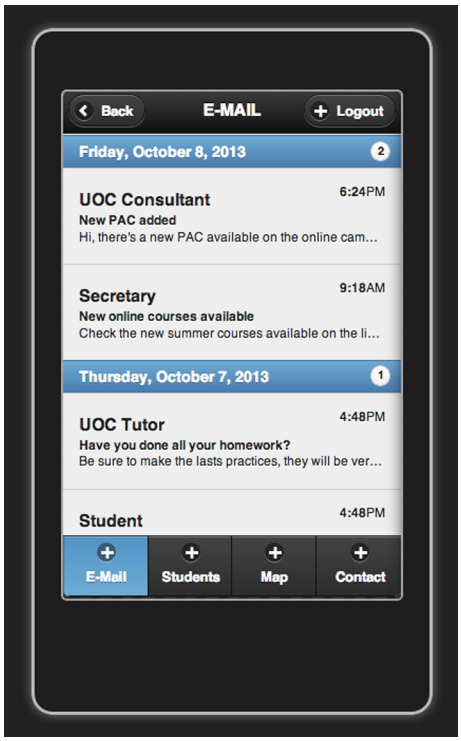
```
<div data-role="content">
   <ul data-role="listview" data-inset="false">
      <li data-role="list-divider">
         Friday, October 8, 2013 <span class="ui-li-count">2</span>
      </li>
      <li>
         <h2>UOC Consultant</h2>
         <p><strong>New PAC added</strong></p>
         <p>Hi, there's a new PAC available on the online campus, check it out.</p>
         <p class="ui-li-aside"><strong>6:24</strong>PM</p>
      </li>
      <li>
         <h2>Secretary</h2>
         <p><strong>New online courses available</strong></p>
         <p>Check the new summer courses available on the link below.</p>
         <p class="ui-li-aside"><strong>9:18</strong>AM</p>
      </li>
      <li data-role="list-divider">
         Thursday, October 7, 2013 <span class="ui-li-count">1</span>
```

```
        </li>
        <li>
           <h2>UOC Tutor</h2>
           <p><strong>Have you done all your homework?</strong></p>
          <p>Be sure to make the lasts practices, they will be very important for the final exam.</p>
           <p class="ui-li-aside"><strong>4:48</strong>PM</p>
        </li>
        <li>
           <h2>Student</h2>
           <p><strong>See you on the exam</strong></p>
           <p>Be prepared for the exam, see you there</p>
           <p class="ui-li-aside"><strong>4:48</strong>PM</p>
        </li>
     </ul>
  </div>
```

El footer no varía en esta página, ya que es idéntico al de la página message.

Nos tendría que quedar un resultado como el siguiente:

Captura del emulador Ripple visualizando la pantalla del buzón



Página de students

El header es igual que el de las secciones anteriores, pero con diferente título.

```
  <div data-role="header" data-position="fixed" data-id="header1">
     <h1>STUDENTS</h1>
```

```
    <a href="#login" data-icon="unlink" class="ui-btn-right">Logout</a>
</div>
```

En la sección de content, añadiremos un listview; en nuestro caso, lo añadiremos vacío sin ningún elemento, puesto que se encargará de llenarlo una función Javascript. Al listado le daremos un id student-list para poderlo identificar más fácilmente y algunas propiedades como data-autodividers, que nos genera de forma automática separadores según la inicial. Con data-filter a false le indicamos que no queremos buscador en la parte superior.

```
<div data-role="content">
   <ul id="student-list" data-role="listview" data-autodividers="true" data-filter="false"
         data-inset="true"></ul>
</div>
```

El footer se mantiene igual, pero cambiaremos el botón seleccionado por el de Students.

```
<div data-id="navbar" data-role="footer" data-position="fixed" data-fullscreen="false">
   <div data-role="navbar">
      <ul>
         <li><a href="#mail" id="tab1" data-icon="custom">E-Mail</a></li>
         <li><a href="#students" id="tab2"  class="ui-btn-active ui-state-persist"
                  data-icon="custom">Students</a></li>
         <li><a href="#map" id="tab3" data-icon="custom">Map</a></li>
         <li><a href="#contact" id="tab4" data-icon="custom">Contact</a></li>
      </ul>
   </div>
</div>
```

Y seguidamente, veremos las funciones Javascript necesarias para poder visualizar los contactos de la agenda en este listado.

En PhoneGap hay una función que nos indica cuándo el dispositivo ha acabado de cargar toda la información; es después de recibir esta función cuando podemos empezar a trabajar; no nos podemos fiar del onLoad habitual utilizado normalmente con el desarrollo web.

Esta función es “deviceready”. Añadiremos el siguiente fragmento de código al archivo main.js, para recibir esta notificación.

```
document.addEventListener("deviceready", onDeviceReady, false);


function onDeviceReady(){
   alert("deviceready");
}
```

Si ejecutamos la aplicación, tanto si estamos utilizando Chrome con Ripple o una herramienta nativa, veremos el mensaje "devideready" cuando esté listo el dispositivo. Esto nos indica que PhoneGap se ha inicializado correctamente y está preparado para recibir nuestras instrucciones.

A continuación, añadiremos un código en lugar de la llamada alert para que se ejecute al iniciarse la aplicación.

```
function onDeviceReady(){

   /*Event called when Students page is initialized*/
   $('#students').on('pageshow', function(event) {
      if (typeof(ContactFindOptions) == 'function'){

         /*set contact options*/
         var options = new ContactFindOptions();
         options.filter="";          // empty search string returns all contacts
         options.multiple=true;      // return multiple results
         filter = ["displayName"];   // return contact.displayName field

         /*asks for addressbook*/
         navigator.contacts.find(filter, successAddressFunction, errorAddressFunction, options);
      }
   });

}
```

Vamos por partes: la primera línea nos indica que queremos que se ejecute esta función solo cuando se visualice la página de Students. Es decir, con el event pageshow.

Cabe notar que podríamos pedir permiso al usuario al abrirse la aplicación, pero no tiene mucho sentido si el usuario no quiere utilizar esta sección, por lo que solo pediremos permiso al abrir la página de Students.

```
$('#students').on('pageshow', function(event) {
```

La segunda línea no es necesaria para el funcionamiento, pero nos permite visualizar la página sin PhoneGap y que no dé ningún error. Básicamente comprueba que exista la función ContactFindOptions, antes de llamarla.

```
if (typeof(ContactFindOptions) == 'function'){
```

Ahora configuramos las opciones con que queremos hacer la llamada, por ejemplo, queremos múltiples entradas y que nos devuelva solo el nombre.

```
/*set contact options*/
```

```
        var options = new ContactFindOptions();
        options.filter="";          // empty search string returns all contacts
        options.multiple=true;      // return multiple results
        filter = ["displayName"];   // return contact.displayName
```

Por último, llamamos a la función que nos devolverá el listado de contactos, le pasamos las opciones y el nombre de las funciones que llamará si el usuario ha dado permiso o si no lo ha dado. También es posible que el dispositivo no disponga de agenda, por lo que también se llamará a la función de error con el código de error.

```
/*asks for addressbook*/
navigator.contacts.find(filter, successAddressFunction, errorAddressFunction, options);
```

Enlaces recomendados

Podéis encontrar toda la documentación del API de PhoneGap en la página:

http://docs.phonegap.com/en/3.0.0/index.html

Y del API de Contacts en:

http://docs.phonegap.com/en/3.0.0/cordova_contacts _contacts.md.html#Contacts

Seguidamente, veremos las funciones o *callbacks* que se llaman para recibir el listado de contactos o cuando se da un error.

La primera función recibe un array de contactos, hace un bucle del tipo for; primero se asegura de que dispone de nombre o displayName, después se les añade el tag <li> y por último los añade al elemento listview student-list.

Una vez acabado el bucle, refrescamos la información del listview.

```
//function when addressbook permission is granted
function successAddressFunction(contacts) {
   for (var i=0; i<contacts.length; i++) {
      if (contacts[i].displayName) {  //check only for contacts with a displayName

         var content = "<li>"+contacts[i].displayName+"</li>";//set the new list item html <li>
         $("#student-list").append( content );//add to the <ul> list
         }
      }

   $("#student-list").listview('refresh');//refresh the list element

}
```

Finalmente, acabaremos con la función de error, en este caso no implementaremos mucho código, simplemente esconderemos la lista y mostraremos un alert con el mensaje de error.

```
//function when an error happens when accessing the addressbook
function errorAddressFunction(err){

   $("#student-list").css("display","none");
   alert("No addressbook found");
}
```

Si ejecutamos la aplicación se nos tendrían que mostrar algunos contactos; el Ripple ya incluye algunos contactos por defecto, pero los simuladores nativos seguramente tengan la agenda vacía, por lo que tendremos que añadir algunos desde el mismo simulador.

Tendríamos que ver un resultado como el siguiente:

Captura del emulador Ripple visualizando la pantalla con el listado de los contactos de la agenda

Página de mapa

En esta página añadiremos un mapa donde aparezca nuestra posición. Para hacerlo, haremos uso de la librería de Google Maps.

Enlace recomendado

Podemos encontrar más información aquí:
https://developers.google.com/maps/documentation/javascript/?hl=es

Necesitaremos añadir la librería Javascript de Google Maps; esto supone añadir la llamada dentro del HTML junto a las otras llamadas a las librerías jQuery.

```
<script src="http://maps.googleapis.com/maps/api/js?sensor=true"></script>
```

Respecto al código de jQuery Mobile, en el header solo cambiaremos el título, como hemos hecho anteriormente.

```
<div data-role="header" data-position="fixed" data-id="header1">
   <h1>MAP</h1>
   <a href="#login" data-icon="unlink" class="ui-btn-right">Logout</a>
</div>
```

En el content sí que hemos de añadir el código donde cargaremos el mapa; básicamente se trata de un donde cargaremos el mapa vía javascript.

```
<div data-role="content" id="map_content">
   <div id="map_canvas"></div>
</div>
```

El footer se queda igual que en las otras secciones, pero indicando la opción del mapa como la activa.

En cuanto al Javascript, añadiremos otro fragmento de código dentro de la función onDeviceReady, en este caso para cargar el mapa cuando accedamos a la página del mapa.

```
/*Event called when Map page is shown*/
$('#map').on('pageshow', function(event) {
      /*asks for current location*/
      navigator.geolocation.getCurrentPosition(successMapFunction, errorMapFunction,{
      maximumAge: 300, timeout: 500, enableHighAccuracy: true });
});
```

Esta función pide el permiso al usuario para compartir su ubicación; si lo acepta se llama a la función successMapFunction; si no lo acepta o hay un error, se llamará a errorMapFunction.

Respecto a la función successMapFunction, recibimos la posición y pasamos los datos a una nueva función creada por nosotros.

```
//get the user position
function successMapFunction(position) {
```

```
    /*user GPS position*/
    var lat = position.coords.latitude;
    var lng = position.coords.longitude;

    initGoogleMaps(lat, lng);
}
```

La función de error, en vez de pasar los datos de posición, pasará unos datos de posición por defecto; en este caso, la posición de la sede de la UOC.

```
//set the default position
function errorMapFunction(err) {

    /*default UOC position*/
    var lat = 41.414703;
    var lng = 2.132653;

    initGoogleMaps(lat, lng);
}
```

Por último, vemos la función initGoogleMaps, la cual es la encargada de dibujar el mapa. Básicamente crea un mapa en la posición que recibe, establece las opciones del mapa a mostrar y crea el mapa en el elemento con id map_canvas. Después coloca un pin o marker en la posición exacta donde nos encontramos o en la de la sede de la UOC si es el caso.

```
function initGoogleMaps(lat, lng) {

    /*init the google maps*/
    latlng = new google.maps.LatLng(lat, lng);

    /*default google maps options*/
    var options = {
        zoom : 15,
        center : latlng,
        mapTypeId : google.maps.MapTypeId.ROADMAP
    };

    /*create google maps map on map_canvas element*/
    map = new google.maps.Map(document.getElementById('map_canvas'), options);

    /*add a marker to the map position*/
    new google.maps.Marker({
        map : map,
        animation : google.maps.Animation.DROP,
        position : latlng
    });
```

```
}
```

Lo más probable es que, si probamos la aplicación, todavía no veamos el mapa o no se muestre correctamente. Para mostrarlo correctamente, tendremos que hacer uso de nuevo del CSS.

Para empezar, forzaremos map_canvas para que ocupe todo el espacio disponible, estableciendo la anchura y la altura en 100%.

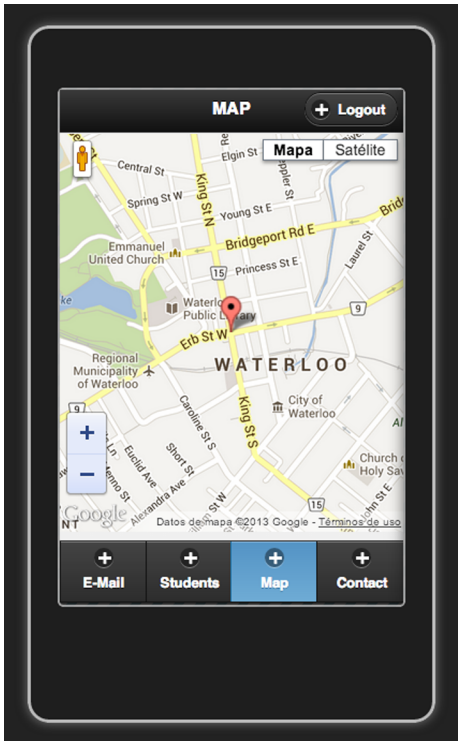
Para que funcione correctamente, tendremos que establecer la posición como absolute; esto supone que la posición va en referencia de la página y no del elemento que lo contenga. Así que fijaremos la nueva posición en parámetros absolutos, el bottom a 66px, que es la altura de la barra inferior, y el top a 43px, que es la altura de la barra superior.

```
div#map_canvas{width:100%;height:100%;}
div#map_content {
   position: absolute;
   padding: 0;
   bottom: 66px;
   top: 43px;
   width: 100%;
}
```

Con esto ya tendríamos que ver el mapa correctamente.

Captura del emulador Ripple visualizando un mapa donde poder localizar nuestra posición y

los puntos de interés para un estudiante de la UOC



Página de Contact

Por último, tenemos la página de Contact, que será la más sencilla de todas.

En el header solo cambiaremos el título por el de Contact y el footer solo cambiará el botón seleccionado.

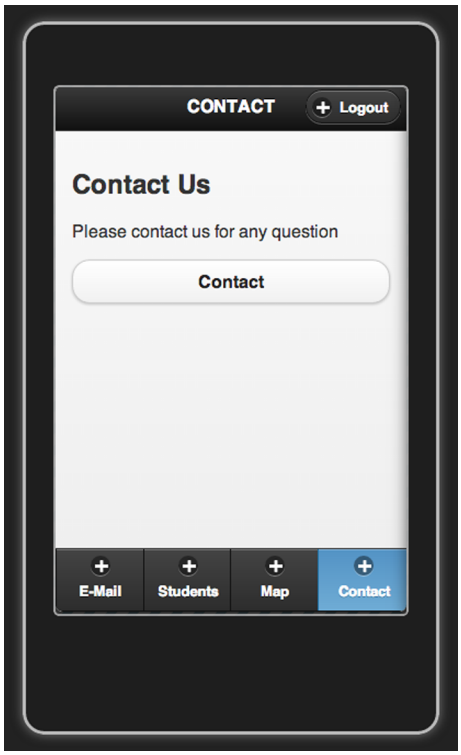
Solo comentaremos la parte de content; constará de una serie de elementos de texto y un botón con un enlace de email.

```
<div data-role="content">
   <h2>Contact Us</h2>
   <p>Please contact us for any question</p>
   <a href="mailto:app@uoc.edu" data-role="button">Contact</a>
</div>
```

El dispositivo es suficientemente inteligente para detectar este enlace de email y abrir la herramienta para escribir emails de forma automática.

El resultado quedaría así:

Captura del emulador Ripple visualizando la pantalla para contactar con los servicios de la UOC



Diseño, temas y mejoras CSS

Para modificar el diseño de nuestra aplicación, jQuery Mobile dispone de diferentes opciones; para empezar, esta dispone de cuatro temas por defecto, los cuales básicamente son el mismo tema pero con diferentes colores; no es mucho, pero es la opción más sencilla.

También existe una herramienta llamada themeRoller (http://jquerymobile.com/themeroller/index.php) que permite crear nuestros propios temas y añadirlos a nuestra aplicación. También existen otros muchos temas de terceros que nos pueden ser útiles para mejorar nuestra aplicación visualmente; algunos simulan interfaces, como la de iOS y Android, con muy buenos resultados.

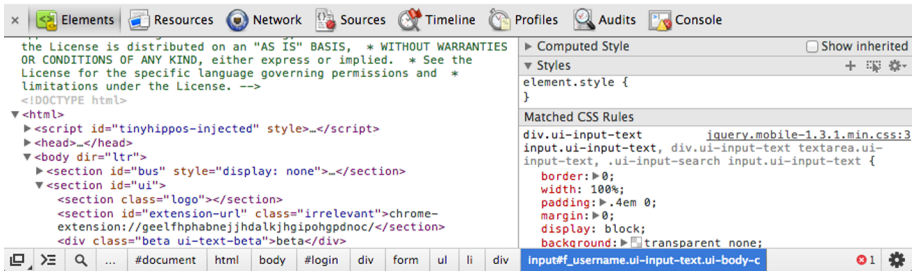
En nuestro caso, haremos cambios directamente sobre los elementos de jQuery Mobile, por lo que con unas pocas líneas CSS podremos adaptar jQuery Mobile a nuestras necesidades.

Para hacerlo, necesitaremos aprender a utilizar el Inspector Web o barra de desarrolladores de nuestro navegador. El objetivo es ver el id o class de los elementos a los que queremos cambiar el aspecto, para poder aplicar nuevas normas CSS.

Abriremos el navegador Chrome y activaremos la herramienta de desarrolladores en Ver > Opciones para desarrolladores > Herramienta para desarrolladores.

Se trata de una barra desarrollada en el proyecto WebKit, por lo que es prácticamente idéntica en Chrome y Safari; además, este componente se utiliza en programas de debugging remoto como Weinre, así que nos será muy útil aprender a utilizarlo.

Captura del Inspector Web incluido en el navegador Chrome



Esta herramienta nos será de gran utilidad ya que dispone de un gran número de opciones, como el inspector de elementos, la visualización de la red o carga de elementos, la consola Javascript, etc.

Dentro del primer apartado, llamado Elementos, pulsando el icono de la lupa de la parte inferior veremos que podemos seleccionar cualquier elemento de la página, y nos mostrará el código HTML y los estilos asociados en la parte derecha.

Siguiendo este paso, podemos modificar los estilos por defecto de jQuery Mobile para adaptarlos a nuestras necesidades.

Empezaremos estableciendo unas imágenes para la barra inferior; para hacerlo, guardaremos las imágenes en una carpeta llamada /img dentro del proyecto y escribiremos los siguientes estilos.

```
/*set icon for tab1*/
div [data-role="navbar"] a#tab1 span.ui-icon {background-image: url('../img/inbox.png');}

/*set icon for tab2*/
div [data-role="navbar"] a#tab2 span.ui-icon {background-image: url('../img/addressbook.png');}

/*set icon for tab3*/
div [data-role="navbar"] a#tab3 span.ui-icon {background-image: url('../img/maps.png');}

/*set icon for tab4*/
div [data-role="navbar"] a#tab4 span.ui-icon {background-image: url('../img/comments.png');}
```

También modificaremos el color de la barra superior e inferior; para hacerlo, añadiremos el siguiente código; para cambiar el color de la barra inferior también necesitaremos cambiar el color hover (acción cuando se pasa por encima del elemento) y el estilo de cuando está activo.

```
/*top bar color*/
.ui-bar-a {background-image: linear-gradient(#2e3455,#2e3455);}

/*tabbar color*/
.ui-btn-up-a{background-image: linear-gradient(#2e3455,#2e3455);font-weight:normal !important;}
.ui-btn-hover-a{background-image: linear-gradient(#3d4570,#3d4570);font-weight:normal !important;}

/*tabbar active color*/
.ui-btn-active{background-image: linear-gradient(#5393c5,#6facd5) !important;}
```

Seguidamente, cambiaremos el redondeo de todos los elementos con una sola instrucción para hacerlos más cuadrados.

```
/*less corner radius*/
.ui-btn-corner-all{border-radius:0.3em;}
```

Cambiaremos también el tipo de letra por otro diferente y reduciremos un poco el tamaño.

```
/*change app font family and reduces size*/
body .ui-body-c{
   font-family:"Arial","Lucida Grande","Helvetica";
   font-size:90%;
}
```

También eliminaremos los iconos que aparecen junto a los botones de la barra superior; para hacerlo, añadiremos el siguiente fragmento de código.

```
/*hides topbar icons*/
.ui-header .ui-btn-icon-left .ui-btn-inner{padding-left:11px;}
.ui-btn-right .ui-icon,.ui-btn-left .ui-icon{display:none;}
```

Y finalmente, aumentaremos el tamaño por defecto de los iconos de la barra inferior, para que destaquen más.

```
/*change the padding of the tabbar items*/
div [data-role="navbar"] .ui-btn .ui-btn-inner {
   padding-top: 40px !important;
}

/*make the tabbar icons bigger*/
div [data-role="navbar"] .ui-btn .ui-icon {
```

```
    width: 45px !important;
    height: 35px !important;
    margin-left: -24px !important;
    box-shadow: none !important;
    border-radius: none !important;
}

/*set tabbar icons*/
div [data-role="navbar"] span.ui-icon {
    background-color: transparent;
    box-shadow: none;
    background-repeat: no-repeat;
    background-position: center;
}
```

Archivo config.xml

Por último, nos faltará acabar de definir la configuración de la aplicación PhoneGap; esto implica definir el archivo config.xml; este archivo XML nos permite configurar cosas bastante importantes, como por ejemplo el nombre de la aplicación, la descripción, los iconos, los plugins utilizados y las preferencias específicas de las plataformas. Si utilizamos el servicio de PhoneGap Build Service, disponemos de la opción que nos genere el mismo servicio por parte de este archivo o también nos permite subirlo de forma manual.

Si no utilizamos el PhoneGap Build Service, lo tendremos que editar manualmente; para hacerlo abriremos el archivo generado al crear la aplicación PhoneGap, o si no tenemos ninguno, crearemos uno nuevo.

La primera línea del documento nos indica que se trata de un fichero XML.

```
<?xml version="1.0" encoding="UTF-8"?>
```

El siguiente elemento que nos encontramos es el <widget>; en este definiremos el identificador de nuestra aplicación y la versión de esta.

Un ejemplo sería:

```
<widget
xmlns     = "http://www.w3.org/ns/widgets"
xmlns:gap = "http://phonegap.com/ns/1.0"
id        = "net.pegpeg.uoc"
version   = "1.0.0">
```

A continuación encontramos los elementos <name>, <description> y <author> con los que definir el nombre de la aplicación que queremos que aparezca; la descripción de esta sí, si el desarrollador lo desea.

```
<name>UOC</name>
<description>This is the brand new UOC multiplatform app.</description>
<author href="http://pegpeg.net" email="info@pegpeg.net">PegPeg</author>
```

A continuación, encontramos los tags <feature> con los cuales tenemos que indicar los permisos que requiere nuestra aplicación para funcionar. En nuestro caso, serán tres.

```
<feature name="http://api.phonegap.com/1.0/device" />
<feature name="http://api.phonegap.com/1.0/contacts"/>
<feature name="http://api.phonegap.com/1.0/geolocation"/>
```

Después podemos definir las preferencias de nuestra aplicación; las hay comunes y específicas para una plataforma en concreto. Para la aplicación que hemos desarrollado, podríamos establecer las siguientes:

```
    <preference name="phonegap-version"           value="2.9.0" />
    <!-- all: current version of PhoneGap -->
    <preference name="orientation"                value="portrait" />
    <!-- all: default means both landscape and portrait are enabled -->
    <preference name="target-device"              value="universal" />
    <!-- all: possible values handset, tablet, or universal -->
    <preference name="fullscreen"                 value="true" />
    <!-- all: hides the status bar at the top of the screen -->
    <preference name="webviewbounce"              value="false" />
    <!-- ios: control whether the screen 'bounces' when scrolled beyond the top -->
    <preference name="prerendered-icon"           value="true" />
    <!-- ios: if icon is prerendered, iOS will not apply it's gloss to the app's icon on the
         user's home screen -->
    <preference name="stay-in-webview"            value="false" />
    <!-- ios: external links should open in the default browser, 'true' would use the webview the
         app lives in -->
    <preference name="ios-statusbarstyle"         value="black-opaque" />
    <!-- ios: black-translucent will appear black because the PhoneGap webview doesn't go beneath
        the status bar -->
    <preference name="detect-data-types"          value="false" />
    <!-- ios: controls whether data types (such as phone no. and dates) are automatically turned
         into links by the system -->
    <preference name="exit-on-suspend"            value="false" />
    <!-- ios: if set to true, app will terminate when home button is pressed -->
    <preference name="show-splash-screen-spinner" value="true" />
    <!-- ios: if set to false, the spinner won't appear on the splash screen during app loading -->
    <preference name="auto-hide-splash-screen"    value="true" />
```

```
    <!-- ios: if set to false, the splash screen must be hidden using a JavaScript API -->
    <preference name="android-minSdkVersion"      value="7" />
   <!-- android: MIN SDK version supported on the target device. MAX version is blank by default. -->
    <preference name="android-installLocation"    value="auto" />
    <!-- android: app install location. 'auto' will choose. 'internalOnly' is device memory.
          'preferExternal' is SDCard. -->
```

A continuación, definiremos los iconos para las plataformas de iOS y Android:

```
<icon src="Icon.png" />
   <icon src="res/Icon-36.png"      gap:platform="android"    gap:density="ldpi" />
   <icon src="res/Icon-48.png"      gap:platform="android"    gap:density="mdpi" />
   <icon src="res/Icon-72.png"      gap:platform="android"    gap:density="hdpi" />
   <icon src="res/Icon-96.png"      gap:platform="android"    gap:density="xhdpi" />
   <icon src="res/Icon.png"        gap:platform="ios"        width="57" height="57" />
   <icon src="res/Icon-72.png"     gap:platform="ios"        width="72" height="72" />
   <icon src="res/Icon@2x.png"     gap:platform="ios"        width="114" height="114" />
   <icon src="res/Icon-72@2x.png"  gap:platform="ios"        width="144" height="144" />
```

Y después las splash screens; son las imágenes que aparecen mientras se inicia la aplicación en el dispositivo:

```
    <gap:splash src="res/screen320x426.png"      gap:platform="android" gap:density="ldpi" />
    <gap:splash src="res/screen320x470.png"      gap:platform="android" gap:density="mdpi" />
    <gap:splash src="res/screen480x640.png"     gap:platform="android" gap:density="hdpi" />
    <gap:splash src="res/screen720x960.png"    gap:platform="android" gap:density="xhdpi" />
   <gap:splash src="res/screen320x480.png"      gap:platform="ios"     width="320" height="480" />
    <gap:splash src="res/screen640x960.png"    gap:platform="ios"     width="640" height="960" />
    <gap:splash src="res/screen768x1024.png"    gap:platform="ios"     width="768" height="1024" />
```

Y finalmente, indicaremos que nuestra aplicación se puede conectar a cualquier dirección externa; por temas de seguridad se recomienda poner solo aquellos dominios con los que trabajaremos, pero de cara al desarrollo, es mejor esta opción para ahorrarnos problemas.

```
<access origin="*"/>
```

Finalmente, ya podemos cerrar el elemento <widget> con la instrucción </widget> para tener el archivo acabado.

Y con este fichero acabado, ya habremos finalizado el desarrollo de la app. Ahora lo tendremos que testear y depurar, y generar el compilado final empaquetado para poderlo subir a las app stores.

Enlace recomendado

Podéis encontrar más información relativa a ello en el documento config.xml en la URL:

https://build.phonegap.com/docs/config-xml

2. Testeo, depuración y compilación de una aplicación sencilla realizada con jQuery Mobile y PhoneGap

En este apartado veremos algunas de las herramientas y métodos que podemos utilizar para testear nuestras aplicaciones, eliminar errores y generar nuestros binarios finales. Tener un buen método para depurar código es seguramente una de las partes más importantes en el desarrollo, además de ahorrarnos mucho tiempo.

Testeo y depuración

En el mundo de las aplicaciones móviles es muy importante probar nuestro código en el máximo de dispositivos posibles, puesto que puede variar la visualización, sobre todo dependiendo de la plataforma, versión de software o tamaño del dispositivo.

Es importante siempre hacer test en el dispositivo final si es posible, a poder ser también durante el desarrollo; de este modo, siempre evitaremos sorpresas de última hora. El problema principal es que esto es bastante lento, por lo que intentaremos utilizar siempre herramientas locales como los emuladores disponibles de cada plataforma que nos sean cómodos y, de vez en cuando, probar que sigue funcionando también de forma correcta en el dispositivo final.

Herramientas locales

Como herramienta principal utilizaremos, sobre todo, el Inspector Web o barra de desarrollador, disponible en Safari y Chrome. Ya lo hemos introducido anteriormente para hacer cambios CSS; es una herramienta muy completa que nos permitirá hacer prácticamente cualquier cosa que queramos.

Dispone de un inspector de elementos HTML, para visualizar o editar rápidamente HTML o CSS, sin tener que modificar cada vez el archivo del proyecto; esto nos da mucha velocidad para probar cosas o encontrar un error.

En el apartado Resources podemos visualizar información almacenada en el supuesto de que la aplicación utilice algún tipo de almacenamiento local, como Web SQL, Web Storage, etc.

En Networks visualizamos rápidamente la carga de los diferentes archivos; si da error alguno, respuestas HTTP, medida de los archivos, etc.

En Console, encontramos la consola javascript en la que visualizaremos los diferentes errores. Prácticamente indispensable es la instrucción console.log(); con esta instrucción podemos visualizar las diferentes variables Javascript de nuestro código, directamente en esta consola.

Enlace recomendado

Podéis encontrar la información completa aquí:
https://developers.google.com/chrome-developer-tools/

Como herramienta local también destacaría el complemento de Chrome Ripple Emulator, que nos facilita mucho el trabajo a la hora de testear el comportamiento de nuestra app con PhoneGap de una manera muy sencilla y rápida.

Nos permite probar nuestra aplicación en muchos dispositivos diferentes (no son dispositivos finales, solo emula tamaño y características técnicas). Y poder probar cosas como el acelerómetro, geolocalización, orientación desde el mismo navegador, es un gran avance.

Al ser un proyecto de BlackBerry no está claro su futuro desarrollo, pero esperamos que siga mejorando.

Cabe destacar también las herramientas CLI cordova y PhoneGap, que cuando se aprenden a utilizar son herramientas muy potentes para poder trabajar con PhoneGap de una manera muy sencilla.

Herramientas remotas

También necesitaremos antes o después herramientas para depurar nuestro código, cuando estemos probando nuestra aplicación, en el emulador oficial o directamente en nuestro dispositivo.

Existen diferentes herramientas y podemos escoger la que más nos convenga.

Una de las más sencillas seguramente es el servicio de jsconsole.com; los pasos para utilizarlo son abrir la web del servicio, escribir :listen y copiar el código que devuelva a nuestro HTML; de este modo enlazaremos nuestra aplicación con jsconsole.com, y a partir de ahora todas las informaciones de consola se mostrarán en su web directamente. Es muy práctico porque siempre funcionará ya sea en local, en el simulador o en el dispositivo; siempre veremos el resultado en su web.

Si utilizamos la opción CLI, los logs de la consola se guardarán de forma automática en un directorio en concreto por si queremos acceder a él. Esto nos ahorra tener que hacer configuraciones complicadas.

Las otras opciones suelen requerir instalaciones o configuraciones más complicadas en las que no entraremos.

A continuación, podéis ver un listado de todas las herramientas remotas a destacar:

- JSConsole: http://jsconsole.com/
- Weinre: http://people.apache.org/~pmuellr/weinre/docs/latest/Hombre.html
- Socketbug: http://socketbug.com/
- iWebInspector: http://www.iwebinspector.com/
- Mobile Safari Remote Debugger

Compilación

Finalmente, una vez tengamos testeada nuestra aplicación, procederemos a generar los binarios.

Esto supone generar un compilado diferente para cada una de las plataformas que deseamos; en el caso de iOS, será necesario disponer de una cuenta de desarrollador, ya que para generar el compilado es necesario una clave privada de desarrollador para poder firmarlo; para el resto de plataformas esto no es necesario; para Android y BlackBerry, firmar la aplicación es opcional.

Dependiendo de la opción que hayamos elegido para desarrollar con PhoneGap, podremos generar el compilado de diferentes maneras. En cualquier caso, siempre podemos utilizar los servicios del PhoneGap Build Service para generar los compilados. Esto supone subir la carpeta www con el proyecto y, si lo deseamos, incluyendo el fichero config.xml; si no, nos lo generará de forma automática.

Si tenemos las herramientas CLI instaladas, podemos utilizar las siguientes instrucciones para generar un compilado de una plataforma en nuestra cuenta.

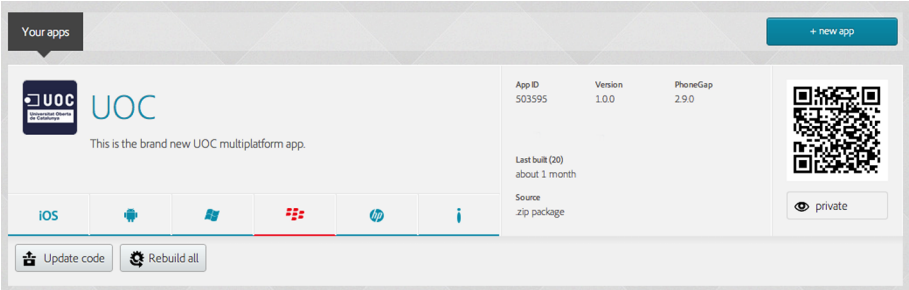
phonegap remote login -u <username> -p <passwpord>

phonegap remote build <platform>

En los dos casos necesitaremos siempre disponer de una cuenta ya creada en el servicio, para poder generar nuestros compilados.

La gran ventaja del PhoneGap Build Service es la posibilidad de compilar de forma fácil para cualquiera de las plataformas; esto implica poder compilar para iOS, Android, Windows Phone, BlackBerry, WebOS y Symbian, sin tener que instalar ninguna herramienta adicional en nuestro ordenador. Cada vez dispone de más características y servicios que lo hacen más práctico, sobre todo a la hora de generar los binarios finales.

Ejemplo de aplicación en el PhoneGap Build Service



Si por otro lado queremos generar los binarios de forma local, necesitaremos tener instaladas las herramientas de desarrollo de la plataforma que deseamos compilar. Podemos generar el binario directamente utilizando la herramienta de la plataforma o, si hemos instalado el CLI, lo podemos generar desde la línea de pedidos.

cordova build <platform>

Con esta instrucción phonegap nos preparará los archivos necesarios y nos compilará la aplicación para la plataforma deseada.

Los binarios para las diferentes plataformas, como ya hemos comentado, son archivos diferentes:

- **iOS**: archivos .ipa
- **Android**: archivos .apk
- **Windows Phone**: archivos .xap
- **BlackBerry**: archivos .ota
- **WebOs**: archivos .ipk
- **Symbian**: archivos .wgz

Una vez tengamos este archivo generado ya lo podremos distribuir o subir a la tienda de aplicaciones correspondiente, siempre dependiendo de las restricciones que nos ponga cada plataforma a la hora de distribuir nuestras aplicaciones.