

# Programación Web para Móviles

## Índice

1	Introducción al desarrollo web para móviles.....	4
1.1	Introducción.....	4
1.2	Instalación de un servidor Web.....	13
1.3	HTML.....	23
1.4	CSS.....	34
2	Ejercicios - Introducción al desarrollo web para móviles.....	44
2.1	Ejercicio 1 - Tablas.....	44
2.2	Ejercicio 2 - Cajas Div.....	44
2.3	Ejercicio 3 - Terminando la Web de ejemplo.....	45
3	HTML5 y CSS3.....	47
3.1	HTML 5.....	47
3.2	CSS3.....	63
4	Ejercicios - HTML5 y CSS3.....	75
4.1	Ejercicio 1 - Estructura.....	75
4.2	Ejercicio 2 - Canvas.....	75
4.3	Ejercicio 3 - Multimedia.....	75
4.4	Ejercicio 4 - Geolocalización.....	76
4.5	Ejercicio 5 - Almacenamiento Offline.....	76
4.6	Ejercicio 6 - CSS3.....	76
5	Introducción a jQuery Mobile.....	78
5.1	¿Qué es jQuery Mobile?.....	78
5.2	Páginas en jQuery Mobile.....	80
5.3	Barras de herramientas.....	97
5.4	Formateo de contenido.....	103
6	Introducción a jQuery Mobile - Ejercicios.....	110
6.1	Ejercicio 1 - Inicio de la aplicación.....	110

6.2 Ejercicio 2 - Página del autor .....	110
6.3 Ejercicio 3 - Listado de libros.....	111
7 Aspectos avanzados de jQuery Mobile.....	112
7.1 Listados de elementos.....	112
7.2 Formularios.....	122
8 Aspectos avanzados de jQuery Mobile - Ejercicios.....	141
8.1 Ejercicio 1: Listado de libros mejorado.....	141
8.2 Ejercicio 2: Registro de usuarios.....	141
8.3 Ejercicio 3: Consumiendo servicios REST.....	142
9 Introducción a Sencha Touch.....	143
9.1 Introducción.....	143
9.2 Sencha Touch vs. JQuery Mobile.....	144
9.3 Instalar Sencha Touch.....	145
9.4 Estructura de carpetas.....	146
9.5 Código HTML básico de una aplicación.....	148
9.6 Instanciar una aplicación.....	150
9.7 Comprobando los resultados.....	151
9.8 Todo en un único fichero.....	153
9.9 Paneles y Layouts.....	153
9.10 Identificadores.....	155
9.11 Toolbars.....	156
9.12 Docked items.....	157
9.13 Botones.....	158
9.14 Transiciones de cambio de vista.....	162
9.15 TabBar.....	164
9.16 Carousel.....	165
9.17 MessageBox.....	166
10 Ejercicios - Introducción a Sencha Touch.....	169
10.1 Ejercicio 1 - Estructura de la aplicación y creación de paneles.....	169
10.2 Ejercicio 2 - Barras de herramientas y contenido de la ayuda.....	170
10.3 Ejercicio 3 - Botones y Transiciones.....	171
11 Aspectos avanzados de Sencha Touch.....	172

11.1 Data Model.....	172
11.2 Data Store.....	173
11.3 Plantillas.....	176
11.4 Data Views.....	178
11.5 Listados.....	178
11.6 Formularios.....	182
11.7 Más información.....	191
12 Ejercicios - Aspectos avanzados de Sencha Touch.....	193
12.1 Ejercicio 1 - Modelo de datos y Listado.....	193
12.2 Ejercicio 2 - Formulario, Crear y Editar notas.....	194
12.3 Ejercicio 3 - Guardar y Borrar notas.....	194
13 Bibliografía.....	196
13.1 Libros.....	196
13.2 Enlaces.....	196
13.3 Twitter.....	196
13.4 Podcasts.....	197

## 1. Introducción al desarrollo web para móviles

### 1.1. Introducción

La Web móvil es la misma Web que la de escritorio, utiliza la misma arquitectura básica y muchas de las mismas tecnologías. Pero existen claras diferencias que impiden que su funcionamiento y manejo sea el mismo, como son: el tamaño de la pantalla, las diferentes formas de manejo (táctil, teclado del móvil, etc.) y el ancho de banda.

Otra diferencia que se debería de tener en cuenta es que es "móvil". Accedemos a Internet desde lugares en los que sería imposible hacerlo con un ordenador de sobremesa o incluso un portátil. Esto influye también en el uso que le damos, se hacen más búsquedas o consultas, además de que la información se debe de disponer de forma más clara y directa.

Debido a todo esto, al programar una Web para móvil debemos de tener en cuenta que ni el contenido, ni la apariencia, ni la estructura de la web va a ser la misma que la que podríamos hacer para un ordenador de escritorio. Hemos de diseñar muy bien este tipo de aplicaciones y orientarlas al uso principal que le va a dar el usuario. Para esto se suele referir a la regla del 20%: el 80% del contenido del sitio web de escritorio no es válido para una web móvil. Por lo que hemos de centrarnos en ese 20% restante, averiguar cuáles, y optimizar nuestro sitio para este uso.

#### 1.1.1. Aplicación móvil vs. Aplicación nativa

Los desarrolladores de aplicaciones nativas tienen la ventaja de poder usar funciones no disponibles para la web móvil:

- El uso de interfaces nativas que proveen los propios SDK como iPhone o Android.
- El uso de bases de datos locales. Aunque en HTML 5 se pueden usar un almacenamiento local, hay que reconocer que estas tecnologías están más avanzadas en los sistemas nativos.
- Notificaciones push. A esto se refiere con los avisos centralizados que muestran las aplicaciones, aún cuando están ocultas. Una fuerte razón que no puede ser implementada en una web móvil.
- Geolocalización. Hemos visto algunos ejemplos de geolocalización a través de HTML5, pero que no acaban de alcanzar la misma experiencia de usuario que una aplicación móvil. Podemos interactuar con el mapa o con las funcionalidades asociadas a la localización del usuario, pero no ir mucho más allá.
- Soporte para cámara o vídeo. Las funciones multimedia están perfectamente acopladas a las aplicaciones nativas, pudiendo añadir funcionalidades específicas a nuestra aplicación.

Sin embargo estas diferencias cada vez se van haciendo menores:

- Gracias al uso de HTML5, CSS3 y JavaScript cada vez se pueden hacer más cosas y obtener mejores resultados.
- Cada vez hay mejores frameworks de desarrollo para aplicaciones móviles, como JQuery Mobile o Sencha Touch. Estos nos permiten crear webs con apariencia cercana a las de las aplicaciones nativas, pero a su vez con toda la potencia de la Web. Estos framework nos permiten adaptar el contenido según el dispositivo usado y sus posibilidades técnicas: pantalla táctil, reproducción de vídeos o resolución de pantalla. Por lo que podríamos decir que una Web móvil es mucho más adaptable (además de multiplataforma) que una aplicación móvil.
- La web sigue siendo el negocio principal de muchas empresas de Internet. El desarrollo web no ha muerto por la inclusión de las aplicaciones móviles, sino que se ve afectado por un proceso de cambio hacia la adopción de tecnologías nuevas como HTML5.
- La inclusión de la tecnología *PhoneGap* cada vez está recortando más estas diferencias. *PhoneGap* es la posibilidad de crear una aplicación nativa instalable a partir de una página Web móvil (que se pueda distribuir también en *Android Market* o en la *App Store*). Además estas tecnologías facilitan el uso de funcionalidades del dispositivo móvil directamente a través de código JavaScript, como puede ser el acceso a la cámara, acelerómetro, geolocalización, listado de contactos, comprobar el estado de la conexión, etc.

### 1.1.2. Reglas de usabilidad

---

#### 1. Reducir la cantidad de contenido

Las aplicaciones móviles deben de ser optimizadas dado que el espacio visual es mucho más limitado que en una pantalla de ordenador. Cada píxel cuenta, y no todo lo que es válido para una Web de escritorio lo es para una Web móvil.

Solo debemos de incluir el contenido y las características principales y más importantes. Los contenidos con menor importancia deben de ser eliminados, como contenidos secundarios, normalmente localizados en columnas laterales.

La web móvil debe de estar enfocada a este contenido principal. Facilitar su lectura y navegación, así como mejorar los tiempos de carga reduciendo imágenes y contenidos.

#### 2. Usar una sola columna

Las páginas Web anchas y con varias columnas dificultan la navegación y lectura en un dispositivo móvil. Incluso en los terminales móviles con pantallas más grandes hay que realizar zoom para moverse y ver bien el contenido. Esta es una práctica que debemos evitar, pues tener que ir realizando zoom añade más pasos a la navegación, y en algunos dispositivos no es tan fácil de realizar como en un iPhone.

Lo mejor es tener nuestro contenido en una sola columna que use todo el ancho de la pantalla. Para añadir contenido lo deberemos de hacer hacia abajo (o creando una página

nueva), nunca hacia los lados (o creando columnas). Esto nos asegura que el contenido se va a visualizar correctamente, además es mucho más intuitivo realizar *scroll* hacia abajo para ir leyendo.

### 3. Muestra los enlaces de navegación de forma diferente

No pongas todos los enlaces de navegación en la parte superior de la pantalla. Si hay muchos desplazarán todo el contenido hacia abajo, y es posible que los tengas que poner muy reducidos.

La página principal debería de contener los enlaces al resto del contenido junto con un buscador (si fuese necesario). El contenido debería de estar en páginas secundarias bien organizado. Por ejemplo, cuando un usuario entra en un sitio de *eCommerce* suelen tener una categoría de producto en mente que quiere consultar, la cual la podrían encontrar usando el buscador o directamente a partir del menú. Es decir, la página principal debe facilitar el acceso rápido a la información más importante de la web.

También hay otras opciones para colocar el menú de navegación, como una lista desplegable o al final de la página. Son muy cómodas las barras de herramientas estáticas que ofrecen las opciones principales (volver a la página inicial, botones principales, etc.).

### 4. Minimiza la cantidad de datos solicitados

Escribir texto utilizando un terminal móvil es mucho más difícil que hacerlo utilizando el teclado de un ordenador de sobremesa. Además los usuarios suelen escribir mucho más lento y cometer más errores. Por estas razones tenemos que intentar minimizar la cantidad de texto solicitado.

Una forma de conseguir esto es permitir almacenar los datos (usuario, contraseñas, configuración, direcciones, etc.), o aprovechar algunas de las funcionalidades que incorporan los dispositivos móviles (como veremos más adelante).

### 5. Decide si necesitas más de una Web para móvil

El tamaño de pantalla, la capacidad de procesamiento y de usabilidad varía enormemente de un terminal a otro. Por esta razón a veces debemos de considerar crear varios sitios web con el mismo contenido pero adaptado a diferentes necesidades. Por ejemplo, Facebook tiene [m.facebook.com](http://m.facebook.com) como sitio web principal para móviles, pero además tiene una versión optimizada para pantallas táctiles ([touch.facebook.com](http://touch.facebook.com)) y una versión optimizada para conexiones más lentas ([0.facebook.com](http://0.facebook.com)).

### 6. Diseña para pantallas táctiles, pero también para teléfonos no-táctiles

La forma de navegar por las páginas web es muy diferente según el dispositivo: pantallas táctiles, trackball, joystick, teclado, etc. Estas características también son importantes a la hora de realizar el diseño. Por ejemplo, la principal dificultad está en la selección y pulsado sobre textos o enlaces pequeños. En las pantallas táctiles además se dificulta pulsar sobre elementos que estén muy juntos.

Por esta razón, los enlaces o elementos que puedan ser seleccionados deben de ocupar un mayor espacio en pantalla, incluirlos en botones o cuadros más grandes, que puedan ser pulsados con facilidad.

## **7. Aprovecha las funcionalidades que incorporan los móviles**

Los teléfonos móviles tienen algunas ventajas sobre los PCs, las cuales pueden facilitar la realización de algunas tareas. Algunas de estas funcionalidades añadidas son:

Realizar llamadas: puede parecer evidente pero es una funcionalidad muy útil que los PCs no pueden realizar tan fácilmente. Por esta razón debemos de aprovecharla para, por ejemplo, llamar directamente al presionar sobre un número de teléfono, facilitar el contacto con un servicio técnico, etc.

Uso de mapas y posición actual: es posible dar la opción al usuario de seleccionar una dirección y que automáticamente se abra en la aplicación de mapas del dispositivo móvil. También es muy interesante el uso de la posición actual para mostrar puntos de interés cercanos, calcular rutas, etc.

Solicitud de información de forma innovadora: como por ejemplo los códigos QR, que se han usado en algunas campañas de publicidad, etc.

### **1.1.3. Buenas prácticas de programación Web para dispositivos móviles**

---

Además al programar un sitio Web para dispositivos móviles también se deben de seguir algunas pequeñas pautas. Al cumplirlas, se incrementará el público que puede acceder a los contenidos, creando sitios Web eficaces y haciendo la navegación accesible desde más dispositivos. Las principales pautas que debemos seguir son:

- URL lo más corta posible.
- La barra de navegación debe de estar en la cabecera y ofrecer solo la navegación mínima necesaria.
- Lo más importante, primero. Ofrece al usuario lo que busca. Si sabes que pide muchas noticias de prensa, eso debe ser el primer enlace de la web.
- No recargar automáticamente la página, a menos que se informe claramente al usuario y se ofrezca una forma de poder detener dicha acción. El usuario se podría encontrar con una factura considerable por un descuido.
- Peso limitado. Nuestra Web para móvil no debe de ocupar mucho, para esto se recomienda reducir todo lo posible las imágenes y el contenido multimedia.
- Scroll. Limite el desplazamiento de la página a una dirección.
- Test. Haga pruebas en emuladores y en dispositivos reales.
- Enlaces:
  - Intente que sean mínimos a recursos externos.
  - Intente conseguir un equilibrio entre los enlaces que hay en la web y lo que un usuario tarda en llegar a donde quiere.
  - Identificar de forma clara el destino del enlace. Si no estamos seguros de si el

formato del destino es soportado por el dispositivo debemos indicarlo. Por ejemplo, para un pdf o para una web no móvil.

- No usar ventanas emergentes.
- No ofrecer más contenido del solicitado por el usuario y que sea estrictamente necesario para la estructura y navegación de la página.

#### 1.1.4. Dominio

---

Existen diferentes alternativas sobre el dominio que podemos usar. En definitiva esta es una decisión personal, pues todas ellas tienen sus ventajas e inconvenientes. La única recomendación que se suele hacer es tener varias opciones disponibles, con la intención de facilitar al máximo el acceso.

Podemos tener un subdominio de nuestro sitio Web especializado para dispositivos móviles. Por ejemplo, si nuestro sitio Web es *www.midominio.com*, el sitio para dispositivos móviles podría ser *m.midominio.com*. Por ejemplo, Facebook tiene disponibles los sitios [m.facebook.com](http://m.facebook.com) (como sitio web para dispositivos móviles) y [touch.facebook.com](http://touch.facebook.com) (para dispositivos táctiles).

También podemos usar el dominio principal y diferenciar (desde el cliente o desde el servidor) si se trata de un dispositivo móvil. En este caso el usuario accedería a la misma dirección pero sería redirigido al sitio correspondiente.

Otra opción es comprar un dominio *".mobi"* (especial para web móvil) con el mismo nombre que la web principal.

Si optamos por dar diferentes opciones de acceso deberemos crear redirecciones 301 al dominio principal seleccionado para manejar ese contenido, de la forma:

```
<?php
header("HTTP/1.1 301 Moved Permanently");
header("location:http://www.nueva_url.com");
?>
```

#### 1.1.5. Detección del navegador

---

Un dilema a la hora de desarrollar contenidos para móviles es cómo diferenciar entre dispositivos móviles y navegadores de escritorio. Esto se puede hacer fácilmente mediante una función de comprobación que nos indique el tipo de navegador que ha solicitado la web. Una vez obtenido el navegador tenemos varias opciones, como se comentaba en la sección anterior: redirigir al dominio correspondiente, o adaptar el código de nuestra página según el cliente.

A continuación se incluye una función en PHP que nos devuelve un número positivo si detecta que el navegador es un dispositivo móvil, y 0 en caso de ser un navegador de escritorio.



```

public static function mobileBrowser()
{
    $mobile_browser = '0';

    //$_SERVER['HTTP_USER_AGENT'] -> el agente de usuario que está
    // accediendo a la página.
    if(preg_match('/(up.browser|up.link|mmp|symbian|smartphone|midp|wap|phone)/i',
        strtolower($_SERVER['HTTP_USER_AGENT'])))
    {
        $mobile_browser++;
    }

    //$_SERVER['HTTP_ACCEPT'] -> Indica los tipos MIME que el cliente
    puede recibir.
    if((strpos(strtolower($_SERVER['HTTP_ACCEPT']), 'application/vnd.wap.xhtml+xml')>0)
        or
        ((isset($_SERVER['HTTP_X_WAP_PROFILE']) or
        isset($_SERVER['HTTP_PROFILE'])))
    {
        $mobile_browser++;
    }

    $mobile_ua = strtolower(substr($_SERVER['HTTP_USER_AGENT'], 0, 4));
    $mobile_agents = array(
        'w3c'
        , 'acs-' , 'alav' , 'alca' , 'amoi' , 'audi' , 'avan' , 'benq' , 'bird' , 'blac' ,
        'blaz' , 'brew' , 'cell' , 'cldc' , 'cmd-' , 'dang' , 'doco' , 'eric' , 'hipt' , 'inno' ,
        'ipaq' , 'java' , 'jigs' , 'kddi' , 'keji' , 'leno' , 'lg-c' , 'lg-d' , 'lg-g' , 'lge-' ,
        'maui' , 'maxo' , 'midp' , 'mits' , 'mmef' , 'mobi' , 'mot-' , 'moto' , 'mwbp' , 'nec-' ,
        'newt' , 'noki' , 'oper' , 'palm' , 'pana' , 'pant' , 'phil' , 'play' , 'port' , 'prox' ,
        'qwap' , 'sage' , 'sams' , 'sany' , 'sch-' , 'sec-' , 'send' , 'seri' , 'sgh-' , 'shar' ,
        'sie-' , 'siem' , 'smal' , 'smar' , 'sony' , 'sph-' , 'symb' , 't-mo' , 'teli' , 'tim-' ,
        'tosh' , 'tsm-' , 'upgl' , 'upsi' , 'vk-v' , 'voda' , 'wap-' , 'wapa' , 'wapi' , 'wapp' ,
        'wapr' , 'webc' , 'winw' , 'winw' , 'xda' , 'xda-');

    //buscar agentes en el array de agentes
    if(in_array($mobile_ua, $mobile_agents)) {
        $mobile_browser++;
    }

    //$_SERVER['ALL_HTTP'] -> Todas las cabeceras HTTP
    if(strpos(strtolower($_SERVER['ALL_HTTP']), 'OperaMini')>0) {
        $mobile_browser++;
    }
    if(strpos(strtolower($_SERVER['HTTP_USER_AGENT']), 'windows')>0) {
        $mobile_browser=0;
    }

    return $mobile_browser;
}

```

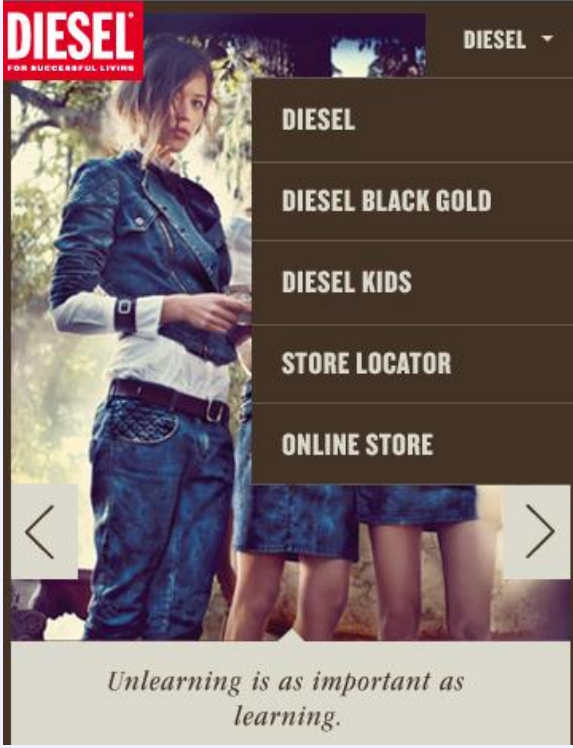

También existen librerías un poco más completas para detectar el cliente usando código PHP. En este ejemplo se cubren los casos más básicos, y funcionará en la mayoría de ellos. Pero si queremos una librería más completa podemos consultar: <http://detectmobilebrowsers.mobi/>

La detección del navegador también la podemos hacer desde el cliente y usando diferentes lenguajes, como por ejemplo JavaScript. En la dirección <http://detectmobilebrowsers.com/> tenemos disponibles librerías que realizan esta función en multitud de lenguajes, como JavaScript, ASP, ASP.NET, C#, Apache, JSP, JQuery,

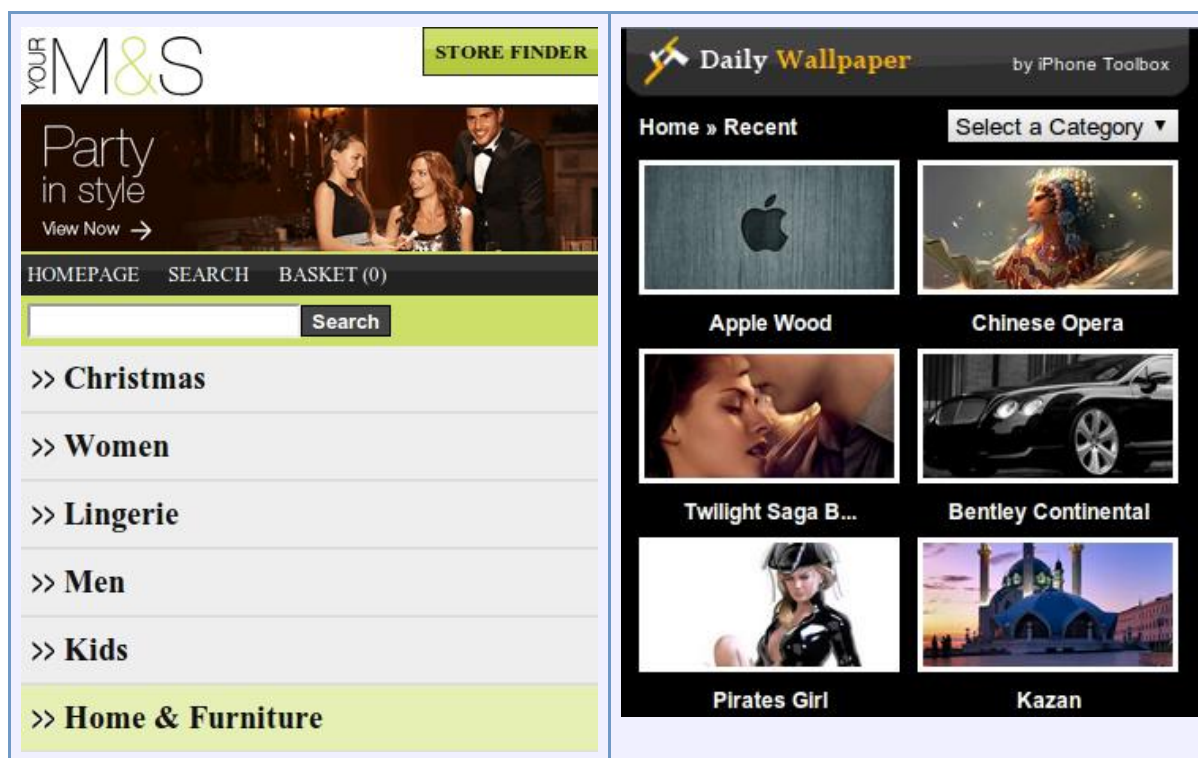
Perl, etc.

### 1.1.6. Inspiración

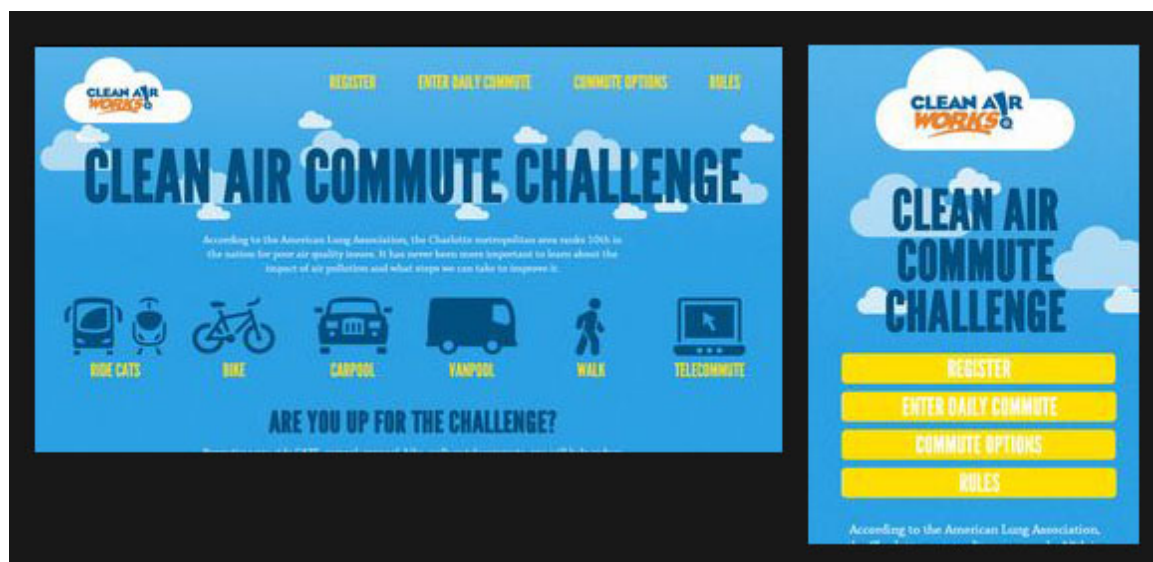
Cuando se empieza en el desarrollo de webs para dispositivos móviles es bueno buscar inspiración, ir a ver que ha hecho gente con más experiencia en el campo. A continuación se incluye una pequeña lista de ejemplos:

<p><a href="http://m.diesel.com">http://m.diesel.com</a></p> 	<p><a href="http://m.flickr.com/">http://m.flickr.com/</a></p> 
<p><a href="http://www.vspink.mobi/">http://www.vspink.mobi/</a></p>	<p><a href="http://mobile.walmart.com/">http://mobile.walmart.com/</a></p>

 <p><b>PINK</b> VICTORIA'S SECRET</p> <p><b>PINK NATION</b> Join Now!</p> <p><b>NEW LOOKS</b> Fall Favorites</p> <p><b>PINK NFL</b> Team Gear &amp; Freebies</p> <p><b>GOODIES</b> Mobile Wallpapers</p> <p><b>FASHION SHOW</b> Pics, Flicks &amp; More</p> <p><b>SHOP</b> PINK on the Go</p> <p><a href="http://m.marksandspencer.com/">http://m.marksandspencer.com/</a></p>	 <p><b>Walmart</b></p> <p>What can we help you find?</p> <p><b>Rollbacks</b> Low prices just got lower.</p> <p><b>Value of the Day</b> Grab it before it's gone</p> <p><b>Rollbacks</b> Low prices just got lower</p> <p><b>Local Ad</b> Find great values at a store nearby</p> <p><b>Pharmacy</b> Browse \$4 prescriptions, order refills</p> <p><b>Photo</b> View, share or print your online photos</p> <p><b>Shopping List</b> Build and manage your shopping lists</p> <p><a href="http://iphonetoolbox.com/dailywallpaper">http://iphonetoolbox.com/dailywallpaper</a></p>
---	--



En todos de ellos debemos de considerar la adaptación que se ha hecho del contenido entre la web de escritorio y la web para dispositivos móviles. En la siguiente imagen se puede ver una comparación en la que varía la disposición. Pero como hemos dicho, los cambios no son únicamente de disposición, tenemos que recordar la regla del 20%.





## 1.2. Instalación de un servidor Web

En la programación Web, una de las herramientas principales que necesitamos es un servidor Web o servidor HTTP. Este es el encargado de compilar el código (según el lenguaje de programación que utilicemos) y enviarlo al cliente utilizando el protocolo de transferencia HTTP.

Dado que instalar un servidor Web completo y configurarlo correctamente es una tarea bastante costosa, para el desarrollo y testeo de aplicaciones en local se suele utilizar un servidor XAMPP. Este es un paquete software de fácil instalación que incluye todo lo necesario para la ejecución de un servidor Web.

Es independiente de plataforma, software libre (licencia GNU), y consiste principalmente en la base de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script: PHP y Perl. El nombre proviene del acrónimo formado por **X** (para cualquiera de los diferentes sistemas operativos), **A**pache, **M**ySQL, **P**HP y **P**erl. Actualmente XAMPP está disponible para Microsoft Windows (WAMPP), GNU/Linux (LAMPP), Solaris y MacOS X (MAMPP).

### 1.2.1. XAMPP para Linux

Para su instalación en Linux tendremos que seguir los siguientes pasos:

1. En primer lugar descargamos el software desde:  
<http://www.apachefriends.org/en/xampp-linux.html>
2. Abrimos una consola y ejecutamos:

```
sudo tar xvfz xampp-linux-1.7.7.tar.gz -C /opt
```

Ahora ya tenemos instalado el servidor en la ruta '/opt/lampp'.

3. Para inicializar el servidor escribimos:

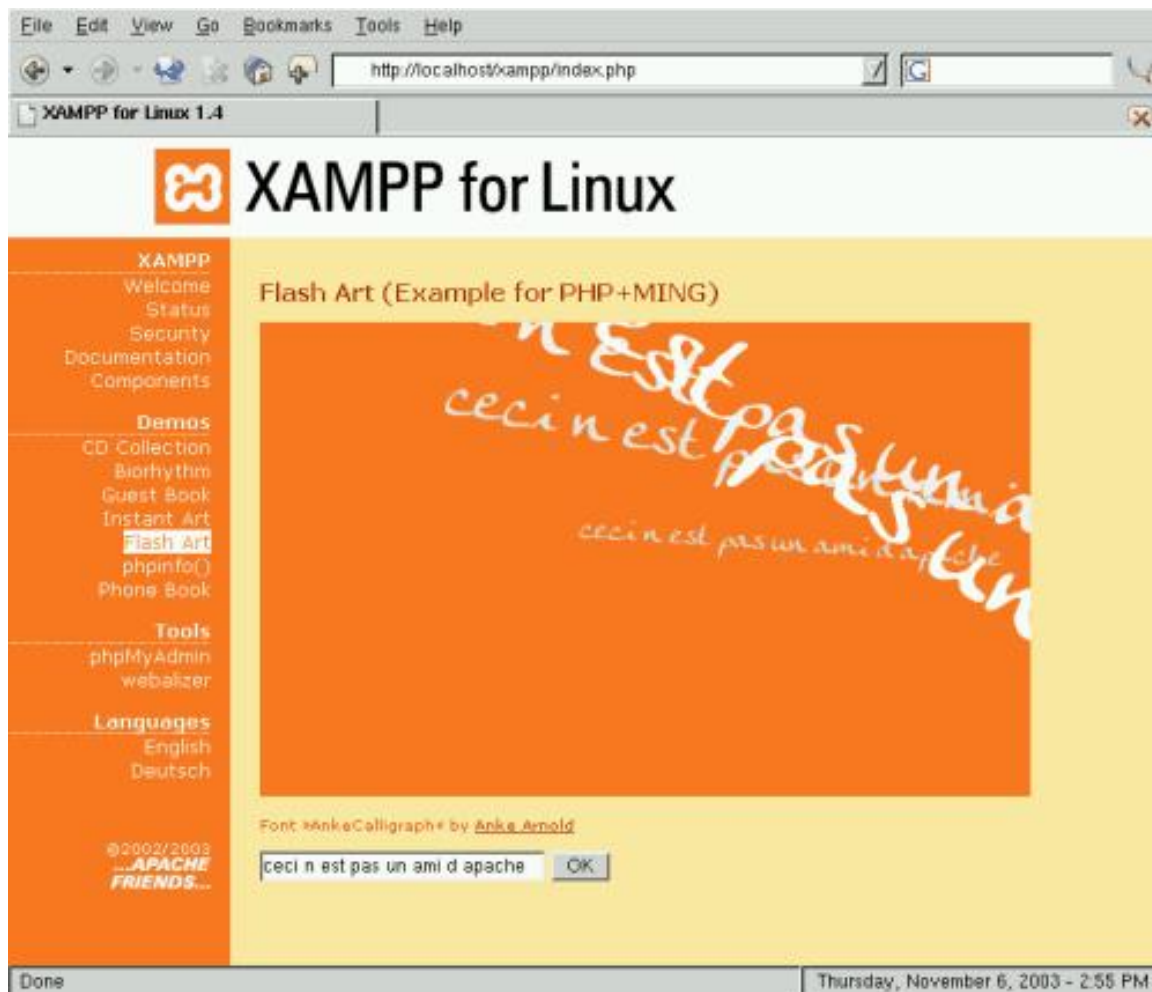
```
sudo /opt/lampp/lampp start
```

Veremos algo como:

```
Starting XAMPP 1.7.7...
LAMPP: Starting Apache...
LAMPP: Starting MySQL...
LAMPP started.
```

Ahora ya tenemos listo nuestro servidor Apache con MySQL.

4. Para comprobar que todo ha ido correctamente abrimos un navegador y escribimos la dirección "<http://localhost>", debería de abrirse una página similar a la siguiente:



El directorio raíz de Apache es “/opt/lampp/htdocs”, que será donde colocaremos nuestras páginas Web.

El servidor viene por defecto sin ninguna opción de seguridad activada (ya que se va a usar en local para pruebas), pero si quisiéramos activarlas tendríamos que escribir:

```
sudo /opt/lampp/lampp security
```

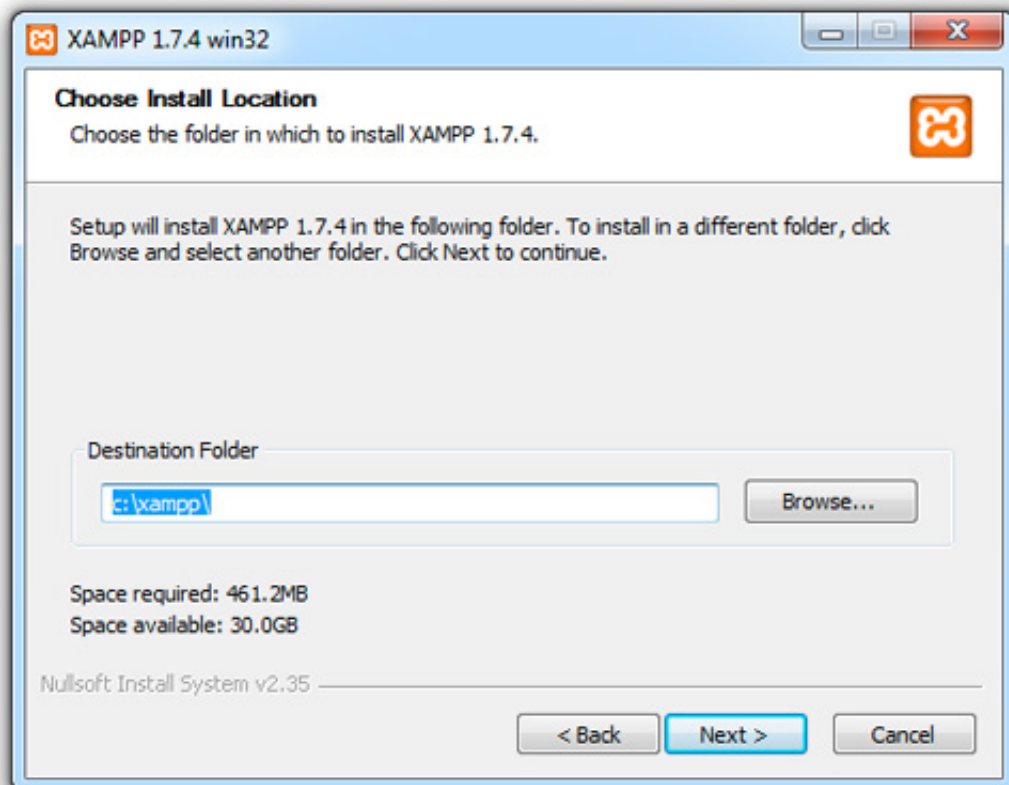
Por último, para detener el servidor Web simplemente ejecutamos en una consola:

```
sudo /opt/lampp/lampp stop
```

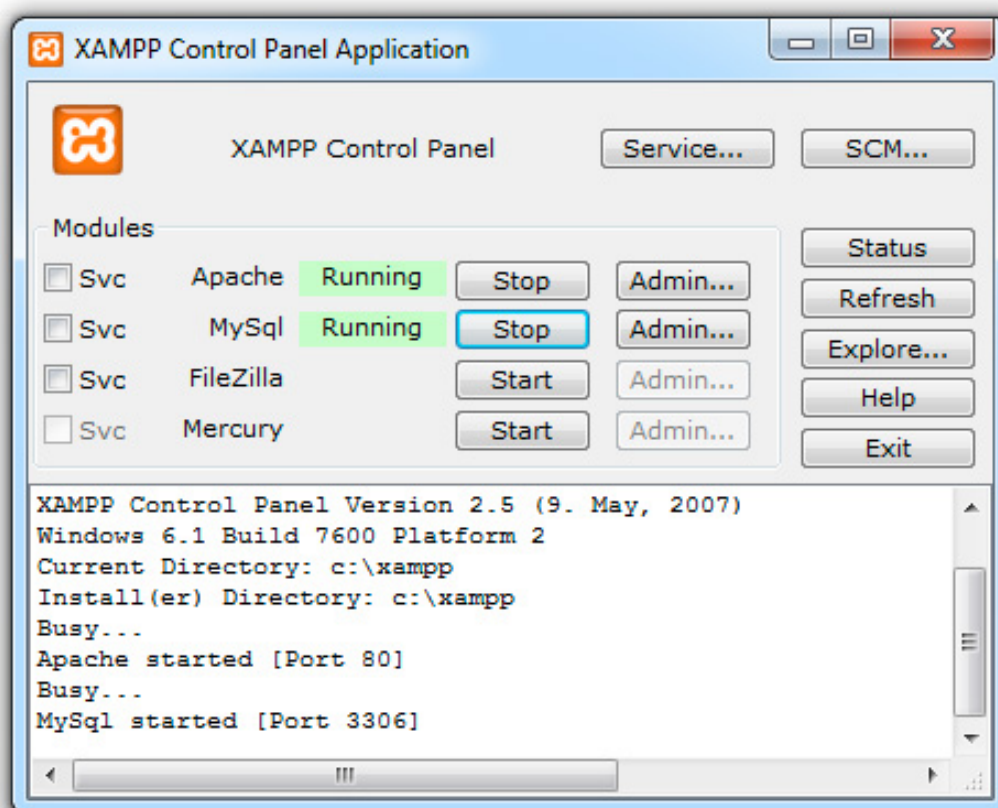
### 1.2.2. XAMPP para Windows

Los pasos para instalar XAMPP en Windows son los siguientes:

1. Descargamos la última versión del software desde:  
<http://www.apachefriends.org/en/xampp-windows.html>  
Existen diferentes opciones de descarga, la más sencilla y la que seguiremos aquí es utilizar el “instalador”.
2. Ejecutamos el instalador utilizando los valores por defecto:

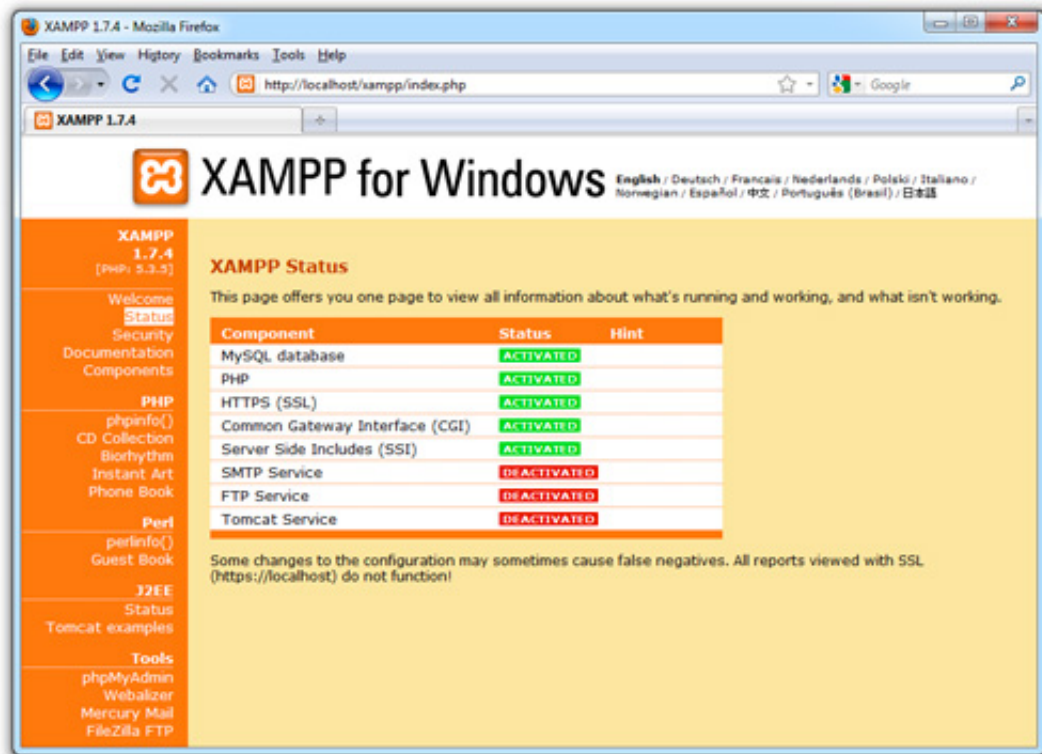


3. Después de la instalación abrimos el Panel de Control de XAMPP (se encuentra en el menú de Inicio > Programas > XAMPP). Desde aquí podemos iniciar y detener cada uno de los servicios individualmente.



4. Para comprobar que el servidor está instalado correctamente abrimos un navegador y escribimos la dirección “<http://localhost/>” (o también “<http://127.0.0.1/>”), donde se nos abrirá una página web como la siguiente:





Si nos avisa el Firewall del sistema tendremos que desbloquear o permitir el acceso a Apache.

En Windows, el directorio raíz de Apache para el contenido Web está en “C:\xampp\htdocs”, que será donde colocaremos nuestras páginas Web.

El servidor viene por defecto sin ninguna opción de seguridad activada (ya que se va a usar en local para pruebas), pero estas opciones se pueden configurar directamente desde un navegador accediendo a la dirección “<http://localhost/security/>”.

### 1.2.3. Instalar un servidor Web para Mac

Existen ciertas ventajas del uso de un Mac para el desarrollo de aplicaciones para móviles. Para empezar, el sistema operativo viene con un servidor web Apache instalado. El navegador por defecto, Safari, renderiza correctamente las aplicaciones basadas en WebKit. Y, por su puesto, tiene un excelente simulador para iOS como parte de Xcode.

Si queremos instalar un servidor XAMPP para Mac podemos acceder a la dirección “<http://www.apachefriends.org/en/xampp-macosx.html>” y seguir los pasos de instalación, muy similares a los ya vistos para Linux y Windows.

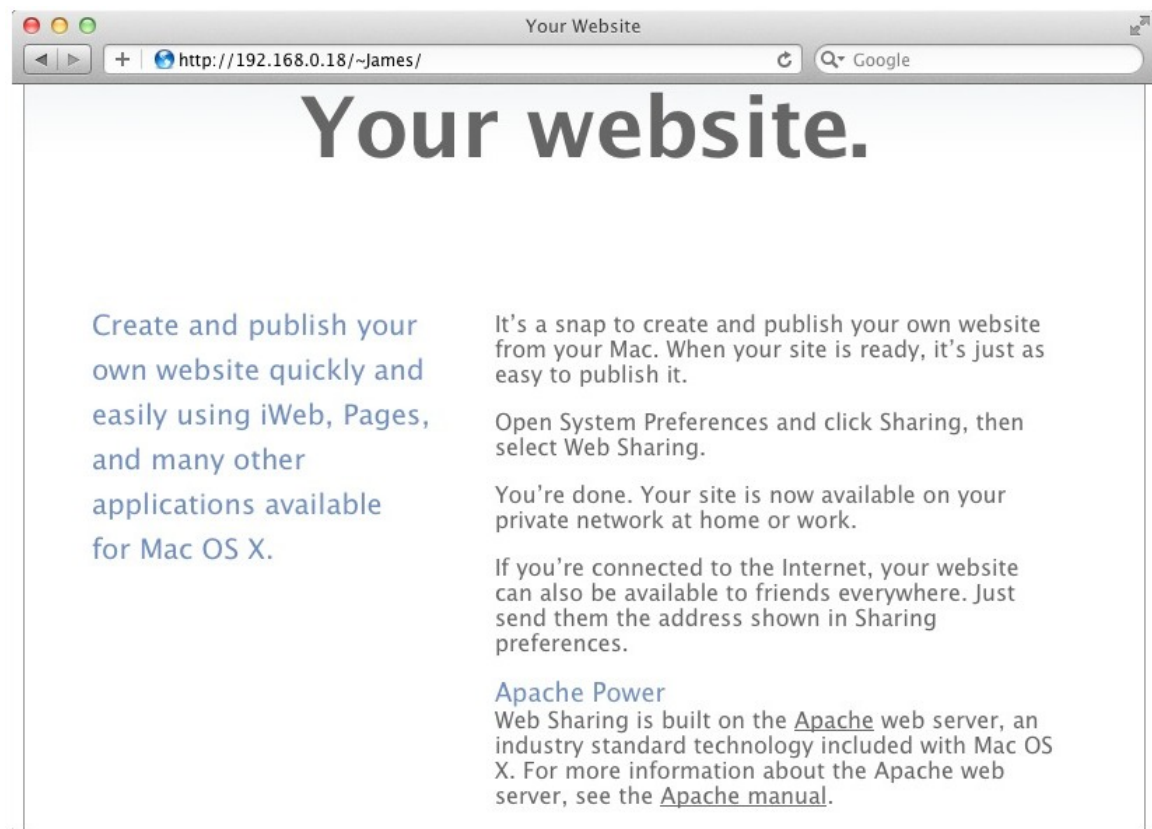
Para este ejemplo vamos a explicar la configuración del servidor Apache que viene con el sistema operativo. En primer lugar abrimos las “Preferencias del Sistema” y vamos al

panel “Sharing”, en el cual deberemos de habilitar la opción “Web Sharing”:



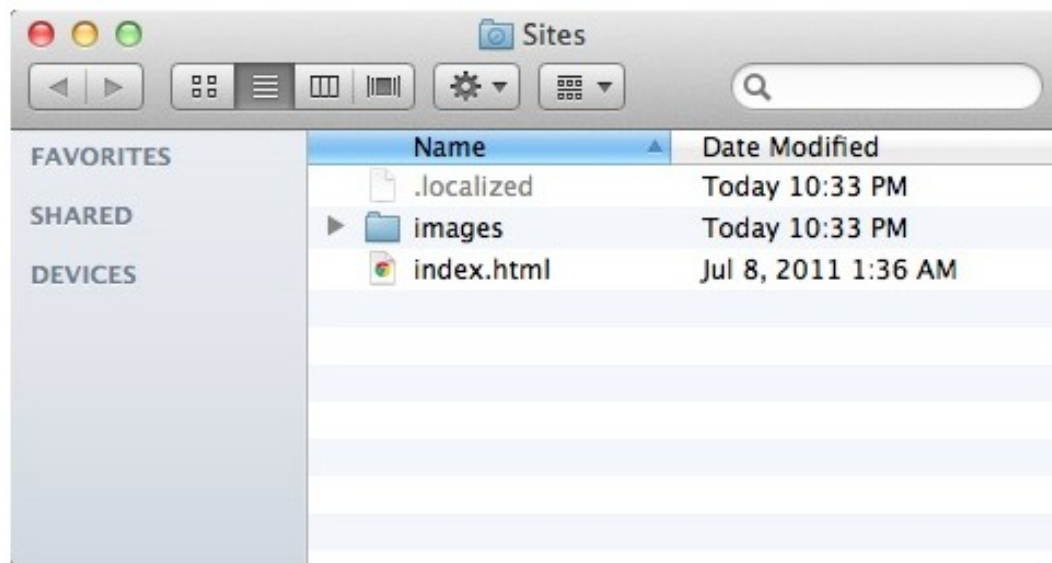
Dependiendo de la versión del sistema operativo que tengamos, aparecerá un botón para abrir la carpeta con el contenido Web y un enlace desde donde podremos comprobar que el servidor se está ejecutando correctamente.

Si lo queremos comprobar nosotros directamente, desde el navegador tendremos que acceder a la dirección IP de nuestra máquina en la red seguida de nuestro nombre de usuario, como podemos ver en la imagen inferior:



Si estuviésemos utilizando un servidor XAMPP desde el navegador podríamos haber accedido directamente a la dirección: "<http://localhost>".

El directorio para el contenido Web se encuentra normalmente dentro de la carpeta principal con el nombre de "Sites":

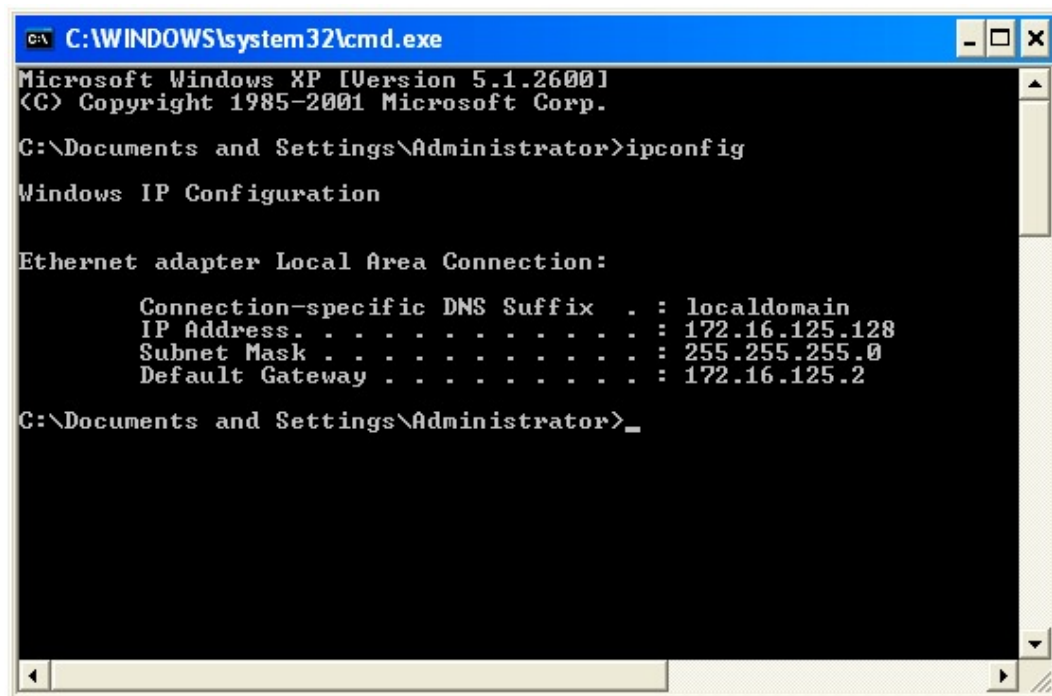


#### 1.2.4. Acceso mediante un dispositivo móvil real

Si estamos trabajando con el servidor Web disponible en **Mac**, podremos acceder a nuestras páginas Webs de forma externa simplemente conectándonos a la misma red WiFi. Para esto nos tendremos que asegurar que en el ordenador que hace de servidor no se esté ejecutando ningún Firewall que pueda bloquear el acceso desde un cliente remoto. La dirección que tendremos que utilizar será igual que la que usaríamos desde un navegador ejecutado en el mismo ordenador: la dirección IP del servidor en esa red seguida de nuestro nombre de usuario, como podemos ver en la imagen inferior:



Si nuestro servidor está corriendo en una máquina con **Linux o Windows**, entonces tendremos que seguir los siguientes pasos. En primer lugar también nos tendremos que asegurar de que no se esté ejecutando ningún Firewall que pueda bloquear el acceso (si fuera así tendríamos que darle acceso). A continuación obtendremos la dirección IP del servidor. Para esto abrimos un terminal y ejecutaremos el comando “ipconfig” (desde Windows) o “ifconfig” (desde Linux), obteniendo un resultado similar a:



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>ipconfig

Windows IP Configuration

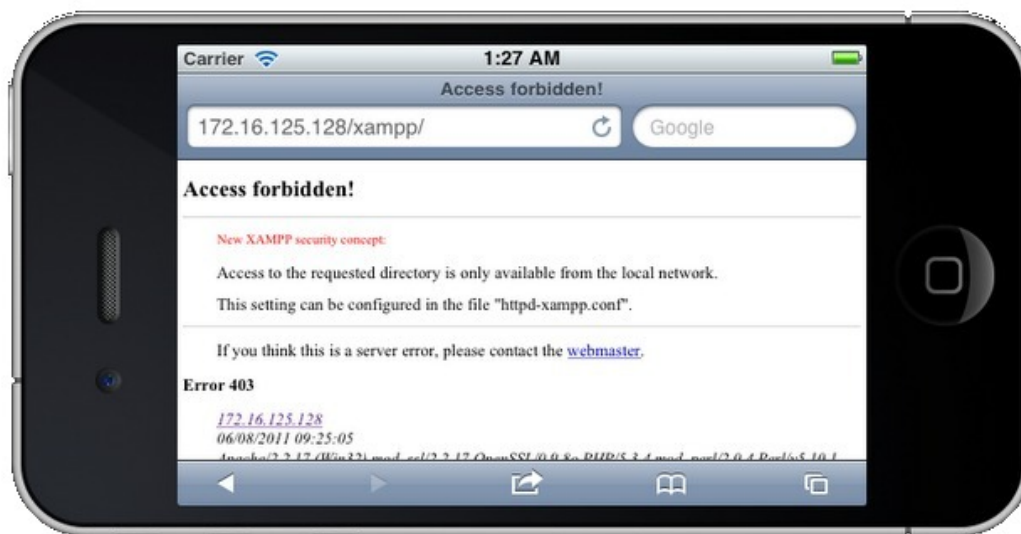
Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : localdomain
    IP Address. . . . .               : 172.16.125.128
    Subnet Mask . . . . .             : 255.255.255.0
    Default Gateway . . . . .         : 172.16.125.2

C:\Documents and Settings\Administrator>
```

Utilizando esta dirección IP y estando conectados a la misma red WiFi, ya podremos acceder a nuestro servidor Web desde un dispositivo móvil externo. Simplemente tendremos que escribir la dirección formada como “[http://172.16.125.128/mi\\_web](http://172.16.125.128/mi_web)”. Donde en primer lugar colocamos la dirección IP seguida del nombre de la carpeta donde se encuentre nuestro proyecto Web.

Es posible que obtengamos un error de seguridad como el siguiente:



En este caso tendríamos que abrir el fichero “httpd-xampp.conf” que contiene la configuración de nuestro servidor XAMPP. Lo encontraremos en “C:\xampp\apache\conf\extra” en nuestro servidor Windows y en la ruta “/opt/lampp/etc/extra” en Linux. Al final de este fichero podremos ver el siguiente trozo de texto:

```
<LocationMatch
"^(?i:(?:xampp|security|licenses|phpmyadmin|webalizer|server-status|server-info))">
    Order deny,allow
    Deny from all
    Allow from 127.0.0.0/8
    ErrorDocument 403 /error/HTTP_XAMPP_FORBIDDEN.html.var
</LocationMatch>
```

Si no nos importa la seguridad (pues es una red local), podemos abrir el acceso a todos los usuarios cambiando la cuarta línea por “Allow from All”, quedando de la forma:

```
<LocationMatch
"^(?i:(?:xampp|security|licenses|phpmyadmin|webalizer|server-status|server-info))">
    Order deny,allow
    Deny from all
    Allow from all
    ErrorDocument 403 /error/HTTP_XAMPP_FORBIDDEN.html.var
</LocationMatch>
```

Tendremos que reiniciar nuestro servidor Apache para que los cambios tengan efecto (ver en las secciones anteriores) y ya tendremos acceso desde nuestro dispositivo móvil.

### 1.2.5. Instalación del SDK de Android

Si queremos utilizar el emulador de Android para testear nuestras Webs tendremos que instalar el SDK de Android completo. Para esto descargamos el software desde

<http://developer.android.com/sdk/index.html>” y procedemos a su instalación. Este proceso simplemente requiere que descomprimos el archivo descargado en una carpeta y ejecutemos el SDK Manager (tools/android) para empezar a trabajar.

Para poder realizar simulaciones y visualizar nuestras páginas Web tendremos que crear un dispositivo virtual de Android (Android Virtual Device o AVD).

Si utilizamos Eclipse como entorno de desarrollo, podemos instalar el plugin de Android para Eclipse (ADT Plugin), el cual lo podremos encontrar en la dirección “<http://developer.android.com/sdk/eclipse-adt.html>”. Este plugin integrará el SDK y nuestro simulador en Eclipse.

### 1.2.6. Instalar Xcode

---

Si trabajamos con el sistema operativo de Mac podremos hacer uso del IDE Xcode y de los emuladores de iPhone e iPad que incorpora para testear nuestras aplicaciones en local. Este software lo podemos descargar desde el Apple Developer Center accediendo a la dirección “<http://developer.apple.com/xcode/>”.

### 1.2.7. Simuladores y Emuladores

---

Los simuladores solo tratan de reproducir el comportamiento, para que el resultado se parezca al que obtendríamos con una ejecución real. Los emuladores, por su parte, modelan de forma precisa el dispositivo (hardware y S.O.), de manera que este funcione como si estuviese siendo usado en el aparato original.

Podemos encontrar algunos simuladores online mediante los cuales realizar pruebas rápidas de nuestros proyectos, como:

- [iPhone 4 Simulator](#)
- [Test iPhone](#)
- [iPhone Tester](#)
- [Opera Mini Simulator](#)
- [Emuladores para Nokia N70 y Sony K750](#)

Para poder usar estos simuladores tendremos que tener nuestro proyecto en un servidor Web para poder acceder a través de la dirección de localhost o de la IP.

Para una completa guía de los emuladores disponibles podemos consultar: <http://www.mobilexweb.com/emulators>.

## 1.3. HTML

---

HTML, siglas de *HyperText Markup Language* (“lenguaje de marcado de hipertexto”), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el



texto con objetos tales como imágenes. El código HTML se escribe en forma de “etiquetas”, mediante las cuales podemos describir la estructura lógica y apariencia del contenido. La apariencia que podemos describir con HTML está bastante limitada, pero el código se puede complementar y mejorar mediante el uso de otros lenguajes como JavaScript o CSS.

### 1.3.1. Editores HTML

El lenguaje HTML puede ser creado y editado con cualquier editor de textos básico, como puede ser Gedit en Linux o el Bloc de notas de Windows. Existen además otros editores para la realización de sitios web con características WYSIWYG (What You See Is What You Get, o en español: “lo que ves es lo que obtienes”). Estos editores permiten ver el resultado de lo que se está editando en tiempo real, a medida que se va desarrollando el documento. Ahora bien, esto no significa una manera distinta de realizar sitios web, sino que una forma un tanto más simple ya que estos programas, además de tener la opción de trabajar con la vista preliminar, tiene su propia sección HTML, la cual va generando todo el código a medida que se va trabajando. Algunos ejemplos de editores son Adobe Dreamweaver, KompoZer o Microsoft FrontPage.

Estos editores aceleran o facilitan la creación de código HTML, pero en algunas ocasiones también generan mucho más código del necesario (como es el caso de Microsoft FrontPage). Lo ideal es tener un control total sobre el código que se escribe y utilizar estos editores sólo como una pequeña ayuda. También podemos utilizar otro tipo de editores que simplemente comprueben que el código HTML escrito es correcto (que las etiquetas y atributos son correctos, las etiquetas se cierran correctamente, etc.).

### 1.3.2. Etiquetas

Las etiquetas HTML deben de ir encerradas entre corchetes angulares “<>”, y pueden ser de dos tipos:

- Se abren y se cierran, como por ejemplo: <b>negrita</b> o <p>texto</p>.
- Sólo se abren, como <br/> o <hr/>.

En caso de que no cerremos una etiqueta que deba ser cerrada se producirá un error en la estructura del documento y probablemente también genere errores en la visualización.

Hay etiquetas que además pueden contener atributos, en este caso los atributos se deben de colocar en la etiqueta de inicio, de la forma:

```
<etiqueta atributo1="valor1" atributo2="valor2">...</etiqueta>
```

O para las etiquetas de solo apertura:

```
<etiqueta atributo1="valor1" atributo2="valor2"/>
```



### 1.3.3. Estructura básica de una Web

Un documento HTML comienza con la etiqueta **<html>** y termina con **</html>**. Dentro del documento (entre las etiquetas de principio y fin de html) hay dos zonas bien diferenciadas: el encabezamiento, delimitado por **<head>** y **</head>**, que sirve para incluir definiciones iniciales válidas para todo el documento; y el cuerpo, delimitado por **<body>** y **</body>**, donde reside la información del documento.

Las etiquetas básicas o mínimas son:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Ejemplo</title>
</head>
<body>
  ¡Hola mundo!
</body>
</html>
```

La primera línea es el DOCTYPE, o el tipo de documento que viene a continuación. En este caso se usa el estándar de HTML 4.01 (el último estándar adoptado en 1999, ya que HTML5 a fecha de 2011 sigue siendo un borrador). La siguiente etiqueta, **<html>**, define el inicio del documento HTML, e indica que lo que viene a continuación debe ser interpretado como código HTML. Como podemos ver en la última línea, se cierra la etiqueta **</html>**.

### 1.3.4. Elementos de la cabecera

La cabecera contiene información sobre el documento que no se muestra directamente al usuario. Como por ejemplo el título de la ventana del navegador, los metadatos o la hoja de estilo utilizada. Dentro de la cabecera **<head></head>** podemos encontrar:

- **<title></title>**: define el título de la página. Por lo general el título aparece en la barra de título encima de la ventana.
- **<link>**: para vincular el sitio con hojas de estilo (ver la sección de CSS para más información):

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

El atributo “rel” es requerido y describe el tipo de documento enlazado (en este caso una hoja de estilo). El atributo “type” es simplemente indicativo del tipo de hoja de estilo enlazada (en este caso CSS).

- **<style></style>**: se utiliza para añadir definición de estilo en línea. No es necesario colocarlo si se va a utilizar una hoja de estilo externa usando la etiqueta **<link>** (que es lo más habitual y recomendable). El uso correcto sería de la forma:

```
<html>
```

```
<head>
  ...
  <style type="text/css">
    Estilos CSS
  </style>
</head>
<body></body>
</html>
```

Para más información ver la sección CSS del manual.

- **<meta/>**: para indicar metadatos como la descripción de la web, los keywords, o el autor:

```
<meta name="description" content="Descripción de la web" />
<meta name="keywords" content="key1,key2,key3" />
<meta name="author" content="Nombre del autor" />
```

- **<script></script>**: permite incluir un script en la Web. El código se puede escribir directamente entre las etiquetas de `<script>` o cargar desde un fichero externo, indicando mediante el atributo `src="url del script"` la dirección del fichero. Se recomienda incluir el tipo MIME en el atributo `type`, que en el caso de código JavaScript sería `"text/javascript"`. Ejemplos:

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

<script type="text/javascript">
  Código de un script integrado en la página
</script>
```

Cuando usamos el atributo `"src"` el contenido de estas etiquetas está vacío (no encierra nada), esto es porque lo carga directamente desde el fichero indicado.

### 1.3.5. Etiquetas básicas HTML

Las etiquetas HTML que utilizaremos para crear el contenido de nuestra página deben de ir dentro de la sección `<body></body>`.

Algunas de las etiquetas HTML que más se suelen utilizar son:

- **<h1></h1>** a **<h6></h6>**: encabezados o títulos del documento con diferente relevancia, siendo `<h1>` la cabecera de mayor nivel.
- **<p></p>**: definición de un párrafo.
- **<br/>**: salto de línea.
- **<b></b>**: texto en negrita (etiqueta desaprobadada. Se recomienda usar la etiqueta **<strong></strong>**).
- **<i></i>**: texto en cursiva (etiqueta desaprobadada. Se recomienda usar la etiqueta **<em></em>**).
- **<s></s>**: texto tachado (etiqueta desaprobadada. Se recomienda usar la etiqueta **<del></del>**).
- **<u></u>**: texto subrayado.
- **<center></center>**: texto centrado.
- **<pre></pre>**: texto preformateado, respeta los espacios y saltos de línea.

- **<sup></sup>**: Superíndice
- **<sub></sub>**: Subíndice
- **<blockquote></blockquote>**: Indica una cita textual, se representa como un párrafo indexado con respecto al margen.
- **<hr/>**: Línea horizontal, usada, por ejemplo, para separar diferentes secciones.
- **<!-- comentario -->**: Comentarios en HTML. El texto del comentario no será visible en el navegador.
- **<span></span>**: Esta etiqueta no aplica ningún formato por si misma, sino que provee una forma de definir un estilo o formato a un trozo de texto. Se utiliza junto con una hoja de estilo. Por ejemplo, lo podemos utilizar para marcar palabras en algún color o con algún formato especial.

### 1.3.6. Listas

---

Para definir una lista utilizamos las siguientes etiquetas:

- **<ol></ol>**: Lista ordenada (con numeración).
- **<ul></ul>**: Lista con puntos (o viñetas).
- **<li></li>**: Elemento de una lista (tanto numerada como no numerada). Esta etiqueta debe de estar entre las etiquetas **<ol></ol>** o **<ul></ul>**.

Ejemplo de lista:

```
<ol>
  <li>Elemento 1</li>
  <li>Elemento 2</li>
</ol>
```

### 1.3.7. Tablas

---

Las tablas se definen básicamente mediante tres etiquetas:

- **<table></table>**: define una tabla.
- **<tr></tr>**: fila de una tabla, debe de estar dentro de las etiquetas de una tabla.
- **<td></td>**: celda de una tabla, debe estar dentro de una fila.

Ejemplo de una tabla:

```
<table>
  <tr>
    <td>Fila 1 izquierda</td>
    <td>Fila 1 derecha</td>
  </tr>
  <tr>
    <td>Fila 2 izquierda</td>
    <td>Fila 2 derecha</td>
  </tr>
</table>
```

Además también podemos utilizar la etiqueta **<th>** en lugar de **<td>** para indicar una

celda de “cabecera”, de esta forma el contenido será resaltado en negrita y en un tamaño ligeramente superior al normal.

En la etiqueta de apertura `<table>` podemos utilizar los siguientes atributos:

- **border=“num”**: Ancho del borde de la tabla en puntos. Si indicamos **border=“0”** tendremos una tabla cuyas divisiones no serán visibles, esta propiedad se suele utilizar para distribuir los elementos en una página Web.
  - **cellspacing=“num”**: Espacio en puntos que separa las celdas que están dentro de la tabla.
  - **cellpadding=“num”**: Espacio en puntos que separa el borde de cada celda y el contenido de esta.
  - **width=“num”**: Indica la anchura de la tabla en puntos o en porcentaje en función del ancho de la ventana. Si no se indica este parámetro, el ancho dependerá de los contenidos de las celdas.
  - **height=“num”**: Indica la altura de la tabla en puntos o en porcentaje en función del alto de la ventana. Si no se indica este parámetro, la altura dependerá de los contenidos de las celdas.
- Este atributo también se puede utilizar en las etiquetas `<tr>` para indicar la altura de cada fila de forma individual.

En las etiquetas de apertura de celda (`<td>` o `<th>`) podemos utilizar los siguientes atributos:

- **align=“pos”**: Indica como se debe alinear el contenido de la celda, a la izquierda (left), a la derecha (right), centrado (center) o justificado (justify).
- **valign=“pos”**: Indica la alineación vertical del contenido de la celda, en la parte superior (top), en la inferior (bottom), o en el centro (middle).
- **rowspan=“num”**: Indica el número de filas que ocupará la celda. Por defecto ocupa una sola fila. Este atributo se utiliza para crear celdas “multifila”, es decir, una celda que por ejemplo ocupe 3 filas. Tendremos que tener en cuenta que esa celda no se deberá de definir en las siguientes 2 filas (para esas filas se definirá una celda menos).
- **colspan=“num”**: Indica el número de columnas que ocupará la celda. Por defecto ocupa una sola columna. Este atributo se utiliza para crear celdas “multicolumna”, es decir, una celda que por ejemplo ocupe 3 columnas. Tendremos que tener en cuenta que en esa fila tendremos que definir 2 celdas menos.
- **width=“num”**: Indica la anchura de la columna en puntos o en porcentaje en función del ancho de la ventana. Si no se indica este parámetro, el ancho dependerá del tamaño de los contenidos.

### 1.3.8. Cajas (etiqueta `<div>`)

La etiqueta `<div></div>` permite crear cajas contenedoras. Estas cajas se utilizan para organizar la disposición de los elementos en la página. Es muy sencillo indicar su posición de forma absoluta o relativa en la página y crear divisiones del espacio para distribuir los elementos. Estas cajas pueden contener cualquier tipo de elemento (texto,

imágenes, etc.) u otras etiquetas `<div>` para crear subdivisiones. Se recomienda su uso junto con CSS, en vez de la etiqueta `<table>`, cuando se desea alinear contenido. Para más información consultar la sección sobre CSS.

### 1.3.9. Enlaces

---

Los enlaces permiten vincular partes del documento con otros documentos o con otras partes del mismo documento. Por ejemplo, que al pulsar con el ratón sobre un texto o sobre una imagen se nos redirija a una nueva Web con un contenido diferente.

Para crear un enlace se utiliza la etiqueta `<a href=""></a>` cuyo atributo href establece la dirección URL a la que apunta el enlace. Por ejemplo, un enlace a la Wikipedia sería de la forma:

```
<a href="es.wikipedia.org">Wikipedia</a>
```

También se pueden crear enlaces sobre otros objetos, tales como imágenes, de la forma:

```
<a href="dirección_URL"></a>
```

La etiqueta de enlace `<a>` también permite el atributo `target="_blank"`, mediante el cual indicamos que el enlace se tiene que abrir en una nueva ventana o en una pestaña nueva del navegador.

### 1.3.10. Imágenes

---

Para incluir una imagen se utiliza la etiqueta `<img src="" alt="" />`, la cual requiere el atributo `src` con la ruta en la que se encuentra la imagen. Es conveniente poner siempre el atributo `alt="texto alternativo"`, el cual indica el texto a mostrar en caso de no poder cargar la imagen y también se utiliza para opciones de accesibilidad.

Por ejemplo, para cargar una imagen de cabecera:

```

```

Además existen otros atributos interesantes como `width` y `height` para redefinir el ancho y la altura de la imagen.

### 1.3.11. Formularios

---

Los formularios permiten solicitar información al visitante de una página Web. Están compuestos por campos de diferente tipo, cuya información se enviará a una dirección URL (indicada en el código) al pulsar el botón de envío.

La declaración de formulario queda recogida por las etiquetas `<form></form>`, los cuales deben encerrar la definición de todos los campos del formulario. En la etiqueta de

apertura `<form>` tenemos que indicar los atributos básicos:

- **action=“”**: Entre comillas se indica la acción a realizar al enviar el formulario. En general se indicará el nombre de un fichero alojado en el servidor, el cual se encargará de procesar la información. Aunque también se le puede indicar una dirección de correo para que envíe directamente todo el contenido, de la forma: `“mailto:direccion_de_correo”`.
- **method=“”** (post o get): Indica el método de transferencia de las variables. El método `“post”` envía los datos de forma no visible, mientras que el método `“get”` los adjunta a la URL a la que se redirige.
- **enctype= “”**: Especifica el tipo de codificación de la información enviada. Con `method= “get”` no se realiza codificación, solo se cambian caracteres especiales como el espacio, por lo que no es necesario indicar `enctype`. Cuando el valor del atributo `“method”` es `“post”`, podemos utilizar los siguientes valores:
  - `application/x-www-form-urlencoded`: Es el valor predeterminado. Codifica todos los caracteres antes de enviarlos.
  - `multipart/form-data`: Es requerido al enviar archivos mediante un formulario. No codifica la información.
  - `text/plain`: No codifica la información, solo cambia los espacios por el símbolo `“+”`.

### Tipos de campos básicos

Para añadir campos al formulario se utiliza la etiqueta `<input/>`, esta etiqueta debe de tener siempre dos atributos:

- **name=“”**: Indica el nombre que se asigna a un determinado campo. Este nombre no aparece visible en la Web, pues se utiliza para poder distinguir cada campo al enviar la información al servidor o por correo. Es como si fuera el nombre de la variable a la que se asigna el valor del campo.
- **type=“”**: Indica el tipo de campo a utilizar. Puede ser de muchos tipos: `text`, `password`, `checkbox`, `radio`, `file`, `hidden`, `submit`, `reset`.

A continuación se describen más detalladamente los diferentes tipos de campos `<input/>` según su valor *type*:

- **type=“text”**: campo de tipo texto de una línea. Sus atributos son:
  - `maxlength= “”`: Seguido de un valor que limitará el número máximo de caracteres.
  - `size= “”`: Seguido de un valor que limitará el número de caracteres a mostrar en pantalla. A diferencia de `maxlength` este atributo no limita la longitud del texto que se puede introducir, sino que modifica el tamaño visible del campo.
  - `value= “”`: Indica el valor inicial del campo.
- **type=“password”**: Este campo funciona exactamente igual que el de tipo `“text”`, pero ocultará el texto introducido cambiando las letras por asteriscos o puntos. Sus atributos son los mismos que para `“text”`.
- **type=“checkbox”**: Este campo mostrará una casilla cuadrada que nos permitirá

marcar opciones de una lista (podremos marcar varias opciones a la vez). Para indicar que varias casillas pertenecen al mismo grupo se les debe de dar el mismo nombre para el atributo “name”. El texto que queramos que aparezca a continuación de la casilla del “checkbox” se tendrá que escribir después de cerrar la etiqueta <input/>. Sus atributos son:

- *value= “”*: Define el valor que será enviado si la casilla está marcada.
- *checked*: Este atributo es opcional, y hace que la casilla aparezca marcada por defecto. No necesita indicarle ningún valor.

Ejemplo:

```
<input type="checkbox" name="option1" value="leche" /> Leche<br/>
<input type="checkbox" name="option1" value="pan" checked/> Pan<br/>
<input type="checkbox" name="option1" value="queso" /> Queso<br/>
```

- **type=“radio”**: El campo se elegirá marcando de entre varias opciones una casilla circular. Al marcar una casilla el resto de casillas de ese grupo de desmarcarán automáticamente. Para indicar que varias casillas pertenecen al mismo grupo se les debe de dar el mismo nombre para el atributo “name” (ver ejemplo). Además debemos de indicar:
  - *value= “”*: Define el valor que será enviado si la casilla está marcada.
  - *checked*: Este atributo es opcional, y hace que la casilla aparezca marcada por defecto. Solo se podrá usar para una casilla. No necesita indicarle ningún valor.

Ejemplo:

```
<input type="radio" name="group1" value="leche" /> Leche<br/>
<input type="radio" name="group1" value="pan" checked/> Pan<br/>
<input type="radio" name="group1" value="queso" /> Queso<br/>
```

- **type=“file”**: El atributo file permite al usuario subir archivos. Necesitaremos un programa que gestione estos archivos en el servidor mediante un lenguaje diferente al HTML. El único atributo opcional que podemos utilizar es *size=“”* mediante el cual podremos indicar la anchura visual de este campo. Ejemplo:

```
<input type="file" name="datafile" size="40" />
```

- **type=“hidden”**: Este valor no puede ser modificado, pues permanece oculto. Se suele utilizar para enviar al método encargado de procesar el formulario algún dato adicional necesario para su procesamiento. Para indicar el valor de este campo utilizamos el atributo: *value = "valor"*.
- **type=“submit”**: Representa el botón de “Enviar”. Al pulsar este botón la información de todos los campos se enviará realizando la acción indicada en <form>. Mediante el atributo:
  - *value= “texto”*: podemos indicar el texto que aparecerá en el botón.
- **type=“reset”**: Al pulsar este botón se borra el contenido de todos los campos del formulario. Mediante el atributo:
  - *value= “texto”*: podemos indicar el texto que aparecerá en el botón.

## Campos de Selección

Mediante la etiqueta `<select></select>` podemos crear listas de opciones, que nos permitirán seleccionar entre una o varias de ellas. Sus atributos son:

- *name*="": Nombre del campo.
- *size*="": Número de opciones visibles a la vez. Si se indica 1 se presentará como un menú desplegable, si se indica mas de uno aparecerá como una lista con barra de desplazamiento.
- *multiple*: Permite seleccionar mas de un valor para el campo.

Las diferentes opciones de la lista se indicarán mediante la etiqueta `<option></option>`. El nombre que se visualizará debe de indicarse dentro de estas etiquetas. Mediante el atributo *value*=" " podemos indicar el valor que se enviará con el formulario. También podemos utilizar el atributo *selected* para indicar la opción seleccionada por defecto. Si no lo especificamos, siempre aparecerá como seleccionado el primer elemento de la lista.

```
<select name="Colores" multiple>
  <option value="r">Rojo</option>
  <option value="g">Verde</option>
  <option value="b">Azul</option>
</select>

<select name="Colores" SIZE="1">
  <option value="r">Rojo</option>
  <option value="g" selected>Verde</option>
  <option value="b">Azul</option>
</select>
```

## Áreas de texto

Mediante las etiquetas `<textarea></textarea>` podemos crear un campo de texto de múltiples líneas. Los atributos que podemos utilizar son:

- *name*="": Nombre del campo.
- *cols*="num": Número de columnas de texto visibles.
- *rows*="num": Número de filas de texto visibles.

### 1.3.12. Eventos

Los eventos permiten ejecutar acciones cuando sucede un determinado evento o se realiza una determinada acción. La forma de definirlos es similar a los atributos (*evento*="acción"), la acción hará referencia a una función o método en lenguaje JavaScript. Algunos de los eventos que podemos utilizar son:

- **onload**: se activa cuando el navegador termina de cargar todos los elementos de la página.
- **onunload**: se activa al cerrar una página.
- **onclick**: cuando se presiona el botón del ratón sobre un elemento.
- **ondblclick**: se activa al hacer doble clic sobre un elemento.



- **onmousedown:** se activa al presionar el botón del ratón (mientras que está presionado).
- **onmouseup:** cuando el botón del ratón es liberado.
- **onmouseover:** se dispara cuando el cursor del ratón pasa sobre un elemento.
- **onmousemove:** cuando se mueve el cursor del ratón mientras está sobre un elemento.
- **onmouseout:** se activa cuando el cursor del ratón sale fuera de un elemento (sobre el que estaba).
- **onfocus:** ocurre cuando un elemento recibe el enfoque (el cursor de escritura), ya sea con el puntero o con mediante la tecla tabulador.
- **onblur:** se dispara cuando un elemento pierde el enfoque (ya sea por hacer clic fuera o por presionar la tecla tabulador).
- **onkeypress:** ocurre cuando se presiona una tecla (dentro de un elemento, por ejemplo un campo de escritura).
- **onkeydown:** se dispara cuando una tecla es presionada (dentro de un elemento)
- **onkeyup:** cuando una tecla es soltada.
- **onsubmit:** se activa cuando un formulario es enviado.
- **onreset:** ocurre cuando un formulario es reseteado.
- **onselect:** cuando el usuario selecciona un texto en un campo de texto.
- **onchange:** ocurre cuando un control pierde el enfoque y su valor ha sido modificado desde que recibió el enfoque.

Ejemplo de uso:

```
<script type="text/javascript">
  function saveText() {
    // acciones JavaScript
  }
</script>

<textarea id="myarea" cols="80" rows="15" onkeyup="saveText()"></textarea>
```

### 1.3.13. Símbolos HTML

Los caracteres especiales como signo de puntuación, letras con tilde o diéresis, o símbolos del lenguaje; se deben convertir en entidades HTML para que se muestren correctamente en un navegador. La siguiente es una lista de caracteres españoles junto con algunos símbolos especiales y su correspondiente entidad HTML:

á	&aacute;		Á	&Aacute;
é	&eacute;		É	&Eacute;
í	&iacute;		Í	&Iacute;
ó	&oacute;		Ó	&Oacute;
ú	&uacute;		Ú	&Uacute;
ü	&uuml;		Ü	&Uuml;

ñ	&ntilde;		Ñ	&Ntilde;
espacio en blanco	&nbsp;		€	&euro;
< (Menor que)	&lt;		> (Mayor que)	&gt;
&	&amp;		° (grados)	&deg;

Algunos servidores realizan esta conversión automáticamente, pero en general es aconsejable escribir los símbolos directamente. Para obtener una lista mucho más completa de símbolos podemos buscar en Google: “HTML symbols” o visitar la siguiente dirección “<http://www.ascii.cl/htmlcodes.htm>”.

## 1.4. CSS

El nombre hojas de estilo en cascada viene del inglés *Cascading Style Sheets*, del que toma sus siglas. CSS es un lenguaje usado para definir la presentación o la apariencia de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). CSS se creó para separar el contenido de la forma, a la vez que permite a los diseñadores mantener un control mucho más preciso sobre la apariencia de las páginas. El W3C (*World Wide Web Consortium*) es el encargado de formular la especificación de las hojas de estilo que sirven de estándar para los navegadores.

En versiones antiguas de HTML se debía de añadir el formato dentro de las propias etiquetas, para indicar por ejemplo su color o tamaño. Esto obligaba a tener que especificar el mismo formato en todas las etiquetas para tener un diseño consistente, además, al cambiar un formato también había que cambiarlo para todas las etiquetas.

Cuando se utiliza CSS, las etiquetas HTML no deberían proporcionar información sobre cómo serán visualizadas. La información de la hoja de estilo será la que especifique cómo se han de mostrar: color, fuente, alineación del texto, tamaño, etc.

Las ventajas de utilizar CSS (u otro lenguaje de estilo) son:

- Control centralizado de la apariencia de un sitio web completo, con lo que se agiliza de forma considerable la actualización del mismo.
- Los navegadores permiten a los usuarios especificar su propia hoja de estilo local, que será aplicada a un sitio web, con lo que aumenta considerablemente la accesibilidad. Por ejemplo, personas con deficiencias visuales pueden configurar su propia hoja de estilo para aumentar el tamaño del texto o remarcar más los enlaces.
- Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestre o, incluso, a elección del usuario. Por ejemplo, para ser impresa o mostrada en un dispositivo móvil.
- El documento HTML en si mismo es más claro de entender y se consigue reducir considerablemente su tamaño.

### 1.4.1. Adjuntar una hoja de estilo

---

La información de estilo puede ser adjuntada de tres formas diferentes:

- **Hoja de estilo externa:** es una hoja de estilo que está almacenada en un archivo diferente al archivo HTML (por ejemplo llamado “estilo.css”). Esta es la manera de programar más potente y la que deberemos de utilizar por defecto, porque separa completamente las reglas de formateo para la página HTML. La hoja de estilo debe de ser enlazada con el código HTML de la forma:

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="estilo.css"/>
  ...
</head>
...
```

- **Hoja de estilo interna:** es una hoja de estilo que está incrustada dentro del documento HTML. En general, la única vez que se usa una hoja de estilo interna, es cuando se quiere diferenciar con algún estilo uno de los ficheros HTML de nuestra Web. Este código debe de estar incluido en la sección de cabecera y entre las etiquetas <style>. Las etiquetas de comentario “<!-- ... -->” sirven para que los navegadores antiguos, que no soportan CSS, no incluyan ese texto en el cuerpo de la página. La forma de incluir este código sería de la forma:

```
<html>
<head>
  <STYLE type="text/css">
    <!--
      H1 {color:blue; text-align:center}
    // -->
  </STYLE>
</head>
...
```

- **Estilo en línea (inline):** es un método para insertar el lenguaje de CSS directamente dentro de una etiqueta HTML. Esta manera de proceder no es totalmente adecuada. El incrustar la descripción del formateo dentro del documento de la página Web, a nivel de código, se convierte en una manera larga, tediosa y poco elegante de resolver el problema de la programación de la página. Este modo de trabajo se podría usar de manera ocasional si se pretende aplicar un formateo con prisa, al vuelo. La forma de incluir un estilo inline sería:

```
<h1 style="color:blue; text-align:center">...</h1>
```

### 1.4.2. Definición de estilos para etiquetas HTML

---

Si lo que queremos es dar formato o redefinir una etiqueta HTML existente, usaríamos la sintaxis:

```
etiqueta {
  estilo CSS 1;
  estilo CSS 2;
```

```
...
}
```

En "etiqueta" pondríamos el nombre de la etiqueta (por ejemplo "h1", "p", etc. pero sin los signos < >) y los estilos que definirían esa etiqueta irían encerrados entre las llaves "{...}".

También podemos redefinir varias etiquetas a la vez, separándolas por comas, de la forma:

```
etiqueta1, etiqueta2, etiqueta3 {
    estilos CSS
}
```

O redefinir etiquetas "dentro" de otras etiquetas. En este caso el estilo CSS solo se aplicará cuando la etiqueta redefinida se encuentre dentro de la etiqueta contenedora:

```
contenedor etiqueta {
    estilos CSS
}
```

Por ejemplo, una etiqueta <span> dentro de una sección <p>:

```
p span {
    estilos CSS
}
```

### 1.4.3. Identificadores y Clases

A veces tenemos varias etiquetas del mismo tipo pero queremos aplicar diferentes estilos según donde estén. Para esto usamos los identificadores y las clases.

La principal diferencia entre ellos es que los identificadores tienen que ser únicos en todo el documento HTML mientras que las clases pueden repetirse todas las veces que queramos. Los identificadores se suelen usar con etiquetas "neutras" como <div> o <span> para marcar partes de un documento y después aplicar diferentes estilos a cada una (como por ejemplo identificar la cabecera, un logotipo, el menú principal, etc.).

En el siguiente ejemplo podemos ver como podemos indicar el identificador o la clase de una etiqueta HTML. Esto se hace con los parámetros "id" y "class" respectivamente, y se pueden aplicar a cualquier etiqueta:

```
<div id="capitulo2">
  <p>...</p>
  <p class="parrafogris">...</p>
  <p>...</p>
  <p class="parrafogris">...</p>
</div>
```

En este ejemplo "capitulo2" sería una sección única marcada en el documento en la cual

podemos aplicar un estilo concreto. El estilo de la clase “parrafogris” se aplicaría sobre las etiquetas “p” indicadas.

En nuestra hoja de estilos, para indicar los estilos que definen un identificador tenemos que escribir el nombre del identificador precedido por una almohadilla “#”, de la forma:

```
#identificador {  
    estilos CSS  
}
```

Esta definición de estilos se puede combinar con lo que hemos visto en la sección anterior. Por ejemplo, para aplicar un estilo en concreto a la etiqueta “etiqueta1” que esté dentro del “identificador1”:

```
#identificador1 etiqueta1 {  
    estilos CSS  
}
```

Para aplicar estilos a clases es parecido pero precediendo el nombre de la clase con un punto “.” en vez de una almohadilla. Por ejemplo:

```
.clase {  
    estilos CSS  
}
```

La definición de una clase también la podemos combinar con lo que hemos visto en la sección anterior. Además también podemos aplicar los estilos de la clase sólo a una determinada etiqueta:

```
etiqueta1.clase1 {  
    estilos CSS  
}
```

En este caso sólo se aplicaría el estilo a las etiquetas “etiqueta1” que se marque que son de la clase “clase1”, por ejemplo: <etiqueta1 class=“clase1”>...</etiqueta1>. Si intentáramos aplicar esta clase a una etiqueta diferente no funcionaría.

#### 1.4.4. Estilos CSS básicos

---

La sintaxis básica para definir un estilo es:

```
atributo: valor;
```

Los diferentes estilos siempre se separan con punto y coma, y después del nombre se pone dos puntos (y no un igual “=”, que es un error típico al confundirse con el HTML).

Muchos de los valores que podemos aplicar a un atributo de estilo son **unidades de medida**, por ejemplo, el valor del tamaño de un margen o el tamaño de la fuente. Las unidades de medida que podemos utilizar son: pixels (px), puntos (pt), centímetros (cm) y

pulgadas (in).

A continuación se incluye un resumen de los principales estilos CSS y los valores que se les pueden aplicar:

### FUENTES:

- **color:** valor RGB o nombre de color  
Ejemplos: color: #009900; color: red;  
Sirve para indicar el color del texto. Lo admiten casi todas las etiquetas de HTML. No todos los nombres de colores son admitidos en el estándar, es aconsejable entonces utilizar el valor RGB. Algunos de los principales nombres de colores son: white, black, gray, blue, red, green o yellow, para más nombres podemos consultar la dirección "[http://www.w3schools.com/cssref/css\\_colornames.asp](http://www.w3schools.com/cssref/css_colornames.asp)".
- **font-size:** xx-small | x-small | small | medium | large | x-large | xx-large | Unidades de CSS  
Ejemplos: font-size: 12pt; font-size: x-large;  
Sirve para indicar el tamaño de las fuentes de manera más rígida y con mayor exactitud.
- **font-family:** serif | sans-serif | cursive | fantasy | monospace | Todas las fuentes habituales  
Ejemplos: font-family: arial, helvetica; font-family: fantasy;  
Con este atributo indicamos la familia de tipografía del texto. Los primeros valores son genéricos (serif, sans-serif, etc.), es decir, los navegadores las comprenden y utilizan las fuentes que el usuario tenga en su sistema.  
También se pueden definir con tipografías normales. Si el nombre de una fuente tiene espacios se utilizan comillas para que se entienda bien.
- **font-weight:** normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900  
Ejemplos: font-weight: bold; font-weight: 200;  
Sirve para definir la anchura de los caracteres, o dicho de otra manera, para poner negrita con total libertad.  
Normal y 400 son el mismo valor, así como bold y 700.
- **font-style:** normal | italic | oblique  
Ejemplos: font-style: normal; font-style: italic;  
Es el estilo de la fuente, que puede ser normal, itálica u oblicua. El estilo "oblique" es similar al "italic".

### PÁRRAFOS:

- **line-height:** normal | unidades CSS  
Ejemplos: line-height: 12px; line-height: normal;  
El alto de una línea, y por tanto, el espaciado entre líneas. Es una de esas características que no se podían modificar utilizando HTML.
- **text-decoration:** none | underline | overline | line-through  
Ejemplos: text-decoration: none; text-decoration: underline;  
Establece la decoración de un texto, si está subrayado, sobre-rayado o tachado.

- **text-align:** left | right | center | justify  
Ejemplos: text-align: right; text-align: center;  
Sirve para indicar la alineación del texto. Es interesante destacar que las hojas de estilo permiten el justificado de texto, aunque recuerda que no tiene por que funcionar en todos los sistemas.
- **text-indent:** Unidades CSS  
Ejemplos: text-indent: 10px; text-indent: 2in;  
Un atributo que sirve para hacer sangrado o márgenes en las páginas.
- **text-transform:** capitalize | uppercase | lowercase | none  
Ejemplos: text-transform: none; text-transform: capitalize;  
Nos permite transformar el texto, para que tenga la primera letra en mayúsculas de todas las palabras, o todo en mayúsculas o minúsculas.

### FONDO:

- **background-color:** Un color, con su nombre o su valor RGB  
Ejemplos: background-color: green; background-color: #000055;  
Sirve para indicar el color de fondo de un elemento de la página.
- **background-image:** El nombre de la imagen con su camino relativo o absoluto  
Ejemplos: background-image: url(mármol.gif); background-image: url(http://www.url.com/fondo.gif)  
Permite colocar una imagen de fondo en cualquier elemento de la página.

### CAJAS (<div> o <table>):

- **width:** Unidades CSS | Porcentaje  
**height:** Unidades CSS | Porcentaje  
Ejemplos: width: 50px; width: 100%; height: 15px;  
Permiten indicar el ancho y altura de un elemento. Se pueden aplicar sobre muchos elementos, como tablas, etiquetas div, imágenes, párrafos “p”, etc. Con algunas etiquetas no funciona, tampoco sirve para indicar espaciado (padding), bordes o márgenes.
- **margin-left:** Unidades CSS  
Ejemplos: margin-left: 1cm; margin-left: 0,5in;  
Indica el tamaño del margen izquierdo.
- **margin-right:** Unidades CSS  
Ejemplos: margin-right: 5%; margin-right: 1in;  
Define el tamaño del margen derecho.
- **margin-top:** Unidades CSS  
Ejemplos: margin-top: 0px; margin-top: 10px;  
Indica el tamaño del margen superior.
- **margin-bottom:** Unidades CSS  
Ejemplos: margin-bottom: 0pt; margin-top: 1px;  
Indica el tamaño del margen inferior.
- **margin:** <arriba> <derecha> <abajo> <izquierda> | <arriba> <derecha> <abajo> | <arriba-abajo> <izquierda-derecha> | <los 4 márgenes>

Ejemplos: margin: 4px 2px 1px 2px; margin: 4px;

También podemos utilizar el estilo “margin” para indicar todos los márgenes a la vez, esta etiqueta nos permite indicarle desde 4 valores (para cada uno de los márgenes), hasta 1 valor (para aplicarlo sobre todos los márgenes).

- **padding-left:** Unidades CSS  
Ejemplos: padding-left: 0.5in; padding-left: 1px;  
Indica el espacio insertado, por la izquierda, entre el borde del elemento-continente y el contenido de este. Es parecido a el atributo cellpadding de las tablas. El espacio insertado tiene el mismo fondo que el fondo del elemento-continente.
- **padding-right:** Unidades CSS  
Ejemplos: padding-right: 0.5cm; padding-right: 1pt;  
Indica el espacio insertado, en este caso por la derecha, entre el borde del elemento-continente y el contenido de este. Es parecido a el atributo cellpadding de las tablas. El espacio insertado tiene el mismo fondo que el fondo del elemento-continente.
- **padding-top:** Unidades CSS  
Ejemplos: padding-top: 10pt; padding-top: 5px;  
Indica el espacio insertado, por arriba, entre el borde del elemento-continente y el contenido de este.
- **padding-bottom:** Unidades CSS  
Ejemplos: padding-bottom: 0.5cm; padding-bottom: 1pt;  
Indica el espacio insertado, en este caso por abajo, entre el borde del elemento-continente y el contenido de este.
- **padding:** <arriba> <derecha> <abajo> <izquierda> | <arriba> <derecha> <abajo> | <arriba-abajo> <izquierda-derecha> | <los 4 márgenes>  
Ejemplos: padding: 4px 2px 1px 2px; padding: 4px;  
Al igual que para “margin”, esta etiqueta nos permite indicarle desde 4 valores (espaciado hasta cada uno de los bordes por separado), hasta 1 valor (para indicar el mismo espaciado hasta todos los bordes).
- **border-color:** color RGB o nombre de color  
Ejemplos: border-color: red; border-color: #ffccff;  
Para indicar el color del borde del elemento de la página al que se lo aplicamos. Se puede poner colores por separado con los atributos border-top-color, border-right-color, border-bottom-color, border-left-color.
- **border-style:** none | dotted | solid | double | groove | ridge | inset | outset  
Ejemplos: border-style: solid; border-style: double;  
El estilo del borde, los valores significan: none=ningún borde, dotted=punteado, solid=solido, double=doble borde, desde groove hasta outset son bordes con varios efectos 3D.
- **border-width:** Unidades CSS  
Ejemplos: border-width: 10px; border-width: 0.5in;  
El tamaño del borde del elemento al que lo aplicamos.
- **border:** <grosor> <tipo> <color>  
Ejemplo: border: 2px solid red;



De esta forma podemos indicar las tres propiedades del borde a la vez. También podemos utilizar `border-top`, `border-right`, `border-bottom` y `border-left` para indicar estas tres propiedades para un borde en concreto.

- **float:** none | left | right

Ejemplo: `float: right;`

Sirve para alinear un elemento a la izquierda o la derecha haciendo que el texto se agrupe alrededor de dicho elemento.

- **clear:** none | right | left

Ejemplo: `clear: right;`

Indica que no se permiten elementos por ese lado del objeto. Por ejemplo, si tenemos varias cajas una a continuación de otra, al poner "`clear:left`" en la última caja, esta pasaría a la siguiente línea.

#### 1.4.5. Pseudo-clases

Una pseudo-clase te permite tener en cuenta diferentes condiciones o eventos al definir una propiedad para una etiqueta HTML, por ejemplo si un enlace ha sido visitado o si el cursor del ratón está sobre un elemento. Algunas de las pseudo-clases que podemos utilizar son:

- *a:link* - enlace que no ha sido explorado por el usuario.
- *a:visited* - se refiere a los enlaces ya visitados.
- *a:active* - enlace seleccionado con el ratón.
- *a:hover* - enlace con el puntero del ratón encima, pero no seleccionado.
- *a:focus* - enlace con el foco del sistema. También puede ser usado para un input.
- *p:first-letter* - primera letra de un párrafo.
- *p:first-line* - primera línea de un párrafo.

Utilizando estos elementos podemos configurar por ejemplo:

```
a:hover { color: blue; }
a:visited { color: darkgreen; }
p:first-letter {color: green; font-size: x-large;}
```

#### 1.4.6. Capas

Normalmente la posición de los elementos de una página es **relativa**, es decir, depende de los demás elementos de la página. Por ejemplo, un párrafo estará más abajo si antes de él hay más párrafos o elementos. Debido a esto, normalmente cuando se quería colocar elementos en un sitio concreto, se recurría al uso de tablas (invisibles, solo para estructurar).

Con CSS podemos colocar los elementos en posición **absoluta**, es decir, indicando el tamaño y coordenadas exactas en las que queremos que se coloque. Para organizar la disposición en una Web con CSS se suele usar el elemento `<div>`. Además se le suele dar un identificador único a cada uno, mediante el cual, desde la hoja de estilo, podemos

configurar su disposición. También podemos colocar estos elementos con posición relativa a otro elemento que lo contenga, por ejemplo, un `<div>` dentro de otro.

Es común en el diseño Web crear contenedores `<div>` generales en una posición absoluta o centrados en la página, con un tamaño definido, los cuales se utilizarán para contener y disponer el resto de elementos de nuestra Web. Estos otros elementos se pueden alinear de forma sencilla con una alineación “relativa” a sus contenedores. Por ejemplo un contenedor para la cabecera que contenga un par de contenedores para la disposición de logotipo y el texto de cabecera.

## Distribución

Para indicar el tipo de distribución o disposición de un elemento lo hacemos mediante el atributo “**position: valor**”. El cual puede tomar los valores:

- *absolute*: La posición del elemento no depende de ninguna otra etiqueta. Esta posición se calcula desde la esquina superior izquierda de la página.
- *fixed*: Al igual que el anterior la posición es absoluta, pero el elemento se queda fijo en el sitio al hacer “scroll”.
- *relative*: Posición relativa a su elemento contenedor. Es la propiedad predeterminada.
- *static*: Al igual que el anterior la posición es relativa, pero no podemos redimensionar (por ejemplo) el objeto.

## Posición

Para indicar la posición concreta de una capa utilizamos los atributos: *top*, *bottom*, *left* y *right*, de la forma:

```
top: <posición>;
left: <posición>;
```

Normalmente sólo se utilizan un par de ellos, como *top* y *left*, o *bottom* y *right*. La posición se especifica mediante unidades de CSS, como por ejemplo en “px”, aunque también admite porcentajes.

Un ejemplo de la definición de una capa sería:

```
#micapa {
  position: absolute;
  top: 200px;
  left: 150px;
  width: 175px;
  height: 175px;
  border: solid 1px blue;
  text-align: center;
  color: gray;
}
```

En nuestro documento HTML tendremos un elemento definido de la forma: `<div id="micapa"> ... </div>`, dentro del cual colocaremos texto u otros elementos.

## Orden

A veces tenemos varias capas, unas por encima de otras, y queremos especificar un orden, para poder controlar las ocultaciones entre capas. Para esto usamos el z-index, de la forma:

```
z-index: <índice>;
```

Las capas con un índice de Z-index mayor aparecerán por encima de las capas con un z-index menor.

### 1.4.7. Más información

---

Para obtener una referencia mucho más completa sobre las hojas de estilo podemos consultar alguna de las siguientes Webs:

- Referencia detallada de todos los estilos: <http://www.w3schools.com/cssref/default.asp>
- Especificaciones: <http://www.w3.org/Style/CSS/>
- Tutoriales: <http://www.desarrolloweb.com/manuales/manual-css-hojas-de-estilo.html>

## 2. Ejercicios - Introducción al desarrollo web para móviles

En esta primera sesión vamos a realizar algunos ejemplos para practicar con HTML y CSS. Descarga la plantilla para los ejercicios desde [sesion01-ejercicios.zip](#). Al descomprimirlo aparecerá un archivo por cada ejercicio.

### 2.1. Ejercicio 1 - Tablas

En este primer ejercicio vamos a hacer una pequeña web de ejemplo utilizando las tablas de HTML para estructurar el contenido. Nuestra tabla va a tener cuatro filas y una única columna. Cada una de estas cuatro celdas tendrá un identificador (*header*, *nav*, *content*, *footer*) para poder asignar el estilo apropiado.

En la primera fila vamos a crear la cabecera (identificador *header*) con el logo del curso (logo.jpg) y el texto "DADM". La imagen tendrá un alto de 50 píxeles y el texto será de color blanco con tamaño de 16 puntos.

A continuación incluiremos la barra de navegación con identificador "nav". Dentro de esta celda vamos a crear a su vez otra tabla con una única fila y tres columnas. Cada columna será de la clase "navElement" y contendrán los textos "Inicio", "Contenidos" y "Profesorado". Como estilo indicaremos que la clase "navElement" tenga el color de fondo "#f0f0f0" y que cambie al color "#005682" cuando el ratón pase por encima.

En la tercera celda de la tabla principal estará el cuerpo del documento, a esta celda le pondremos como identificador "content". Además para los 4 últimos párrafos crearemos una lista del tipo UL (lista no ordenada). Cada uno de los párrafos los tendremos que identificar como elementos de esta lista (mediante las etiquetas LI).

A la última celda le pondremos como identificador "footer" e incluiremos el texto "Dept. Ciencia de la Computación e IA". Como estilo le aplicaremos el color gris.

### 2.2. Ejercicio 2 - Cajas Div

En este ejercicio vamos a modificar la Web que hemos hecho para el primer ejercicio, y en lugar de utilizar una tabla para la disposición lo vamos a hacer mediante elementos DIV.

Para la cabecera crearemos un DIV con identificador "header", en el que incluiremos el logo del curso (logo.jpg, con un alto de 50px) y el texto "DADM".

Para la barra de navegación crearemos un elemento DIV con identificador "nav", y dentro de este a su vez crearemos 3 cajas tipo DIV de la clase "navElement". Las cajas tendrán los textos "Inicio", "Contenidos" y "Profesorado".

En la sección central solo tendremos que escribir la etiqueta DIV de apertura, con el

identificador "content", y la etiqueta DIV de cierre al final de este contenido.

El pie de página lo crearemos también utilizando una etiqueta DIV con el identificador "footer" y el texto "Dept. Ciencia de la Computación e IA".

Ahora tenemos que terminar de ajustar los estilos CSS para que la página se vea correctamente. Es importante fijarse en los cambios de esta hoja de estilo con respecto a la utilizada para las tablas. Para los elementos "header", "nav", "content" y "footer" definiremos un ancho del 100% e indicaremos que no se permiten elementos por su lado izquierdo (es decir, que deben de estar en una nueva línea, esto lo haremos mediante `clear: left;`). Para el estilo "navElement" indicaremos que se tiene que situar a continuación del anterior (en la misma línea, esto lo haremos mediante `float: left;`).

### 2.3. Ejercicio 3 - Terminando la Web de ejemplo

En este ejercicio vamos a terminar la web de ejemplo que hemos hecho para el ejercicio 2, añadiéndole el contenido de los enlaces que faltan. En primer lugar, dado que no tenemos plantilla, copiamos el fichero resultado del ejercicio anterior y lo renombramos a "sesion01-ejercicio3.html". En este fichero vamos a modificar las opciones de menú para añadir enlaces a *Inicio* (un enlace a este mismo fichero), *Contenidos* (enlace a "sesion01-ejercicio3\_contenidos.html") y *Profesorado* (enlace a "sesion01-ejercicio3\_profesorado.html").

Al añadir estos enlaces se modificará su apariencia, por lo que tendremos que modificar la hoja de estilo. Añadiremos dos nuevos estilos ".navElement a" para indicar que el color de los enlaces es negro y que no se dibuje la línea de subrayado del enlace (`text-decoration: none;`). Y otro estilo ".navElement a:hover" para que el enlace cambie a color blanco cuando el ratón pase por encima.

El siguiente paso es pasar todos estos estilos a un fichero independiente, llamado "sesion01-ejercicio3\_css.css". Simplemente tendremos que crear este fichero y cortar y pegar en él todos los estilos que ya tenemos creados. En el fichero principal HTML tendremos que cargar esta hoja de estilo, quedando solo una línea (`<link href="sesion01-ejercicio3_css.css" rel="stylesheet" type="text/css" />`).

Ahora vamos a crear los dos ficheros HTML que nos faltan. Para esto realizaremos dos copias del fichero HTML principal y las renombraremos a "sesion01-ejercicio3\_contenidos.html" y "sesion01-ejercicio3\_profesorado.html". En cada una de estas copias solo tendremos que cambiar el contenido de la zona central. Para el fichero de contenidos buscaremos el índice general de contenidos del curso y lo añadiremos en una lista no numerada (UL). Y para el fichero de profesorado añadiremos también en una lista no numerada los nombres de los profesores.

Por último vamos a hacer que al cambiar de sección se quede marcado el enlace correspondiente. Esto lo haremos añadiendo el estilo "visited" a la clase del enlace actual visitado, por lo que este enlace tendrá dos estilos. Para añadir más de un estilo a un

elemento HTML lo podemos hacer separando los estilos con espacios, de la forma: `<div class="navElement visited">`. Finalmente definimos el estilo "visited" en la hoja de estilo con el color de fondo "#005682".

## 3. HTML5 y CSS3

---

### 3.1. HTML 5

---

La quinta revisión del lenguaje de programación HTML pretende remplazar al actual (X)HTML, corrigiendo problemas con los que los desarrolladores web se encuentran, así como rediseñar el código y actualizándolo a nuevas necesidades que demanda la web de hoy en día.



Actualmente se encuentra en modo experimental, lo cual indica la misma W3C; aunque ya es usado por múltiples desarrolladores web por sus avances, mejoras y ventajas.

A diferencia de otras versiones de HTML, los cambios en HTML5 comienzan añadiendo semántica y accesibilidad implícitas. Establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Algunos de ellos son técnicamente similares a las etiquetas `<div>` y `<span>`, pero tienen un significado semántico, como por ejemplo `<nav>` (bloque de navegación del sitio web) o `<footer>`. Otros elementos proporcionan nuevas funcionalidades a través de una interfaz estandarizada, como los elementos `<audio>` y `<video>`.

Algunos elementos de HTML 4.01 han quedado obsoletos, incluyendo elementos puramente de presentación, como `<font>` y `<center>`, cuyos efectos se deben de realizar utilizando CSS. También hay un renovado énfasis en la importancia del scripting DOM para el comportamiento de la web.

#### 3.1.1. Navegadores que lo soportan

---

Actualmente, de los navegadores de escritorio, el que mayor soporte da es Google Chrome, seguido muy de cerca por Mozilla Firefox y Apple Safari. El que menor

compatibilidad ofrece es Internet Explorer.

Para comprobar la compatibilidad de un navegador podemos visitar la Web “<http://www.html5test.com/>” donde se realiza un test de todas las funcionalidades de HTML5.

### 3.1.2. Doctype

---

El doctype es el encargado de indicarle al navegador el tipo de documento que está abriendo, con el fin de renderizar la pagina de manera correcta. Por ejemplo, el doctype de HTML 4 es:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Para HTML 5 el doctype se ha simplificado mucho y además es compatible con las versiones anteriores de HTML:

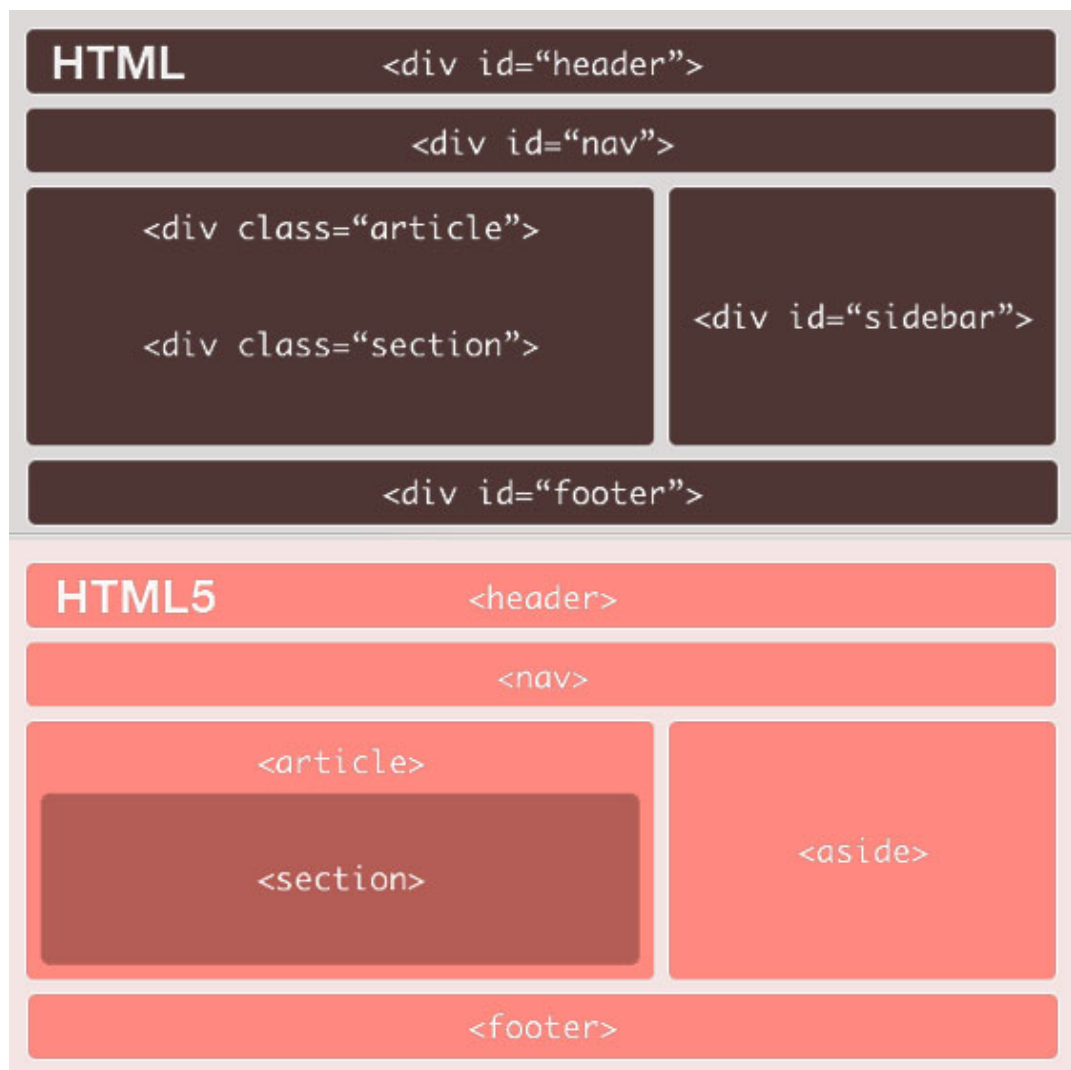
```
<!DOCTYPE html>
```

### 3.1.3. Mejor estructura

---

Hasta ahora se utilizaba de forma abusiva la etiqueta <div> y las tablas para estructurar una web en bloques. El HTML5 nos brinda nuevas etiquetas que perfeccionan esta estructuración. Estas nuevas etiquetas introducen un nuevo nivel semántico que hace que la estructura de la web sea más coherente y fácil de entender. Además los navegadores podrán darle más importancia a determinadas secciones, facilitándole además la tarea a los buscadores, así como cualquier otra aplicación que interprete sitios Web. En la siguiente imagen se puede ver una comparación entre la estructuración realizada con HTML (hasta la versión 4) y HTML 5:





Las Webs se dividirán en los siguientes elementos:

- **<section></section>**: Se utiliza para representar una sección “general” dentro de un documento o aplicación, como un capítulo de un libro. Puede contener subsecciones y si lo acompañamos de las etiquetas `<h1>..<h6>` podemos estructurar mejor toda la página creando jerarquías del contenido, algo muy favorable para el buen posicionamiento web. Por ejemplo:

```
<section>
  <h1>Introducción al elemento section</h1>
  <p>El elemento section se usa para agrupar contenido relacionado entre
si.</p>
  <p>...</p>
</section>
```

- **<article></article>**: Se usa para definir contenido autónomo e independiente, con la

intención de ser reutilizable de modo aislado. El elemento `article` puede contener uno o varios elementos `section`. Si por ejemplo nuestro contenido puede ser redistribuido como RSS y sigue manteniendo íntegro su significado, entonces, probablemente es un elemento `article`. De hecho, el elemento `article` está especialmente indicado para sindicación. El elemento `article` es especialmente útil para posts en blogs, noticias en prensa digital, comentarios y posts en foros.

La especificación de HTML5 añade además que el elemento `article` debe ser usado por widgets autónomos como; calculadoras, relojes, marcos de clima y cosas por el estilo. Hay que analizar si el contenido de un widget es autónomo, independiente y puede ser reutilizable o incluso sindicado.

- **`<aside></aside>`**: Representa una sección de la página que abarca un contenido no directamente relacionado con el contenido que lo rodea, por lo que se le puede considerar un contenido independiente. Dentro de este elemento pueden incluirse: elementos publicitarios, barras laterales, grupos de elementos de la navegación, efectos tipográficos, u otro contenido que se considere separado del contenido principal de la página.
- **`<header></header>`**: Es la cabecera de la página o de una sección. Existe una diferencia clave entre el elemento `header` y el uso habitual del término `header` (o cabecera) utilizado comúnmente para situar los elementos del encabezado de un sitio web.

Una página web debe definir un `header` principal donde normalmente irá el logo o el nombre del sitio y seguramente un menú de navegación, pero además puede —y debe— definir otros elementos `<header>` dentro de los elementos `<section>`:

```
<section>
  <header>
    <h1>Cabecera se sección</h1>
  </header>
  <p>...</p>
</section>
```

- **`<nav></nav>`**: Contiene información sobre la navegación por el sitio web, usualmente una lista de enlaces. Este elemento debe de ser utilizado solo para la navegación principal del sitio y no para enlaces externos por ejemplo. Normalmente el elemento `nav` aparece dentro de un elemento `header` o `footer`.
- **`<footer></footer>`**: Representa el pie de una sección o la parte inferior de una página Web, contiene información acerca de la página/sección que poco tiene que ver con el contenido de la página, como el autor, el copyright, la fecha de última modificación, etc. Igual que con la etiqueta `<header>`, este elemento también se puede utilizar dentro de una sección para indicar información como: quien lo ha escrito, información de propiedad intelectual, enlaces, etc.

Es muy importante tener en cuenta que estas etiquetas no indican su posición en la página Web, sino su valor semántico. Por ejemplo, las etiquetas `header`, `footer` o `aside` no indican que esos elementos tengan que ir en la parte superior, inferior o lateral del contenido principal, sino que indican su función en esa sección o en esa página.

Además debemos tener en cuenta que estas nuevas etiquetas se comportan igual que una etiqueta de caja `<div>` por lo que podemos aplicarles los mismos estilos CSS. Podemos redefinir la propia etiqueta o aplicarle una clase, por ejemplo:

```
header { width: 100%; padding: 10px; margin-bottom: 20px; }
.webheader { height: 30px; border: 1px solid gray; background-color: silver; }
.sectionheader { font-size: 20px; }
```

### 3.1.4. Formularios

La estructura de los formularios con HTML 5 no varía con respecto a las anteriores de HTML. Pero sí que se añaden muchos nuevos tipos de campos que podemos utilizar, cada uno específico para cada tipo de dato.

En el caso de que utilicemos estas características y el navegador no sea compatible, simplemente las ignorará sin causarnos mayores problemas. También podemos detectar si el navegador soporta una determinada característica y en caso negativo emularla mediante código JavaScript (para más información ver la sección “Detectar funcionalidades de HTML5”).

Los nuevos tipos de campos son:

- **search**: se utiliza para crear cajas de búsqueda. Tiene un aspecto similar a un campo de tipo texto. Además podemos utilizar el atributo *results*=“num” para añadir un histórico de búsquedas con “num” resultados. De momento no funciona ni en Firefox ni en Chrome.

```
<label for="busqueda">Búsqueda con histórico: </label>
<input type="search" name="busqueda" id="busqueda" results="5"/>
```

- **number**: campo numérico, incorpora dos botones para incrementar o decrementar el valor del campo. Además podemos usar atributos para asignar restricciones, como *min*=“”, *max*=“” o *step*=“”. El valor es almacenado en el atributo *value*=“”.



- **range**: campo numérico que permite seleccionar mediante una barra de desplazamiento un valor entre dos valores predeterminados, especificados mediante *min*=“” y *max*=“”. El valor actual es almacenado en el atributo *value*=“”. Además podemos indicar el incremento mínimo al desplazar la barra con *step*=“”.



- **color**: permite seleccionar un color. De momento solo funciona en Opera 11.
- **tel**: es un campo de texto normal pero valida si el valor introducido es un número telefónico (todavía no funciona).

- **url**: valida direcciones web. De momento requiere “http://” o “http:” simplemente. En algunos navegadores cambia el aspecto del campo.
- **email**: valida direcciones de email. Funciona en algunos navegadores, mostrando además un aspecto diferenciado. Para iPhone además adapta el teclado.
- **date**: seleccionar un día en un calendario. En algunos navegadores (para móvil) aparece un calendario desplegable (como en Opera).

Date:

- **month**: selector para meses. En algunos navegadores (para móvil) aparece un calendario desplegable.

Month:

- **week**: selector para semanas. En algunos navegadores (para móvil) aparece un calendario desplegable.

Week:

- **time**: campo con formato para hora.

Time:

- **datetime**: permite seleccionar fecha y hora.

Datetime:

- **datetime-local**: permite seleccionar fechas y hora local.

Datetime-local:

- **output**: este campo se utiliza para visualizar valores, por ejemplo el valor de un campo “range”. De momento solo funciona en Opera. Se suele utilizar junto con la propiedad “onformchange” para actualizar su valor:

```
<output onformchange="value = rango.value">0</output>
```

Además, junto con estos nuevos tipos de campos, también se han incorporado nuevos tipos de atributos. Estos nuevos atributos son aplicables a la mayoría de los campos:

- **Autocomplete**: La mayoría de los navegadores incorporan la funcionalidad de autocompletar algunos campos de los formularios con valores introducidos anteriormente. Esta funcionalidad no siempre resulta útil, sobre todo si alguien nos roba nuestro portátil o dispositivo móvil. La nueva especificación de HTML5 nos permite desactivar el autocompletado en un formulario completo o solo en campos específicos. El atributo *autocomplete* nos permite definir dos valores: “on” o “off”.

```
<form action="formaction.php" autocomplete="off">
  ...
```

```
</form>
```

El código anterior desactivaría el autocompletado de todo el formulario. Si por el contrario solo queremos desactivar el autocompletado de un solo campo podemos especificarlo así:

```
<input type="text" name="cuentadelbancosupersecreta" autocomplete="off" />
```

Esta funcionalidad no se puede emular mediante código JavaScript.

- **Placeholder:** El atributo *placeholder*=“texto” se utiliza para colocar el valor de su texto dentro del campo a modo de ayuda. Si se focaliza dicho campo, se elimina el *placeholder*. Si abandonamos el campo sin añadir ningún valor, se vuelve a añadir el *placeholder*. Esta funcionalidad siempre ha requerido del uso de JavaScript para ser llevado a cabo, pero con la nueva especificación este comportamiento puede definirse de la forma:

```
<label for="referer">Nombre</label>
<input id="referer" name="referer" type="text"
      placeholder="Escribe tu nombre completo" />
```

Obteniendo como resultado:

**Nombre**

- **Required:** Una de las tareas de validación más extendidas es la de los campos requeridos. La nueva especificación de HTML5 incluye el atributo *required* que nos sirve para definir si un campo es requerido o no. Si un campo requerido está en blanco el formulario no será enviado y además avisará con un mensaje:

```
<label for="username">Su nombre de usuario</label>
<input id="username" name="username" type="text" required/>
```

**NOTA:** Es un error grave de seguridad validar los formularios únicamente desde el lado del cliente, es imprescindible además realizar la validación en el servidor.

- **Autofoco:** El atributo de autofocus asigna el foco (cursor de escritura) al campo indicado en cuando la página se ha cargado. Sólo se puede asignar a un elemento de la página. De momento este atributo solo lo soportan Safari, Chrome y Opera. Firefox e IE, lo ignoran, pero se puede emular fácilmente mediante código JavaScript (ver la siguiente sección “Detectar funcionalidades de HTML5”).

```
<input name="b" autofocus/>
```

- **List:** Usando el atributo list con un elemento `<input>` podemos especificar una lista de opciones. Esto permite al usuario seleccionar un valor de la lista o escribir uno que no esté en ella (este tipo de elemento se suele llamar *Combo Boxes*). Los elementos de la lista se deben de indicar utilizando otro nuevo elemento de HTML5, el `<datalist>`. El cual simplemente nos permite crear una lista de valores. En algunos navegadores estas funcionalidades todavía no funcionan, como en Chrome.

```
<label for="diasemana">Día de la semana:</label>
<input type="text" name="diasemana" id="diasemana" list="dias"/>
```

```
<datalist id="dias">
  <option value="Lunes" />
  <option value="Martes" />
  <option value="Miércoles" />
  <option value="Jueves" />
  <option value="Viernes" />
  <option value="Sábado" />
  <option value="Domingo" />
</datalist>
```

Con este código obtendríamos un resultado similar al de la siguiente imagen:



- **Pattern (formatting):** Este atributo se utiliza para validar la entrada del usuario mediante expresiones regulares. En la dirección [“http://es.wikipedia.org/wiki/Expresi%C3%B3n\\_regular”](http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular) podemos obtener más información sobre las expresiones regulares. Ejemplo de uso (en Firefox y Chrome funciona):

```
<label for="cp">Código Postal</label>
<input id="cp" name="cp" pattern="[\d]{5}(-[\d]{4})" />
```

### 3.1.5. Mark

HTML5 también introduce un conjunto nuevo de elementos *inline*, solo que ya no se llaman elementos inline sino *text-level semantics* o semántica a nivel de texto. Uno de ellos es la etiqueta mark. Cuando realizamos una búsqueda en ciertos sitios, los elementos encontrados en la página aparecen remarcados para facilitar su localización. Hasta ahora el estilo se aplicaba con etiquetas `<span>`, pero esta solución no es semántica. Es ahí donde entra en escena la nueva etiqueta `<mark>`:

```
<h1>Resultados de la búsqueda de la palabra 'anillo'</h1>
<ol>
  <li>El señor de los <mark>anillo</mark>s...</li>
  <li>el cliente compró este <mark>anillo</mark></li>
</ol>
```

Si queremos podemos redefinir el estilo de esta nueva etiqueta de la misma forma que lo hacíamos con las etiquetas de HTML, por ejemplo, para cambiar el color de fondo a rojo:

```
<mark style="background-color: red;">anillo</mark>
```

```
mark { background-color: red; }
```

### 3.1.6. Canvas

El elemento canvas puede definirse como un entorno para crear imágenes dinámicas. Utilizando su API en JavaScript podemos manipular el elemento canvas para dibujar en él y crear gráficos dinámicos de todo tipo (incluidas interfaces de aplicaciones web completas). La API, aunque de momento está en desarrollo, la podemos encontrar en: <http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html>

Para empezar a usarlo lo único que hay que especificar son sus dimensiones. El texto que escribamos entre la apertura y cierre de la etiqueta canvas solamente será interpretado por navegadores que no soporten esta etiqueta:

```
<canvas id="myCanvas" width="360" height="240">
  <p>Tu navegador no soporta canvas</p>
</canvas>
```

El resto de trabajo con canvas se ha de realizar con código JavaScript. Primero debemos referenciar este elemento y adquirir su contexto (que de momento solo está disponible para 2D):

```
var canvas = document.getElementById( 'myCanvas' );
var context = canvas.getContext( '2d' );
```

Una vez adquirimos el contexto podemos empezar a dibujar. La API bidimensional ofrece muchas de las herramientas que podemos encontrar en cualquier aplicación de diseño gráfico: trazos, rellenos, gradientes, sombras, formas y curvas Bézier. Los principales métodos disponibles son:

- **fillRect(x, y, width, height):** dibuja un rectángulo relleno de color según el estilo activado.
- **strokeRect(x, y, width, height):** dibuja solo el borde de un rectángulo, el interior será transparente.
- **clearRect(x, y, width, height):** borra el área indicada.
- **beginPath():** inicializa el dibujado de un “trazo”.
- **closePath():** cierra la figura creando una línea desde el último punto hasta el primero.
- **moveTo(x, y):** mueve el puntero del trazo hasta las coordenadas indicadas (para poder seguir dibujando).
- **lineTo(x, y):** dibuja un trazo desde la posición actual hasta las coordenadas indicadas.
- **stroke():** dibuja el trazo indicado desde el último “beginPath()”.
- **fill():** cierra el trazo definido desde el último “beginPath()” y lo rellena.
- **arc(x, y, radius, startAngle, endAngle, anticlockwise):** dibuja un arco con centro en “x, y” y el radio definido. Los ángulos se definen en radianes (radianes = (PI/180)\*grados) y el último parámetro es un valor booleano.
- **quadraticCurveTo(controlx, controly, x, y):** dibuja una curva de bezier cuadrática.
- **bezierCurveTo( control1x, control1y, control2x, control2y, x, y):** dibuja una curva

de bezier cúbica.

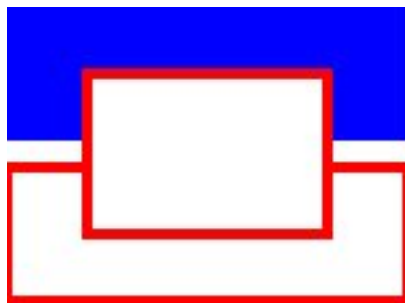
- **drawImage(x, y)**: dibuja una imagen (como objeto JavaScript) en el canvas.
- **createImageData(width, height)**: crea un objeto ImageData como un array de píxeles para ser manipulado como un array de enteros.
- **getImageData(x, y, w, h)**: carga un objeto ImageData a partir del dibujo actual para ser manipulado.
- **putImageData(imageData, x, y)**: mapea los valores de un objeto ImageData en el dibujo actual.
- **strokeText(string, x, y)**: dibuja una cadena de texto usando solo su borde.
- **fillText(string, x, y)**: dibuja una cadena de texto.

A continuación mostramos un ejemplo de dibujado en un objeto canvas una vez capturado su contexto:

```
// Primero definimos las propiedades con las que vamos a dibujar
context.fillStyle = '#0000ff'; // color de relleno azul
context.strokeStyle = '#ff0000'; // color de borde rojo
context.lineWidth = 4; // grosor de línea

// Y a continuación dibujar algunas figuras
context.fillRect(0, 0, 150, 50); // rectángulo relleno
context.strokeRect(0, 60, 150, 50); // rectángulo solo borde
context.clearRect(30, 25, 90, 60); // borrar área del canvas
context.strokeRect(30, 25, 90, 60); // Orden de coordenadas: izqda,
arriba, ancho, largo
```

Obteniendo finalmente un resultado similar a:



Webs muy importantes están cambiando sus contenidos a canvas y dejando de usar Flash, como Slideshare (ver <http://www.slideshare.net/AmitRanjan/slideshare-is-html5-now>).

### 3.1.7. Audio

El nuevo elemento audio permite insertar archivos sonoros en diferentes formatos, incluyendo mp3 y ogg. Además provee de una interfaz de control sobre la reproducción del mismo con una API en JavaScript sin necesidad de plugins de ningún tipo (como Flash). Añadir un reproductor de audio en HTML5 es muy simple:

```
<audio src="archivo.mp3" controls>
```



```
<p>Tu navegador no soporta el elemento audio</p>
</audio>
```

En Firefox obtendríamos un resultado similar a:



El texto que se encuentra entre las etiquetas audio solo es tenido en cuenta por navegadores que no soporten la nueva etiqueta. El atributo “controls” indica al navegador que muestre los controles de reproducción. En caso de no activarlo no se visualizaría nada, pero podríamos controlar la reproducción mediante funciones JavaScript, de la forma:

```
<audio id="player" src="archivo.mp3"></audio>
<button
onclick="document.getElementById('player').play();">Reproducir</button>
<button
onclick="document.getElementById('player').pause();">Pausa</button>
<button onclick="document.getElementById('player').volume += 0.1;">Subir
Volumen</button>
<button onclick="document.getElementById('player').volume -= 0.1;">Bajar
Volumen</button>
```

También podemos usar los atributos “autoplay” y “loop” para que se auto-reproduzca y para que se cree un bucle de reproducción infinito:

```
<audio src="archivo.mp3" autoplay loop></audio>
```

El formato de audio a utilizar vendrá impuesto por el navegador usado y no por el estándar:

Códec	IE<=9	Firefox	Chrome	Safari	Opera
Ogg Vorbis	no	sí	sí	no	sí
WAV PCM	no	sí	sí	sí	sí
MP3	sí	no	sí	sí	sí
AAC	sí	no	sí	sí	sí
Speex	no	no	sí	no	no

Como puede verse, combinando Vorbis y MP3 podremos ofrecer audio a todos los navegadores mayoritarios. Existe una forma de definir más de un archivo de audio para la etiqueta audio, en lugar de usar el atributo “src”, utilizaremos la etiqueta “source” para poder definir múltiples archivos. Estas etiquetas se irán leyendo de arriba a abajo hasta que el navegador encuentre un formato soportado. De esta manera podremos complacer

las necesidades de todos los usuarios sin discriminar a ningún navegador.

```
<audio controls>
  <source src="archivo.ogg" type="audio/ogg" />
  <source src="archivo.mp3" type="audio/mpeg" />
  <object type="application/x-shockwave-flash"
data="player.swf?soundFile=archivo.mp3">
    <param name="movie" value="player.swf?soundFile=archivo.mp3" />
    <a href="archivo.mp3">Descarga el archivo de audio</a>
  </object>
</audio>
```

En este ejemplo hemos añadido además una tercera línea con un reproductor Flash por si no fuesen soportados ninguno de los formatos anteriores, y un link directo de descarga para aquellos que tampoco soporten Flash. Así estaremos ofreciendo nuestro contenido a todos los navegadores y dispositivos manteniendo unas buenas prácticas en cuanto a accesibilidad del contenido se refiere.

### 3.1.8. Vídeo

La nueva especificación de HTML5 soporta la inclusión de vídeo empotrado en las páginas web de forma nativa. El elemento *video* no especifica el formato del mismo sino que el uso de uno u otro vendrá impuesto por el fabricante del navegador:

Códec	IE≥9	Firefox	Chrome	Safari	Opera
Ogg Theora	no	sí	sí	no	sí
H.264	sí	no	no	sí	no
VP8	no	sí	sí	no	sí

El elemento *video* dispone de los atributos “autoplay”, “loop” y “preload”, para activar la auto-reproducción, para indicar que se reproduzca en bucle y para activar/desactivar la precarga del vídeo. Asimismo puedes utilizar los controles que te ofrece el navegador de forma nativa utilizando el atributo *controls* o bien puedes ofrecer tus propios controles en JavaScript. Dado que el vídeo ocupa un espacio, también podremos definir sus dimensiones con los atributos “width” y “height”. E incluso podemos indicar una imagen para que se muestre antes de la reproducción mediante el atributo “poster”:

```
<video src="archivo.mp4" controls width="360" height="240"
poster="poster.jpg"> </video>
```

Con lo que obtendríamos un resultado similar a:



Para dar soporte a todos los navegadores, podemos especificar diferentes archivos en diferentes formatos. Además podemos usar el mismo truco que usábamos con el elemento audio para seguir dando soporte al *plugin* de Flash a través de la etiqueta *object*, e incluso incluir un link de descarga:

```
<video controls width="360" height="240" poster="poster.jpg">
  <source src="archivo.ogv" type="video/ogg" />
  <source src="archivo.mp4" type="video/mp4" />
  <object type="application/x-shockwave-flash" width="360" height="240"
data="player.swf?file=archivo.mp4">
    <param name="movie" value="player.swf?file=archivo.mp4" />
    <a href="archivo.mp4">Descarga la película</a>
  </object>
</video>
```

### 3.1.9. Geolocalización

La geolocalización es la forma de obtener tu posición en el mundo y si quieres, compartir esta información. Existen muchas maneras de descubrir donde te encuentras, por tu dirección IP, la conexión de red inalámbrica, la torre de telefonía móvil por la que se conecta tu móvil, o usando directamente el posicionador GPS.

HTML5 incorpora una nueva funcionalidad para facilitar esta tarea, que dependerá de que el navegador le de soporte. Está disponible a partir de las versiones de Opera 10.6, Firefox 3.5, Chrome 5, Safari 5 e Internet Explorer 9.

```
if (navigator.geolocation)
{
```

```

    navigator.geolocation.getCurrentPosition(showPosition);
}

function showPosition( position )
{
    var lat = position.coords.latitude;
    var lng = position.coords.longitude;

    alert( "Latitud: " + lat + ", longitud: " + lng );
}

```

### 3.1.10. Almacenamiento Offline

El almacenamiento web está ampliamente soportado por los navegadores modernos, tanto en plataforma escritorio como en plataforma móvil, Android 2.1+, iPhone 3.1+, iPad 4.2+, Opera Mobile 11.00+, Palm WebOS 1.4+ y BlackBerry 6.0+, Chrome 4.0+, Firefox 3.5+, IE 8.0+, Opera 10.5+ y Safari 4.0+.

#### Tipos de almacenamiento

El almacenamiento web ofrece dos áreas de almacenamiento diferentes, el almacenamiento local (localStorage) y el almacenamiento por sesión (sessionStorage), que difieren en alcance y tiempo de vida. Los datos alojados en un almacenamiento local es solo accesible por dominio y persiste aún cuando se cierre el navegador. El almacenamiento por sesión es por ventana y su tiempo de vida está limitado a lo que dure la ventana (o pestaña).

Los datos se almacenan de forma muy sencilla, por pares clave/valor, de la forma:

```

// Para almacenamiento persistente en local:
localStorage.setItem("miValor", valor);

// Para almacenamiento por sesión:
sessionStorage.setItem("miValor", valor);

```

Para recuperarlos posteriormente solo tenemos que hacer:

```

var miValor = localStorage.getItem("miValor");
var miValor = sessionStorage.getItem("miValor");

```

Las variables guardadas con sessionStorage sólo se mantendrían en caso de que cambiemos de página o que el navegador se refresque, mientras que localStorage guardaría los datos aunque el navegador sea cerrado.

También podemos borrar los valores almacenados, indicando un valor en concreto o todos ellos:

```

localStorage.remove("miValor");
localStorage.clear();

```

#### Offline Application Cache (appCache)

Esta nueva característica de HTML5 permite ejecutar aplicaciones Web aun cuando no estamos conectados a Internet. Al visitar por primera vez una página web (que use appCache) el navegador descarga y guarda todos los archivos necesarios para esa página. La siguiente vez que la visitemos el navegador usará directamente los archivos descargados (a no ser que estemos conectados y se compruebe que hay una versión más actual de la Web).

El principal componente del appCache es el archivo de manifiesto (manifest file), un archivo de texto con la lista de archivos que el navegador cliente debe almacenar. En primer lugar, para usar esta característica debemos de indicar el archivo de manifiesto en la etiqueta de apertura HTML:

```
<html manifest="app.manifest">
```

Este fichero debe de empezar con el texto CACHE MANIFEST. A continuación en cada nueva línea indicaremos un recurso a almacenar (usando URLs absolutas o relativas), además podemos poner comentarios anteponiendo el símbolo “#”.

```
CACHE MANIFEST
# Esto es un comentario
index.html
js/scripts.js
css/estilos.css
imgs/logo.gif
imgs/imagen1.jpg
```

Una vez cargada la página, la única petición que realizará el navegador será por el fichero de *manifest*. Aunque solo haya cambiado un letra del fichero, se descargarán todos los recursos de nuevo. Para asegurarnos que servimos la última versión de nuestra página cuando realizamos cambios, la forma más sencilla y segura es actualizar el fichero de manifiesto con un comentario indicando la fecha de la última actualización (o un número de versión, etc.), de la forma:

```
CACHE MANIFEST
# Actualizado el 2011-10-12
```

Para más información podéis consultar las fuentes:

- <http://www.w3.org/TR/offline-webapps/>
- <http://www.w3.org/TR/html5/offline.html>

### 3.1.11. Detectar funcionalidades de HTML5

*Modernizr* es una librería de JavaScript con licencia MIT de código abierto que detecta si son compatibles elementos de HTML5 y CSS3. Podemos descargar la librería desde “<http://www.modernizr.com/>”. Para utilizarla solo hay que incluir en el <head> de tu página de la forma:

```
<head>
  <script src="modernizr.min.js"></script>
</head>
```

*Modernizr* se ejecuta automáticamente, no es necesario llamar a ninguna función. Cuando se ejecuta, se crea un objeto global llamado *Modernizr*, que contiene un *set* de propiedades *Booleanas* para cada elemento que detecta. Por ejemplo si su navegador soporta elementos *canvas*, la propiedad de la librería “*Modernizr.canvas*” será “*true*”. Si tu navegador no soporta los elementos *canvas*, la propiedad será “*false*”, de la forma:

```
if (Modernizr.canvas) {
  // sí que hay soporte
} else {
  // no hay soporte para canvas
}
```

Para comprobar elementos de un formulario también podemos crearnos dos simples funciones que validan el soporte para diferentes tipos de inputs y atributos:

### Comprobar si un input es soportado

Con la siguiente función podemos comprobar si un navegador soporta o no los nuevos tipos de inputs:

```
function inputSupports(tipo) {
  var input = document.createElement('input');
  input.setAttribute('type', tipo);
  if (input.type == 'text') {
    return false;
  } else {
    return true;
  }
}
```

Por lo que podemos usarlo de la siguiente forma:

```
if (!inputSupports('range')) {
  // Input tipo range no soportado
}
```

### Comprobar si un atributo es soportado

Para comprobar si hay soporte para un atributo

```
function attrSupports(el, attr) {
  var telement = document.createElement(el);
  if (attr in telement) {
    return true;
  } else {
    return false;
  }
}
```

Por lo que podemos usarlo para comprobar, por ejemplo, los atributos autofocus,

placeholder o required:

```
if (!attrSupports('input', 'autofocus')) {  
    document.getElementById('search_string').focus();  
}  
if (!attrSupports('input', 'placeholder')) {  
    // Atributo placeholder no soportado  
}  
if (!attrSupports('input', 'required')) {  
    // Atributo required no soportado  
}
```

### 3.2. CSS3

La especificación de CSS3 viene con interesantes novedades que permitirán hacer webs más elaboradas y más dinámicas, con mayor separación entre estilos y contenidos. Dará soporte a muchas necesidades de las webs actuales, sin tener que recurrir a trucos de diseñadores o lenguajes de programación.



Aunque CSS3 está todavía en fase de desarrollo, la mayoría de navegadores ya dan soporte a casi todos los nuevos estilos, como Firefox, Chrome o Safari. Por el contrario Internet Explorer no ha empezado a incorporar estos nuevos elementos hasta la versión 9.

Las principales propiedades nuevas en CSS3 son:

- Selectores de atributos y propiedades
- Nuevas pseudo-clases
- Formatos de color: colores HSL, colores HSLA, colores RGBA, Opacidad
- Bordes: border-color, border-image, border-radius, box-shadow

- Fondos: background-origin, background-clip, background-size, capas con múltiples imágenes de fondo
- Texto: text-shadow, text-overflow, rotura de palabras largas, Web Fonts, creación de múltiples columnas de texto
- Modelo de caja básico: overflow
- Transiciones y transformaciones

A continuación veremos con más detalle cada una de estas nuevas propiedades.

### 3.2.1. Nuevos selectores de atributos

En primer lugar encontramos 3 nuevos selectores de atributos:

- **elemento[atributo^="valor"]**: Selecciona los elementos con ese atributo y que su valor comienza por la cadena de texto indicada en "valor".
- **elemento[atributo\$="valor"]**: Selecciona los elementos con ese atributo y que su valor termina por la cadena de texto indicada en "valor".
- **elemento[atributo\*="valor"]**: Selecciona los elementos con ese atributo y que su valor contiene la cadena de texto indicada en "valor".

Por ejemplo:

```
// Selecciona todos los enlaces que apunten a una dirección de correo:
a[href^="mailto:"]{...}
// Selecciona todos los enlaces que apuntan a páginas .php
a[href$=".php"]{...}
// Selecciona todos los enlaces que lleven a una página que contenga la
palabra ejemplo:
a[href*="ejemplo"]{...}
```

También incorpora nuevas formas de seleccionar etiquetas adyacentes:

- **h1 + h2{...}**: Etiquetas inmediatamente adyacentes.
- **h1 ~ h2{...}**: Selector general de hermanos. Válido cuando <h2> se encuentre después de <h1>, pero puede haber otras etiquetas de por medio.

Ejemplo:

```
<h1>Título</h1>
<h2>Subtítulo adyacente</h2>

<h1>Título</h1>
<p> párrafo de separación</p>
<h2>Subtítulo con selector general de hermanos</h2>
```

También podemos indicar atributos específicos de una etiqueta, con:

- **etiqueta1[atributo1="valor1"]**: seleccionaría todas las etiquetas "etiqueta1" que contengan un atributo llamado "atributo1" cuyo valor sea igual a "valor1". Por ejemplo, si queremos indicar un estilo para todas las etiquetas input que sean de tipo texto:



```
input[type="text"] {
  background: #eee;
}
```

### 3.2.2. Nuevas pseudo-clases

Una pseudo-clase es un estado o uso predefinido de un elemento al que se le puede aplicar un estilo independientemente del estilo aplicado al de su estado por defecto. En CSS3 se han añadido muchas nuevas pseudo-clases para facilitar a los programadores el uso de algunos estilos avanzados en el diseño de páginas Web. Las nuevas pseudo-clases son:

- **:nth-child(n)** - Fija el aspecto de una ocurrencia específica del elemento nodo hijo especificado. Por ejemplo, el tercer elemento nodo hijo de una lista sería “li:nth-child(3)”. Además se pueden usar pequeñas expresiones como parámetro para por ejemplo seleccionar todos los elementos impares: “nth-child(2n+1)” los pares “nth-child(2n)”, etc. Los elementos impares y pares también se pueden seleccionar usando “nth-child(odd)” y “nth-child(even)”
- **:nth-last-child(n)** - igual que “:nth-child(n)” pero empezando a contar desde el final.
- **:nth-of-type(n)** - Fija la apariencia de una ocurrencia específica del elemento con el tipo de selector especificado en un elemento padre. Por ejemplo la segunda lista no ordenada sería ul:nth-of-type(2). También permite los mismos parámetros que “:nth-child( # )”.
- **:nth-last-of-type(n)** - igual que “:nth-of-type(n)” pero empezando a contar desde el final.
- **:first-child** - Fija el aspecto del primer elemento de un tipo de selector solo si es el primer nodo hijo de su elemento padre, por ejemplo la primera etiqueta <li> de una lista <ol>.
- **:last-child** - Ultimo elemento de una lista de elementos de un tipo dado.
- **:first-of-type** - Selecciona el primer elemento de un tipo concreto dentro de la lista de hijos.
- **:last-of-type** - Selecciona el último elemento de un tipo.
- **:only-child** - Selecciona el elemento si es el único elemento hijo.
- **:only-of-type** - Selecciona el elemento si es el único elemento hijo de ese tipo.
- **:empty** - Selecciona los elementos que no tienen hijos (incluyendo nodos de texto).
- **:enabled** - Selecciona los elementos de la interfaz que tengan el estado “enable”.
- **:disabled** - Selecciona los elementos de la interfaz que tengan un estado “disabled”.
- **:not(s)** - Selecciona los elementos que no coincidan con el selector especificado.
- **:lang(language)** - nos permite especificar estilos que dependen del idioma especificado por la propiedad language (en, sp, etc.)

Ejemplos de uso:

```
tr:nth-child(even) {
  background: silver;
```

```

}
tr:nth-child(odd) {
    background: white;
}
p:lang(en) {
    color: gray;
    font-style: italic;
}

```

## Formularios

Además también se han añadido nuevas pseudo-clases que podemos usar en los formularios para aplicar un formato según el estado de un campo. Estas propiedades van en concordancia con los nuevos campos introducidos en HTML5 (ver la sección de formularios de HTML5). estas son:

- **:valid** - campo válido (dependerá del tipo de campo).
- **:invalid** - campo inválido (dependerá del tipo de campo).
- **:required** - campo requerido (marcado con el atributo “required”).
- **:optional** - campo opcional (campo no marcado con el atributo “required”).
- **:checked** - elemento marcado (o checked, válido para radio button o checkbox).
- **:in-range** - valor dentro del rango indicado (para campos numéricos o de rango).
- **:out-of-range** - valor fuera de rango (para campos numéricos o de rango).
- **:read-only** - campo de solo lectura.
- **:read-write** - campo de lectura / escritura.

Algunos ejemplos de uso:

```

<head>
  <style>
    #form1 input:valid { background:lightgreen; }
    #form1 input:invalid { border-color:red; }
    #form1 specialInput input:valid { background:green; }
  </style>
</head>
<body>
  <form id="form1" name="form1" method="post" action="formaction.php">
    <p>Nombre:
      <input type="text" name="nombre" id="nombre" required/>
    </p>
    <p>Usuario:
      <specialInput>
        <input type="text" name="usuario" id="usuario" required/>
      </specialInput>
    </p>
  </form>
</body>

```

En este ejemplo cabe destacar la etiqueta “specialInput”, que no es ninguna etiqueta existente, sino una nueva etiqueta que hemos creado para aplicar un formato especial.

Además podemos aplicar estas pseudo-clases en cadena y hacer cosas como:

```

input:focus:required:invalid {
    background: pink url(ico_validation.png) 379px 3px no-repeat;
}

```

```
}  
input:required:valid {  
    background-color: #fff; background-position: 379px -61px;  
}
```

Dado que Internet Explorer 6-8 no soporta la mayoría de pseudo-clases se han desarrollado algunas librerías de JavaScript que realizan las mismas funciones para estos navegadores, como “selectivizr” que podréis descargar de su página oficial “<http://selectivizr.com/>”.

### 3.2.3. Color

En CSS3 se han incorporado nuevas formas para definir los colores:

- **rgba( red, green, blue, opacity );** - Color RGBA. El valor de opacidad debe de estar entre 0 y 1, siendo 0 totalmente transparente. Por ejemplo, podemos usarlo de la forma:

```
background-color: rgba(255, 115, 135, 0.5);  
color: rgba(255, 115, 135, 0.5);
```

- **hsl( hue, saturation, lightness );** - Modelo de color HSL.
- **hsla(hue, saturation, lightness, alpha);** - Modelo de color HSLA.
- **cmymk(cyan, magenta, yellow, black);** - Modelo de color CMYK.
- **opacity: 0.5;** - También podemos indicar el valor de transparencia u opacidad por separado, debiendo de estar este valor entre 0 y 1, siendo 0 totalmente transparente y 1 totalmente opaco. Para dar también soporte a Internet Explorer usaremos: “*filter:alpha(opacity=50);*”.

### 3.2.4. Bordes

En CSS3 se han incorporado cuatro nuevas propiedades para dar formato a los bordes de una caja. Estas propiedades no están todavía plenamente soportadas en todos los navegadores, por lo que para que funcione en la mayoría de ellos tendremos que usar también las propiedades nativas del navegador (simplemente añadiremos los prefijos -webkit- y -moz-). Las nuevas propiedades son:

- **border-radius:** permite crear cajas con esquinas redondeadas. Hasta ahora esto solo se podía hacer insertando imágenes que simularan esta característica. Ahora lo podemos hacer de una forma mucho más sencilla:

```
-webkit-border-radius: 30px;  
-moz-border-radius: 30px;  
border-radius: 30px;
```

Además también podemos indicar cada uno de los bordes por separado:

```
-moz-border-radius-topleft: 10px;  
-moz-border-radius-topright: 20px;  
-moz-border-radius-bottomright: 30px;
```

```
-moz-border-radius-bottomleft: 40px;
-webkit-border-radius: 10px 20px 30px 40px;
border-radius: 10px 20px 30px 40px;
```

- **border-image:** este nuevo estilo nos permite usar una imagen como borde de una caja. Tenemos que indicar tres atributos: la imagen a utilizar, el grosor y la forma de aplicar la imagen (stretch, repeat, round, none). Ejemplo de uso:

```
-webkit-border-image: url(imagen.png) 27 repeat;
-moz-border-image: url(imagen.png) 27 repeat;
border-image: url(imagen.png) 27 repeat;
```

El resultado dependerá de la imagen que utilicemos para el borde, pero por ejemplo podríamos obtener resultados como el siguiente:



- **border-color:** Permite crear degradados en los bordes de una caja indicando la secuencia de colores del degradado (píxel a píxel y de dentro hacia fuera), de la forma:

```
-webkit-border-bottom-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-webkit-border-top-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-webkit-border-left-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-webkit-border-right-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-moz-border-bottom-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-moz-border-top-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-moz-border-left-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
-moz-border-right-colors:#555 #666 #777 #888 #999 #aaa #bbb #ccc;
border: 8px solid #000;
```

Con lo que obtendríamos un resultado similar a:



- **box-shadow:** Permite dar sombra a elementos de bloque. Tiene 4 atributos: la distancia horizontal de la sombra, la distancia vertical de la sombra, el desenfoque (blur) y el color de la sombra. Además podemos usar valores negativos para las distancias horizontal y vertical para crear sombras en otros sentidos. Un ejemplo de sombra en color gris:

```
-moz-box-shadow:3px 3px 6px #888888;
-webkit-box-shadow:3px 3px 6px #888888;
box-shadow:3px 3px 6px #888888;
```

Con lo que obtendríamos un resultado similar a:



### 3.2.5. Fondos

---

CSS3 también ha introducido nuevas propiedades para definir el estilo de las imágenes de fondo:

- **background-origin: border-box | padding-box | content-box** - permite definir el origen de coordenadas sobre el que se va a colocar la imagen de fondo. Acepta tres posibles valores: “border-box” para que el fondo empiece desde el mismo borde del elemento, “padding-box” para que la imagen de fondo se coloque a partir del espaciado de padding, y por último “content-box” para que la imagen de fondo se coloque donde empieza el contenido del elemento, sin tener en cuenta el borde ni el padding.
- **background-clip: border-box | padding-box | content-box** - define el área sobre la que se extiende la imagen de fondo, puede tomar tres valores: “border-box” se extiende por toda el área dentro de la zona definida a partir del borde, “padding-box” se extiende a partir del espaciado de padding y “content-box” el fondo se extiende solo dentro del área de contenido.
- **background-size:** Permite indicar el tamaño de la imagen de fondo. Acepta diferentes atributos:
  - background-size: 200px; // especifica ancho y alto a la vez
  - background-size: 200px 100px; // 200px de ancho y 100px de alto
  - background-size: auto 200px; // ajustar la anchura automáticamente
  - background-size: 50% 25%; // También podemos indicar el tamaño con porcentajes
  - background-size: contain; // Escalar la imagen al tamaño máximo posible (conservando las proporciones originales) para que quepa dentro del área asignada.
  - background-size: cover; // Escalar la imagen para que cubra completamente el área asignada (conservando las proporciones originales).
- Capas con múltiples imágenes de fondo: Con la propiedad **background** ahora podemos indicar varias imágenes de fondo, simplemente separándolas con comas. Para cada propiedad background debemos definir cuatro valores: imagen de fondo, posición vertical, posición horizontal, modo de repetición (repeat, repeat-x, repeat-y, no-repeat). Ejemplo:

```
background: url(imagen1.png) 10px center no-repeat,  
            url(imagen2.png) 0 center repeat-x;
```

Dado que estas propiedades no son soportadas todavía en todos los navegadores, deberemos de definir las también añadiendo los prefijos “-webkit-” y “-moz-” para dar un mayor soporte.

### 3.2.6. Texto

---

Las nuevas propiedades de CSS3 para dar formato a textos son:

- **text-shadow:** Permite dar sombra a un texto. Sus propiedades son distancia horizontal, distancia vertical, desenfocado (*blur*) y color de la sombra. Por ejemplo:

```
text-shadow: 2px 2px 2px #9e9e9e;
filter: dropshadow(color=#9e9e9e, offx=2, offy=2);
```

Con lo que obtendríamos un resultado similar a:

## Text Shadow

- **word-wrap: break-word;** - Permite separar palabras muy largas dentro de un elemento de bloque. Por defecto toma el valor “normal”, por lo que las palabras largas se saldrían del borde del elemento. Con el valor “break-word” indicamos que las palabras pueden ser partidas para que quepan en el ancho de la caja, de la forma:



AlSerUnTextoMuyLargo  
DeberíaVerseSeparadoE  
nAlgúnMomento.

- **text-overflow: clip | ellipsis;** - Indica la forma de partir texto cuando excede el tamaño de su contenedor. Con “clip” el texto sobrante será cortado directamente aunque se quede una palabra por la mitad, mientras que “ellipsis” quitará la última palabra que no quepa y pondrá en su lugar unos puntos suspensivos. Esta propiedad de momento no funciona en Firefox.
- **font-face:** Permite utilizar tipografías diferentes a las estándar, que serán importadas desde un fichero indicado. De momento soporta los formatos: .eot, .ttf y .otf. Para importar una fuente hay que seguir la siguiente sintaxis:

```
@font-face{
  font-family:<nombre_fuente>;
  src: <source>;
  [font-weight:<weight>];
  [font-style:<style>];
}
```

Con “font-family” indicamos el nombre que le damos a la fuente, y “src” nos permite seleccionar el fichero a cargar. Los otros dos parámetros son opcionales y tendrán valor “normal” por defecto. Por ejemplo:

```
@font-face {
  font-family: 'LeagueGothic';
  src: url(LeagueGothic.otf);
}

// Ahora ya podemos usar esta fuente:
p {
  font-family: 'LeagueGothic';
}
```

### 3.2.7. Columnas

Se han añadido nuevas propiedades que nos permiten crear columnas directamente a partir de un texto, estas son:

- **column-count:** Define el número de columnas en el que se va a dividir el texto. El texto será dividido de la mejor forma posible para que ocupe todo el espacio.
- **column-width:** Define el ancho de la columna (en unidades CSS).
- **column-gap:** Define el espacio entre las columnas (en unidades CSS).
- **column-rule:** Mediante esta propiedad podemos añadir una línea separadora entre las columnas, si no especificamos esta propiedad no se añadirá ninguna línea. Debemos de indicarle tres valores: ancho de la línea (en unidades CSS), estilo de la línea (solid, dotted, double, etc.) y color de la línea.

Para dar un mayor soporte antepondremos los prefijos -webkit- y -moz-, de la forma:

```
-webkit-column-count: 3;
-webkit-column-rule: 1px solid silver;
-webkit-column-gap: 10px;
-moz-column-count: 3;
-moz-column-rule: 1px solid silver;
-moz-column-gap: 10px;
column-count: 3;
column-rule: 1px solid silver;
column-gap: 10px;
```

Con lo que obtendríamos un resultado similar a:

<p>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</p>	<p>nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu</p>	<p>fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
--	--	--

### 3.2.8. Modelo de caja básico

Se han añadido nuevas propiedades para la disposición de elementos dentro de una caja:

- **overflow: visible | hidden | scroll | auto;** - permite indicar que ocurrirá si el contenido excede el área de un elemento, acepta cuatro posibles valores:
  - *visible*: No se recorta el contenido, la parte que quede fuera será visible. Es el valor por defecto.
  - *hidden*: El contenido que sobresalga será ocultado y tampoco se mostrará la barra de scroll.
  - *scroll*: El contenido se recorta y el navegador muestra la barra de scroll para ver el

resto del contenido.

- **auto:** Si el contenido se recorta el navegador mostrará una barra para ver el resto del contenido.
- **overflow-x:** igual que overflow pero indicaremos solo la propiedad en horizontal.
- **overflow-y:** igual que overflow pero solo para vertical.
- **resize: none | horizontal | vertical | both;** - habilita la posibilidad de redimensionar “manualmente” una caja. Puede tomar los valores: none, horizontal (permitir redimensionar solo en horizontal), vertical (solo en vertical), o both (redimensionar ambas dimensiones). Se recomienda además añadir la propiedad “overflow: hidden” para ocultar los elementos al redimensionar. Por ejemplo:

```
resize:both;
overflow:auto;
```

### 3.2.9. Transiciones

Una de las propiedades más novedosas que incorpora CSS3 es la posibilidad de crear animaciones mediante transiciones y transformaciones. Se pueden aplicar transiciones a la mayoría de propiedades (posiciones, fondo, color, tamaño, etc.). Desafortunadamente, no todos los navegadores usan los nombres estándares, por lo que tendremos que añadir los prefijos “-webkit-”, “-moz-” y “-o-” para dar un mayor soporte. La buena noticia es que la sintaxis para los valores en todos ellos es consistente:

- **transition-property: propertyName;** - Indica la propiedad sobre la que se aplicará la transición. Se puede aplicar sobre casi todas las propiedades: background, color, height, width, border, etc. Además también podemos usar el valor “all” para que se aplique sobre todas las propiedades disponibles, por ejemplo:

```
-webkit-transition-property: all;
-moz-transition-property: all;
-o-transition-property: all;
transition-property: all;
```

- **transition-duration: duration;** - Indica el tiempo que debe durar la transición en segundos (0.5s) o en milisegundos (500ms):

```
-webkit-transition-duration: 1s;
-moz-transition-duration: 1s;
-o-transition-duration: 1s;
transition-duration: 1s;
```

- **transition-timing-function: timingFunction;** - Es la función de tiempo que seguirá la transición, indica los cambios de velocidad a lo largo de la animación. Puede tomar cinco valores diferentes: ease (valor por defecto), linear, ease-in, ease-out, ease-in-out y cubic-bezier(cp1x, cp1y, cp2x, cp2y). Por ejemplo:

```
-webkit-transition-timing-function: linear;
-moz-transition-timing-function: linear;
-o-transition-timing-function: linear;
```



```
transition-timing-function: linear;
```

- **transition-delay: delay;** - Permite establecer un retraso inicial antes de ejecutar la transición. El tiempo de retraso se debe de indicar en segundos (0.5s) o en milisegundos (500ms):

```
-webkit-transition-delay: 0.2s;
-moz-transition-delay: 0.2s;
-o-transition-delay: 0.2s;
transition-delay: 0.2s;
```

- **transition: propertyName duration timingFunction delay;** - También podemos indicar las cuatro propiedades explicadas en una sola línea:

```
-webkit-transition: all 1s linear 0.2s;
-moz-transition: all 1s linear 0.2s;
-o-transition: all 1s linear 0.2s;
transition: all 1s linear 0.2s;
```

En general, lo mejor es declarar la transición en la propiedad base, sin pseudo-clases. De esta forma conseguiremos que se ejecute en ambas direcciones, por ejemplo:

```
.btn1 {
  background: #9c3;
  -webkit-transition: background 0.3s ease;
  -moz-transition: background 0.3s ease;
  -o-transition: background 0.3s ease;
  transition: background 0.3s ease;
}
.btn1:hover {
  background: #690;
}
```

### 3.2.10. Transformaciones

La propiedad “**transform**” nos permite aplicar transformaciones 2D o 3D a un elemento. Por ejemplo nos permite rotar, escalar, mover, etc. el elemento indicado. Esta propiedad todavía no es soportada por todos los navegadores, por lo que tendremos que añadir los prefijos “-ms-”, “-webkit-”, “-moz-” y “-o-” para dar un mayor soporte. Algunas de las funciones de transformación que podemos utilizar son:

- **none:** Indica que no se tiene que aplicar ninguna transformación.
- **translate(x,y):** Define una traslación 2D.
- **translateX(x):** Traslación en la coordenada X.
- **translateY(y):** Traslación en la coordenada Y.
- **scale(x,y):** Define una transformación de escalado 2D, deberemos de indicar valores entre 0.1 y 2.
- **scaleX(x):** Escalado en la coordenada X, deberemos de indicar valores entre 0.1 y 2.
- **scaleY(y):** Escalado en la coordenada Y, deberemos de indicar valores entre 0.1 y 2.
- **rotate(angle):** Aplica una rotación, el ángulo debe ser indicado en grados (ejem: “30deg”).

- **skew(x-angle,y-angle):** Define una transformación 2D de sesgo (o torsión), indicada en grados (deg).
- **skewX(angle):** Define una transformación de sesgo sobre la coordenada X (indicada en grados).
- **skewY(angle):** Define una transformación de sesgo sobre la coordenada Y (indicada en grados).

Además también podemos indicar varias transformaciones en una misma línea, de la forma:

```
#myDIV {
  -moz-transform: scale(1.2) rotate(9deg) translate(5px, 2px) skew(5deg,
5deg);
  -webkit-transform: scale(1.2) rotate(9deg) translate(5px, 2px)
skew(5deg, 5deg);
  -o-transform: scale(1.2) rotate(9deg) translate(5px, 2px) skew(5deg,
5deg);
  -ms-transform: scale(1.2) rotate(9deg) translate(5px, 2px) skew(5deg,
5deg);
  transform: scale(1.2) rotate(9deg) translate(5px, 2px) skew(5deg,
5deg);
}
```

Hay muchos más tipos de transformaciones, aunque algunas de ellos no son funcionales todavía (sobre todo las funciones 3D), para más información consulta: “[http://www.w3schools.com/cssref/css3\\_pr\\_transform.asp](http://www.w3schools.com/cssref/css3_pr_transform.asp)”.

### 3.2.11. Más información

Existen algunas páginas Web que proporcionan “generadores de estilos CSS”, facilitando en ocasiones este tipo de tareas:

- <http://css3generator.com/>
- <http://www.colorzilla.com/gradient-editor/>

Además, para obtener mucha más información sobre CSS3 podemos consultar:

- <http://www.w3schools.com/css3/default.asp>
- <http://www.w3.org/TR/2001/WD-css3-roadmap-20010523/>

## 4. Ejercicios - HTML5 y CSS3

En esta sesión vamos a realizar algunos ejemplos para practicar las nuevas funcionalidades que hemos visto de HTML5 y CSS3. Descarga la plantilla para los ejercicios desde [sesion02-ejercicios.zip](#). Al descomprimirlo aparecerá un archivo por cada ejercicio.

### 4.1. Ejercicio 1 - Estructura

En este primer ejercicio vamos a hacer una pequeña web de ejemplo utilizando las nuevas etiquetas semánticas de HTML5. En primer lugar creamos una cabecera con el logo del curso (logo.jpg) y el texto "DADM". La imagen tendrá un alto de 50 píxeles y el texto será de color blanco con tamaño de 16 puntos.

A continuación incluiremos la barra de navegación. Dentro de esta sección definiremos tres capas (etiquetas DIV) de la clase "navElement" con los textos "Inicio", "Contenidos" y "Profesorado". Como estilo indicaremos que la barra de navegación ocupe el 100% del ancho, y que cada uno de sus elementos "navElement" tengan el color de fondo "#f0f0f0" y que cambien al color "#005682" cuando el ratón pase sobre ellos.

El cuerpo principal del documento estará dentro de una sección "ARTICLE", con un ancho del 100%.

Por último incluiremos un pie de página con el texto "Dept. Ciencia de la Computación e IA", a cuyo texto aplicaremos el color gris.

### 4.2. Ejercicio 2 - Canvas

Para practicar con el elemento Canvas vamos a dibujar unas sencillas figuras geométricas. En primer lugar, en el cuerpo del documento, tenemos que crear el canvas con identificador "myCanvas" y dimensiones de 360x240.

En el código JavaScript obtendremos la instancia del canvas a partir de su identificador "myCanvas" y adquiriremos su contexto 2D. Definiremos un estilo de relleno con color '#0000ff', un color de borde '#ff0000' y un grosor de línea de 4 píxeles.

Dibujamos un rectángulo relleno en las coordenadas (0, 0, 150, 50), y otro rectángulo usando solo el borde en las coordenadas (0, 60, 150, 50). Por último dibujaremos un triángulo usando la herramienta trazo (path). Iniciamos el trazo (beginPath), definimos el primer punto en (160, 0) y los siguientes puntos en (270, 0), (160, 110) y (160, 0). Por último indicamos que rellene la figura y que cierre el trazo.

### 4.3. Ejercicio 3 - Multimedia

Para probar los elementos audio y vídeo vamos a crear un pequeño ejemplo de cada uno.

Creemos un elemento audio con los controles activados, y le indicamos que cargue en primer lugar el archivo "media/audio.ogg", en caso de que no soporte este formato que lo intente con "media/audio.mp3", seguido del reproductor flash (usando "media/audio.mp3") y el enlace para descarga directa del archivo mp3.

Para el elemento vídeo activaremos el uso de los controles, de la precarga y el autobuffer. Le indicaremos que en primer lugar utilice el archivo "media/video.ogv", y que en caso de no soportarlo intente utilizar "media/video.mp4", seguido por el reproductor en flash (usando "media/video.mp4") y la descarga directa del archivo mp4.

#### 4.4. Ejercicio 4 - Geolocalización

---

En este ejercicio solo tenemos que añadir una línea, y es el comando necesario para obtener la posición actual, al cual le pasaremos como parámetro el nombre de la función "showPosition". Esta función será la encargada de mostrar nuestras coordenadas (utilizando la API de Google Maps).

#### 4.5. Ejercicio 5 - Almacenamiento Offline

---

Para practicar con la nueva funcionalidad de almacenamiento Offline vamos a hacer un pequeño ejemplo que guarde de forma automática una nota. Si abrimos la plantilla correspondiente solo tenemos que definir las funciones de cargar, guardar y borrar la nota. Para esto utilizaremos el almacenamiento en local (localStorage) y el identificador 'savedNote', y para el borrado eliminaremos todo el contenido directamente (clear). Además en la sección de estilo CSS indicaremos para el "contenedor-nota" que utilice la imagen de fondo "imgs/stickynote.jpg".

#### 4.6. Ejercicio 6 - CSS3

---

En este ejercicio vamos a probar algunas de las funcionalidades nuevas de CSS3. Para todos los ejemplos recuerda indicar los nombres de las propiedades usando también los nombres nativos de los navegadores con los prefijos -webkit-, -moz- y -o-.

En el primer ejemplo "borderRadius" vamos a indicar que el cuadro tenga bordes redondeados con un radio de 30 píxeles.

En el segundo ejemplo (borderShadow) crearemos una sombra para el borde, con los siguientes atributos: distancia horizontal de la sombra de 5px, distancia vertical de la sombra 5px, desenfoque de 6px y color de la sombra grisáceo (#888888).

En el tercer ejemplo (textShadow) vamos a crear una sombra para el texto de 2px para sus distancias horizontal y vertical, de 2 píxeles para el desenfoque y "#9e9e9e" como color de sombra.

En el cuarto ejemplo (multiColumn) vamos a probar la funcionalidad de columnas. Aquí solo tendremos que indicar que el número de columnas es de 2 y que el espacio entre las columnas es de 15px.

En el último ejemplo (boxTransition) vamos a crear un efecto de transición. Usando la propiedad "transition" (para establecer todos los valores en una sola línea), indicaremos que vamos a realizar la transición sobre "margin-left", con una duración de 3s, y usando la función de tiempo "ease-in-out".

## 5. Introducción a jQuery Mobile

En esta sección introduciremos el framework jQuery Mobile y veremos como podemos crear una aplicación web para móviles de forma muy rápida y sencilla.

### 5.1. ¿Qué es jQuery Mobile?

jQuery Mobile es un framework de interfaz gráfica especialmente diseñado para el desarrollo de aplicaciones web para móviles que pretende unificar el diseño de interfaces de usuario para la mayoría de dispositivos móviles del mercado. Como su propio nombre indica, jQuery Mobile tiene como base el más que sólido framework javascript jQuery. Además, es un framework muy ligero, algo totalmente necesario por las propias características de los dispositivos móviles y sus capacidades de conexión. Por otro lado, el diseño es fácilmente modificable.

jQuery Mobile está apoyado en estos momentos por empresas tan importantes como Mozilla Corporation, Palm, Blackberry, Nokia y Adobe entre otras, y es que estas empresas no quieren seguir perdiendo mercado que dispositivos como Android y iPhone con sus aplicaciones nativas les ha hecho perder en los últimos tiempos.

El objetivo principal de este framework es conseguir una misma sensación de navegación por parte del usuario final en la mayoría de los dispositivos móviles. Por otro lado, si pensamos en el desarrollador, jQuery Mobile pretende que éste se centre en las características del producto y no tanto en el dispositivo móvil al que va dirigido tomando como suya la frase *"write less, do more"* (escribe menos, consigue más).



Dispositivos funcionando con jQuery Mobile

### 5.1.1. Características de jQuery Mobile

---

Las principales características de jQuery Mobile son las siguientes:

- Basado en el núcleo de jQuery
- Compatible con la mayoría de los dispositivos y navegadores
- Tamaño reducido, alrededor de los 20k
- Uso de HTML5 y sus características para evitar tener que escribir scripts
- Mejora progresiva, introduciendo todas las nuevas características a la gran mayoría de dispositivos
- Accesibilidad, asegurando que las aplicaciones implementadas con jQuery Mobile funcionará correctamente en los lectores de pantalla (como VoiceOver en iOS)
- Se soportan la mayoría de los eventos de ratón y de contacto
- Sencilla modificación del diseño base

### 5.1.2. Soporte de jQuery Mobile

---

Siguiente con la idea de abarcar el mayor número de dispositivos posibles, desde jQuery Mobile se ha dividido en 3 grados el soporte que se da a los diferentes dispositivos con lo que tendremos el grado A, B y C. El grado A indica aquellos dispositivos que soportan todas las características de jQuery Mobile entre las que destaca la navegación entre páginas web mediante AJAX y las transiciones entre las mismas. El grado B representa aquellos dispositivos que no soportan la característica de navegación con AJAX. Y por último, el grado C que representa los dispositivos en los que únicamente se garantiza el

comportamiento básico de las aplicaciones web creadas con jQuery Mobile.

La siguiente tabla representa aquellos dispositivos y navegadores y el grado de funcionamiento que jQuery Mobile garantiza en éstos.

Grado A	Grado B	Grado C
Apple iOS 3.2-5.0 beta	Blackberry 5.0	Blackberry4.x
Android 2.1-2.3	Opera Mini (5.0-6.0)	El resto de dispositivos y navegadores
Android Honeycomb	Windows Phone 6.5	
Windows Phone 7	Nokia Symbian^3	
Blackberry 6.0		
Blackberry 7		
Blackberry Playbook		
Palm WebOS (1.4-2.0)		
Palm WebOS 3.0		
Firebox Mobile (Beta)		
Opera Mobile 11.0		
Kindle 3		
Chrome Desktop 11-13		
Firefox Desktop 3.6-4.0		
Internet Explorer 7-9		
Opera Desktop 10-11		

### 5.1.3. ¿Cómo funciona jQuery Mobile?

El funcionamiento de jQuery Mobile es muy sencillo, ya que el programador simplemente debe encargarse de crear páginas con formato HTML (con unas pequeñas modificaciones) y jQuery Mobile será el encargado de realizar una serie de transformaciones en los elementos del DOM de esas páginas para que la interfaz de usuario sea lo más atractiva posible para los clientes de dispositivos móviles.

Estas transformaciones se realizan cuando el navegador recibe el contenido del documento HTML solicitado y siempre antes de que sea mostrado al usuario.

## 5.2. Páginas en jQuery Mobile



### 5.2.1. Anatomía de una página

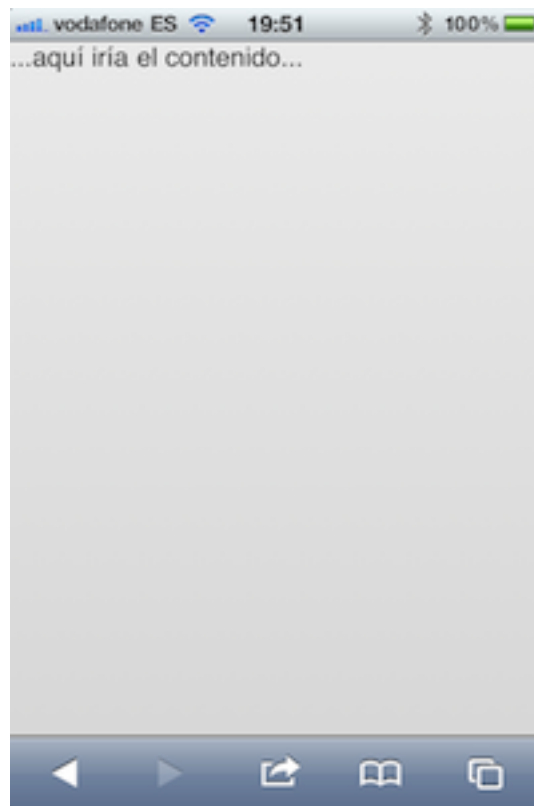
En jQuery Mobile tenemos dos posibilidades para organizar una aplicación web. Por un lado podemos tener páginas individuales y utilizar los típicos enlaces para navegar entre las diferentes páginas de la aplicación, pero por otro lado, jQuery Mobile también nos da la posibilidad de tener varias páginas web en un sólo documento html. De esta forma, nuestra página tardará un poco más en cargar pero la navegación por parte del usuario final será más amena.

Pero empecemos creando una página básica con jQuery Mobile analizando de esta forma la estructura. En primer lugar, debemos empezar especificando el doctype de HTML5. No te preocupes si tu dispositivo no soporta HTML5 porque directamente ignorará esta directiva. Lo siguiente que debemos hacer es referenciar a las librerías tanto de jQuery como de jQuery Mobile en la etiqueta head así como al archivo css de jQuery Mobile. Es aconsejable referenciar estos archivos en algún CDN para conseguir una mejor experiencia por parte del usuario, ya que si estos archivos ya han sido cacheados al acceder a otra aplicación implementada en jQuery Mobile, el acceso a nuestra aplicación será mucho más rápida. Con estos datos, nuestra página web podría ser así:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Título de la página</title>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width,
initial-scale=1"/>
    <link rel="stylesheet"
href="http://code.jquery.com/mobile/1.0rc1/jquery.mobile-1.0rc1.min.css"
/>
    <script type="text/javascript"
src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
    <script type="text/javascript"
src="http://code.jquery.com/mobile/1.0rc1/jquery.mobile-1.0rc1.min.js"></script>
  </head>
  <body>
    ...aquí iría el contenido...
  </body>
</html>
```

Quizás te llame la atención la utilización de la etiqueta `<meta name="viewport" content="width=device-width, initial-scale=1"/>`. Con esta etiqueta, muchos navegadores ajustan el nivel de zoom de la página al tamaño correcto.

Si cargásemos esta página en un móvil, notaríamos como jQuery Mobile se encarga automáticamente de hacer desaparecer la barra de la dirección url para aprovechar al máximo el espacio disponible en la pantalla del dispositivo con lo que veríamos la siguiente página.



Aplicación web con jQuery Mobile

Pero sigamos creando la base de nuestra aplicación web para móviles. En jQuery Mobile cada página de nuestra aplicación debe estar en un elemento HTML de tipo `div` al cual le tenemos que pasar un atributo llamado `data-role` indicándole que su valor será `page`.

```
<div data-role="page">
  ....
</div>
```

Dentro de esta capa se puede utilizar cualquier elemento HTML válido, sin embargo lo más habitual en nuestras aplicaciones con jQuery Mobile será que utilicemos otras capas con el atributo `data-role` con los valores `header`, `content` y `footer`.

```
<div data-role="page">
  <div data-role="header">Título</div>
  <div data-role="content">Contenido</div>
  <div data-role="footer">Pie de página</div>
</div>
```



Aplicación web con jQuery Mobile

Hasta el momento, en nuestro documento html únicamente hemos creado una página web, pero tal y como comentábamos anteriormente, podemos tener más de una página web en un único documento html para que la navegación entre las páginas de nuestra aplicación sea más eficiente. Si queremos hacer esto, en cada bloque `div` con el atributo `data-role` a `page` debemos añadir el atributo `id`, que deberá tener un valor único en cada documento html. Posteriormente, los enlaces entre las páginas se debe hacer especificando dicho identificador precedido del símbolo `#` de la siguiente forma `<a href="#acercade">`.

```
<body>

<!-- Inicio de la primera página -->
<div data-role="page" id="home">

    <div data-role="header">
        <h1>Home</h1>
    </div><!-- /header -->

    <div data-role="content">
        <p>
            Esta es mi primera aplicación con jQuery Mobile
            y ha sido creada por <a href="#author">Fran
García</a>
        </p>
    </div><!-- /content -->

    <div data-role="footer">
```

```

        <h4>Pie de página</h4>
      </div><!-- /footer -->
</div><!-- /page -->

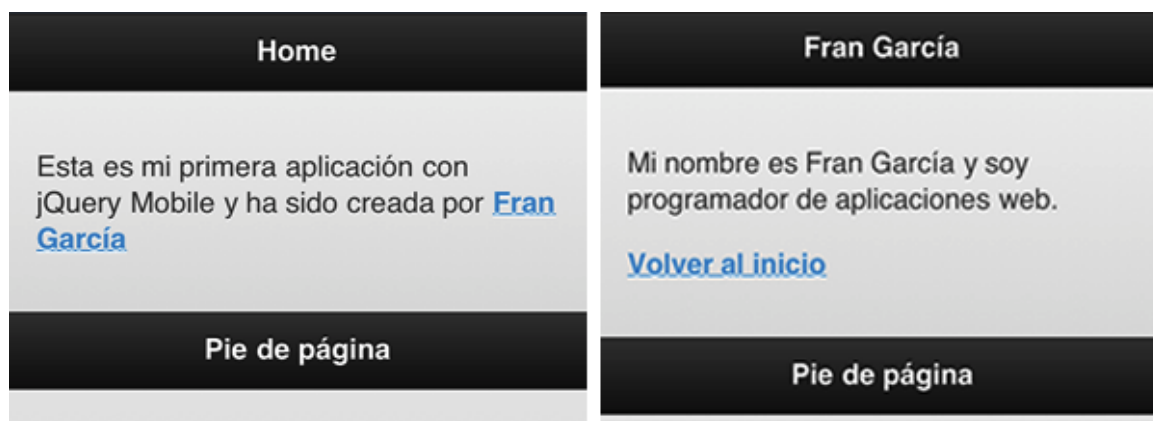
<!-- Start of second page -->
<div data-role="page" id="author">

  <div data-role="header">
    <h1>Fran García</h1>
  </div><!-- /header -->

  <div data-role="content">
    <p>Mi nombre es Fran García y soy programador de
aplicaciones web.</p>
    <p><a href="#home">Volver al inicio</a></p>
  </div><!-- /content -->

  <div data-role="footer">
    <h4>Pie de página</h4>
  </div><!-- /footer -->
</div><!-- /page -->
</body>

```



Varias páginas en un sólo documento HTML

Cuando en un documento html coexisten más de un elemento `div` con el atributo `data-role` a `page`, jQuery Mobile mostrará únicamente la primera página encontrada. Si ahora probamos nuestra aplicación con un dispositivo móvil, comprobaremos que si hacemos clic sobre el enlace del autor de la aplicación, ésta se muestra mediante una transición animada.

Hay que tener en cuenta que con jQuery Mobile no es posible enlazar directamente a una "página" con un enlace del tipo `<a href="index.html#foo">`. Esto es debido a que jQuery Mobile intentará buscar una "página" con un identificador `#foo` en lugar del comportamiento nativo de hacer scroll directo hasta el ancla en cuestión.

Comentar también que todos los elementos vistos hasta el momento son opcionales, sin embargo, puede ser una buena forma de estructurar nuestra aplicación web.

### 5.2.2. Títulos de las páginas

---

Cuando en una página desarrollada con jQuery Mobile cargamos una página, hacemos clic en un enlace o enviamos un formulario, jQuery Mobile utiliza Ajax para cargar el contenido de la página destino. El problema de esta forma de trabajar es que el título de la página siempre será el de la primera página cargada. Sin embargo, jQuery Mobile se encarga de recoger el título de la nueva página cargada y sustituirlo por la página que ha realizado la llamada Ajax.

Algo parecido sucede con los documentos con más de una página, en los cuales, el título de la misma se comparte en todas ellas. Sin embargo, jQuery Mobile nos permite cambiar este título añadiendo un nuevo atributo en la página en cuestión llamado `data-title`.

```
<div data-role="page" id="foo" data-title="Page Foo">
...
</div>
```

### 5.2.3. Enlaces entre páginas

---

jQuery Mobile está especialmente diseñado para funcionar de forma muy sencilla con una serie de convenciones a la hora de enlazar páginas y lo mejor es que esa convención no supone ningún cambio en la forma habitual de trabajar y será jQuery Mobile el encargado de gestionar esas llamadas a otras páginas (utilizando Ajax siempre que sea posible) para generar el comportamiento deseado.

En algunos casos no es posible realizar esta llamada Ajax como por ejemplo en aquellos enlaces a otros dominios o bien cuando se especifique directamente en los atributos del enlace (esto lo veremos más adelante). En estos casos, se realizará una petición http normal y corriente.

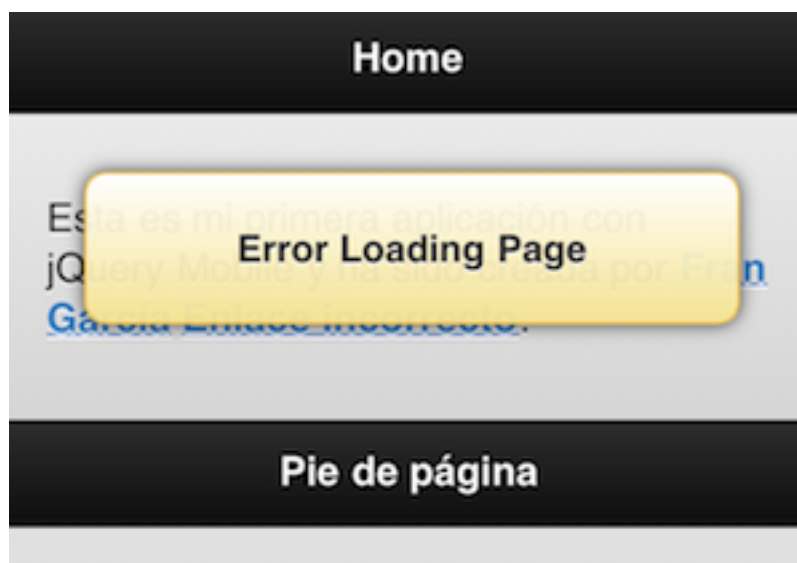
La idea de este modelo es permitir a los desarrolladores crear aplicaciones web con jQuery Mobile sin ninguna configuración especial pero que al mismo tiempo parezcan aplicaciones nativas.

#### 5.2.3.1. Comportamiento básico de los enlaces: Ajax

---

Para permitir las transiciones entre las páginas, todos los enlaces que apunten a una página externa, se cargarán vía Ajax y para que esto sea lo menos intrusivo posible, jQuery Mobile recoge la página destino, la procesa y la añade al DOM de la página actual. Mientras esto se está realizando, se mostrará la típica imagen que indica que el contenido de la página se está cargando en estos momentos. En caso de que la nueva página, se cargue correctamente, ésta se muestra mediante una transición.

En caso de que en la petición Ajax se produzca un error, jQuery Mobile mostrará un pequeño mensaje de error que desaparecerá a los pocos segundos.



Página de error

#### 5.2.3.2. Enlaces sin utilizar Ajax

En algunas ocasiones, jQuery Mobile no cargará los enlaces de nuestras aplicaciones utilizando Ajax. Estos casos son los siguientes:

- Enlaces que apunten a dominios externos
- Enlaces que tengan el atributo `rel="external"`
- Enlaces que tengan el atributo `data-ajax="false"`
- Enlaces que utilicen el atributo `target`

En estos casos, las páginas no se cargarán con Ajax y por lo tanto la página se recargará por completo y por supuesto, sin animación. Es necesario sin embargo diferenciar que el atributo `rel="external"` se debe utilizar cuando se enlaza a otro sitio o dominio, mientras que el atributo `data-ajax="false"` se utilizará en aquellos casos en que por cualquier motivo necesitemos cargar una página en nuestra aplicación sin utilizar Ajax.

#### 5.2.3.3. Enlaces en un documento multi-página

Como hemos comentado anteriormente, un único documento HTML puede contener una o varias páginas únicamente especificando diversos elemento de tipo `div` con el atributo `data-role="page"` y un valor diferente para cada página del atributo `id`. Esto hace que podamos tener una aplicación completa en un único documento HTML. Cuando un usuario acceda a nuestra aplicación, jQuery Mobile mostrará la primera página que encuentre.

Si un enlace de un documento multi-página apunta a un ancla, como por ejemplo `#foo`, jQuery Mobile buscará un elemento de tipo `div` con el atributo `id="foo"`. En caso de que lo encuentre, se mostrará el contenido de esa página mostrando anteriormente una

transición. Del mismo modo podemos tener también enlaces a otros documentos HTML y la única diferencia en este caso es que jQuery Mobile mostrará una imagen al usuario indicando que la página se está cargando. En ambos casos, la url se actualizará de tal forma que el usuario pueda utilizar el botón para volver atrás, enlazar directamente un determinado contenido o bien utilizar la opción de favoritos del navegador.

**Aviso:**

Es importante saber que si estamos enlazando desde una página que fue cargada vía AJAX a un documento HTML con varias páginas, es necesario añadir el atributo `rel="external"` o `data-ajax="false"`. Esto indicará a jQuery Mobile que la página debe ser cargada por completo. Si no hacemos esto, el usuario experimentará problemas a la hora de navegar por nuestra aplicación, especialmente si utiliza el botón para volver atrás.

```
<a href="multipagina.html" rel="external">Multi-página</a>
```

#### 5.2.3.4. Botón Atrás

En jQuery Mobile implementar la posibilidad de que el usuario pueda volver atrás en nuestra aplicación es muy sencillo y podemos conseguirlo simplemente especificando el atributo `data-rel="back"` en un enlace. jQuery Mobile se encargará de obviar el valor del atributo `href`.

Hay que tener en cuenta también que si únicamente se desea que se realice una transición que simule una vuelta atrás podemos utilizar el atributo `data-direction="reverse"`.

**Aviso:**

Hay que tener cuidado con el atributo `data-rel="back"` porque si nuestra aplicación no es secuencial y podemos saltar a diversas partes de la misma desde un mismo punto, esta característica que implementa el botón atrás automáticamente puede darnos problemas.

#### 5.2.4. Transiciones entre páginas

Actualmente, jQuery Mobile permite hasta seis tipos de transiciones en nuestras aplicaciones. Estas transiciones están implementadas mediante CSS y se pueden aplicar a cualquier objeto o página. Por defecto, siempre se aplica la transición que muestra el nuevo contenido de derecha a izquierda.

Para especificar una transición diferente, se puede añadir el atributo `data-transition` al enlace en cuestión. Los posibles valores de este atributo son los siguientes:

- *slide*, se mostrará el nuevo contenido con una transición de derecha a izquierda
- *slideup*, se mostrará el nuevo contenido con una transición de arriba a abajo
- *slidedown*, se mostrará el nuevo contenido con una transición de abajo a arriba
- *pop*, se mostrará el nuevo contenido con una transición que empieza desde el centro de la página

- *fade*, se mostrará el nuevo contenido con una transición que se mostrará desvaneciendo al contenido antiguo y mostrando poco a poco el nuevo
- *flip*, se mostrará el nuevo contenido con una transición que simula la apertura de una puerta. Mencionar que esta transición no funciona en la mayoría de las versiones de Android porque carece de transformaciones 3D con CSS.

Comentar por último que también es posible forzar una transición de "vuelta atrás" especificando el atributo `data-direction="reverse"`.

### 5.2.5. Diálogos

---

jQuery Mobile permite simular la creación de cuadros de diálogo en nuestras aplicaciones.

#### 5.2.5.1. Creando cuadros de diálogos

---

Cualquier página de nuestras aplicaciones con jQuery Mobile pueden ser mostradas como un cuadro de diálogo simplemente añadiendo el atributo `data-rel="dialog"` al enlace que muestra la página destino. Cuando se detecta este atributo, jQuery Mobile se encarga de poner unos bordes redondeados a la nueva página, crear automáticamente unos márgenes alrededor de la página y poner un fondo oscuro para que parezca que la nueva página está suspendida por encima de la página que lo carga.

```
<a href="foo.html" data-rel="dialog">
  Abrir cuadro de diálogo
</a>
```





Cuadro de diálogo

#### 5.2.5.2. Transiciones

Por defecto, el dialogo se abre con una transición del tipo *pop*, pero como en todas las páginas, se puede especificar cualquier otra transición simplemente añadiendo el atributo `data-transition` y especificar cualquiera de los valores que veíamos anteriormente. Para que simular lo más posible los efectos de los cuadros de diálogo, se aconseja utilizar las transiciones *pop*, *slideup* o *flip*.

```
<a href="foo.html" data-rel="dialog" data-transition="pop">
  Abrir cuadro de diálogo
</a>
```

#### 5.2.5.3. Cerrando cuadros de diálogos

Al hacer clic en cualquier enlace dentro de un cuadro de diálogo, jQuery Mobile se encargará automáticamente de cerrarlo y mostrar la transición correcta. Sin embargo, si queremos generar el típico botón de *Cerrar* en un cuadro de diálogo, simplemente debemos añadir el atributo `data-rel="back"`.

#### 5.2.5.4. Historial y botón Atrás

Habitualmente, los cuadros de diálogo se generan en el contexto de otras páginas. jQuery Mobile se encarga de no incluir estos cuadros de diálogo en el historial de navegación de nuestro navegador. Del mismo modo, cuando un usuario haga clic en el botón Atrás del navegador, no será posible llegar de nuevo a un cuadro de diálogo, con lo que no tenemos preocuparnos en absoluto por este tema.

### 5.2.6. Precarga y caché de páginas

---

A continuación vamos a ver algunas técnicas presentes en jQuery Mobile para mejorar la experiencia del usuario a la hora de navegar por nuestra aplicación.

#### 5.2.6.1. Precarga de páginas

---

Si en tu aplicación utilizas un documento html con una sola página web, pero sin embargo, prefieres cargar el contenido de determinadas páginas sin mostrar la típica imagen de "Cargando...", podemos hacer una precarga de estas páginas simplemente añadiendo el atributo `data-prefetch` a cualquier enlace que queremos precargar. jQuery Mobile se encargará de cargar el contenido de esta página enlazada sin que el usuario vea nada. Este podría ser un ejemplo:

```
<a href="enlaceprecargado.html" data-prefetch>Enlace precargado</a>
```

Así es como funciona internamente jQuery Mobile en estas precargas. Una vez la página principal se ha cargado por completo, jQuery Mobile buscará entre todos los enlaces aquellos que tengan el atributo `data-prefetch` para automáticamente cargar el contenido de esos enlaces. De esta forma, cuando el usuario haga clic en estos enlaces, la carga del contenido se hará mucho más rápida que se haría si no hubiéramos hecho esta precarga. Además, la imagen de "Cargando..." ya no volverá a aparecer salvo que por cualquier motivo, el contenido de la página enlazada todavía no se haya podido cargar en nuestra aplicación.

Es importante saber que estas precargas realizarán una serie de peticiones que en ocasiones nunca se utilizarán, con lo que debemos sólo utilizar esta precarga en determinadas situaciones y cuando sepamos con un alto grado de certeza que el usuario hará clic en ese enlace.

#### 5.2.6.2. Caché de páginas

---

Cuando se realizan las transiciones entre las diferentes páginas de nuestra aplicación, debemos tener en cuenta que ambas páginas deben estar cargadas en el DOM y que conforme vamos navegando por las mismas, más páginas se irán añadiendo a este DOM, lo que en versiones anteriores de jQuery Mobile provocaba en ocasiones que el rendimiento de la aplicación bajara o que incluso el navegador se cerrara inesperadamente.

Para solucionar este problema, jQuery Mobile se encarga de gestionar las páginas

almacenadas en el DOM y lo hace añadiendo un *flag* a estas páginas una vez ya hemos accedido a otra página del DOM. En caso de que volvamos a una página que ya haya sido eliminada del DOM previamente, el navegador intentará cargar la página de su propia caché o será de nuevo solicitada al servidor. Esto sucede únicamente con aquellas páginas que se cargan vía Ajax y no con los documentos html multi-páginas.

Sin embargo, jQuery Mobile también especifica una forma de gestionar nosotros mismos la caché de determinadas páginas de nuestra aplicación que consideremos interesantes. Esta gestión de la caché del DOM, supone que nosotros somos quienes deberemos encargarnos de controlar que el tamaño del mismo no exceda unos determinados límites.

Podemos hacerlo de dos formas. Por un lado aplicando esta gestión de la caché a toda nuestra aplicación.

```
$.mobile.page.prototype.options.domCache = true;
```

O bien especificarlo únicamente en determinadas páginas mediante el atributo `data-dom-cache="true"`.

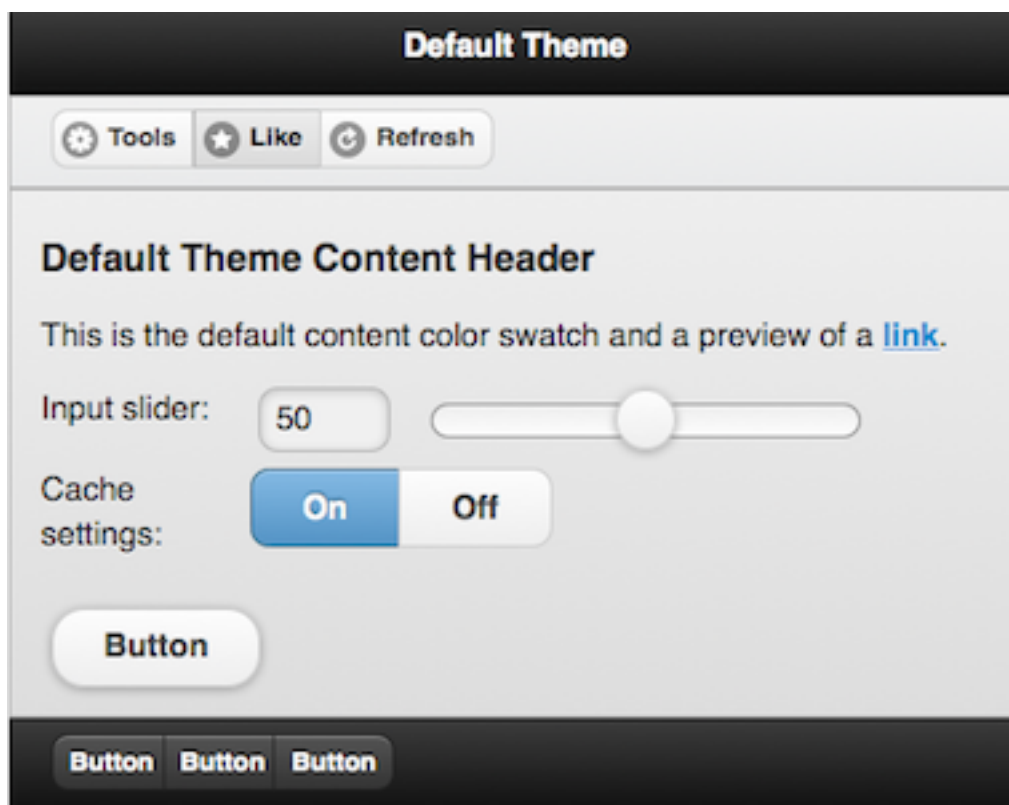
```
<a href="foo/bar/baz" data-dom-cache="true">link text</a>
```

### 5.2.7. Estilo de los componentes jQuery Mobile

---

jQuery Mobile dispone de un sistema muy robusto y sencillo para estilizar nuestra aplicación de varias formas. Cada componente de jQuery Mobile tiene la posibilidad de ser estilizado de una forma diferente. Para aplicar estos estilos, estos componentes pueden añadir el atributo `data-theme` y puede tomar los valores, *a*, *b*, *c*, *d* o *e* para elegir cualquiera de los cinco temas de los que dispone actualmente jQuery Mobile. Cada uno de estos temas, utiliza una serie de combinaciones de colores y formas diferentes. Por defecto, jQuery Mobile utiliza una combinación de estos temas tal y como vemos en las siguientes imágenes.

#### Tema por defecto



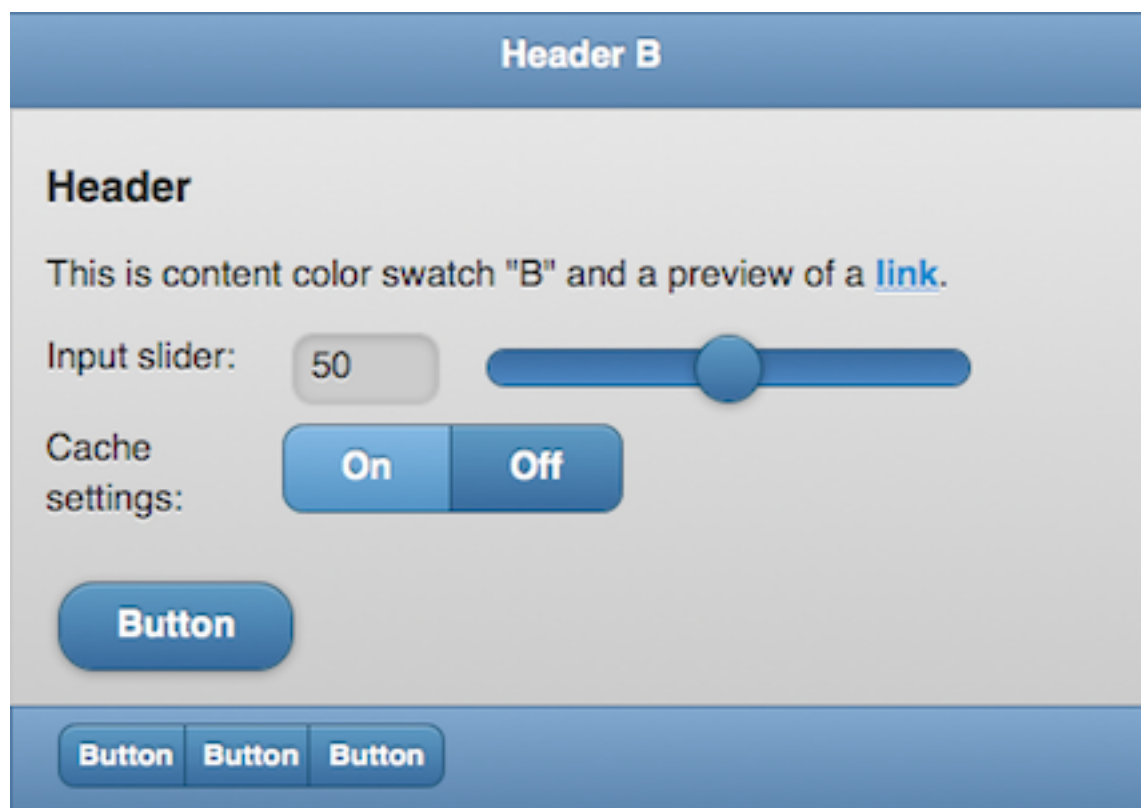
Tema por defecto

**Tema A**



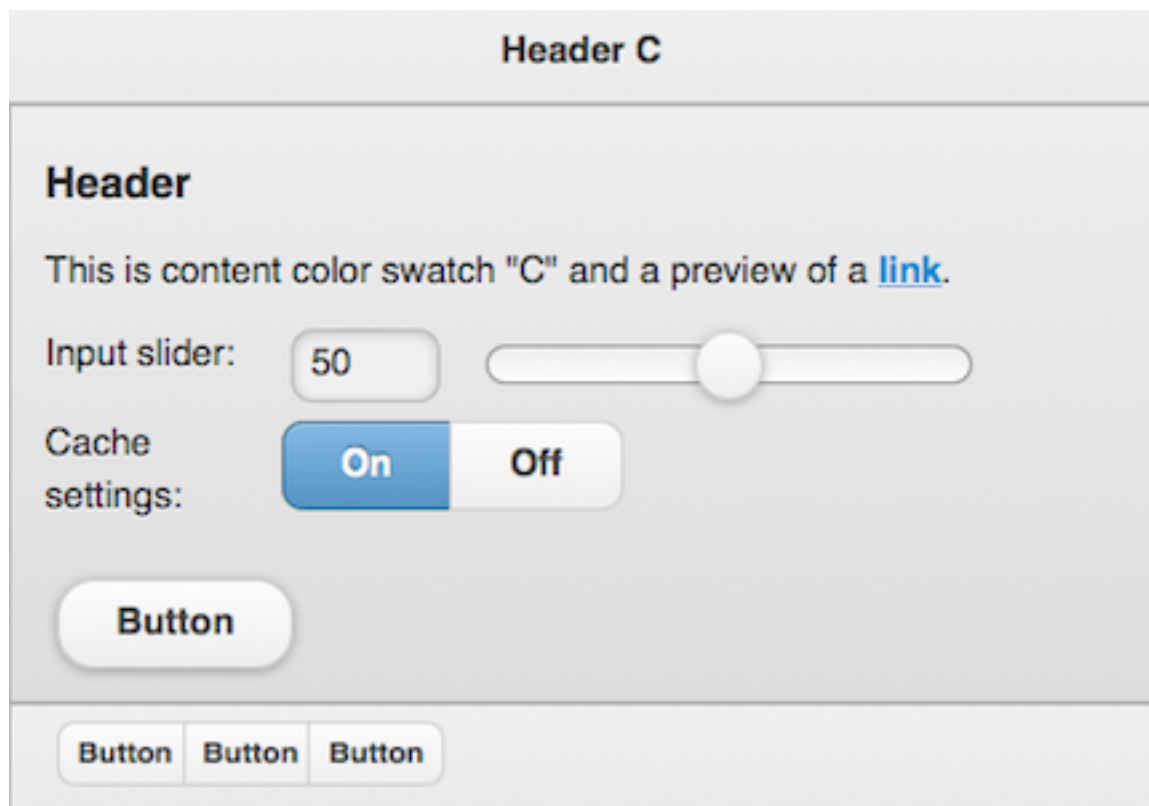
Tema A

## Tema B



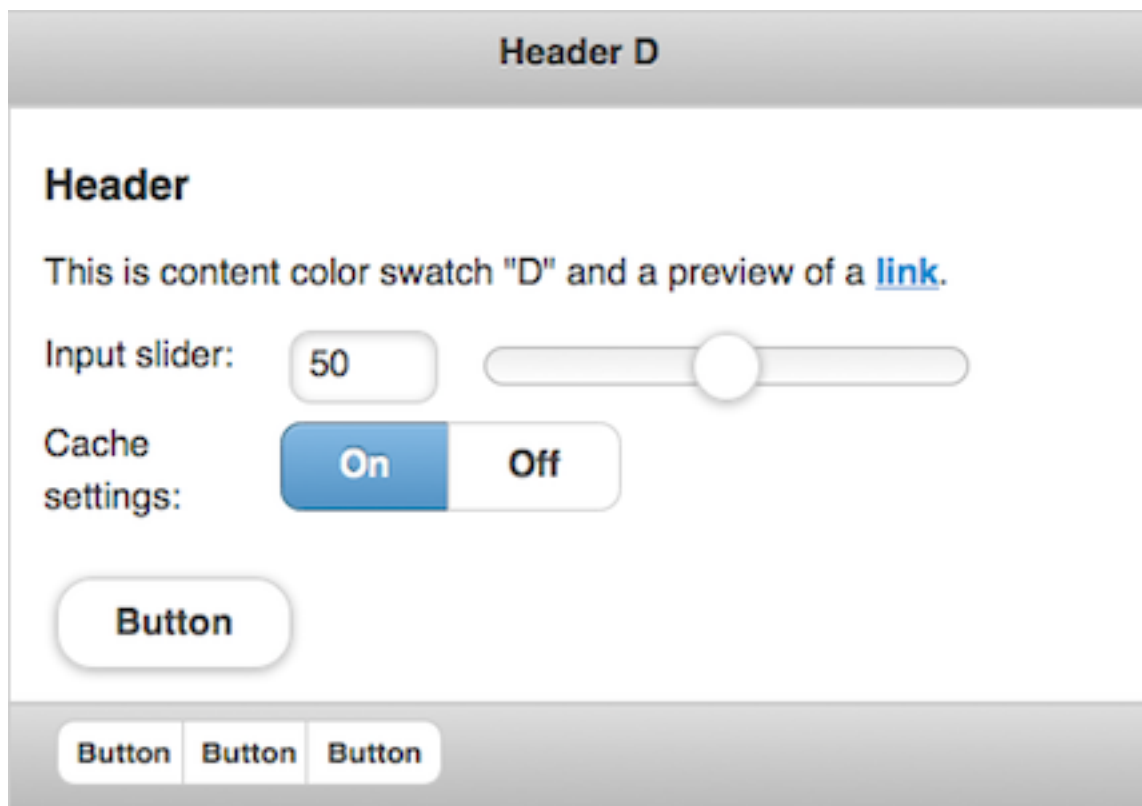
Tema B

Tema C



Tema C

## Tema D



Tema D

## Tema E





Tema E

### 5.3. Barras de herramientas

Las barras de herramientas son utilizadas habitualmente en las cabeceras y los pies de nuestras aplicaciones en cualquier aplicación web móvil. Por este motivo, jQuery Mobile nos ofrece una serie de componentes ya preparados para ser utilizados en nuestras aplicaciones.

#### 5.3.1. Tipos de barras de herramientas

En una aplicación jQuery Mobile estándar, veremos dos tipos de barras de herramientas: las cabeceras (*headers*) y los pies de página (*footers*):

- La **barra de herramientas de la cabecera** se utiliza para indicar el título de la página actual, casi siempre es lo primero que se muestra en la aplicación y es aconsejable que como mucho tenga dos botones, uno a la izquierda del título y otro a la derecha.
- La **barra de herramientas del pie de página** es habitualmente el último elemento en cada página y los desarrolladores la pueden utilizar de diversas formas, aunque lo normal es que haya una combinación entre texto y botones.

También es muy habitual que las aplicaciones realizadas con jQuery Mobile utilicen una

navegación horizontal que suele estar incluida en la cabecera y/o en el pie de página. Para ello, jQuery Mobile introduce una barra de navegación predeterminada que convierte una lista desordenada de enlaces en una barra horizontal del siguiente estilo:



Barra de navegación

Para facilitarnos aún más el trabajo con estas barras de herramientas, jQuery Mobile facilita incluso el posicionamiento de éstas en una aplicación. Por defecto, estas barras se colocan una detrás de la otra del mismo modo en el que se definen en nuestro documento html en lo que se conoce como posición *inline*.

Sin embargo, en ocasiones deseamos que una determinada barra de herramientas esté siempre visible en nuestra aplicación para facilitar su uso por parte del usuario en lo que se conoce como posición *fixed*. La barra de herramientas aparecerá en la misma posición como si hubiera sido definida de tipo *inline*, pero en cuanto el usuario haga *scroll* en la aplicación, la barra de herramientas se desplazará para ocupar una posición visible en la aplicación.

Incluso estas barras de herramientas desaparecerán y aparecerán de nuestra aplicación con cada contacto que el usuario haga en el dispositivo móvil. Para indicar que una barra de herramientas debe tener este tipo de posición podemos poner el atributo `data-position="fixed"`.

Por otro lado, jQuery Mobile también dispone del modo a *pantalla completa* para las barras de herramientas. Básicamente funcionan de la misma forma que la posición *fixed*, salvo que ésta ya no se muestra al inicio o al final de la página y sólo se muestra cuando el usuario hace clic sobre la página. Este tipo de posicionamiento es muy útil en aplicaciones donde se muestren fotos o vídeos, en los cuales queremos cargar el contenido a pantalla completa y esta barra de herramientas únicamente se mostrará cuando el usuario toque la pantalla.

Para habilitar esta característica en nuestras aplicaciones debemos especificar el atributo `data-fullscreen="true"` al elemento `div` que contiene el atributo `data-role="page"` y además, debemos indicar también el atributo `data-position="fixed"` en la cabecera y en el pie de la página.

### 5.3.2. Cabeceras

Como comentábamos anteriormente, la cabecera se sitúa al inicio de las páginas y habitualmente contiene un título y opcionalmente puede tener hasta dos botones a los lados del título. Para el título de la cabecera habitualmente se utiliza el encabezado `<h1>`, aunque también posible utilizar cualquiera de los otros encabezados (*h1-h6*). Por ejemplo, un documento html multi-página puede tener un título de tipo *h1* en la primera página y un título *h2* en la segunda, sin embargo, por defecto, jQuery Mobile los mostrará con el

mismo estilo por motivos de consistencia visual. Comentar también que por defecto, las cabeceras utilizan el estilo *a* (fondo negro y letras en blanco).

### 5.3.2.1. Botones

En la configuración estándar de las cabeceras se ha dejado un espacio para añadir hasta dos botones a ambos lados del título y habitualmente se utilizan enlaces como botones. jQuery Mobile localiza el primero de estos enlaces y lo coloca automáticamente a la izquierda del título y el segundo enlace (en caso de que lo haya), a la derecha del título.

```
<div data-role="header" data-position="inline">
  <a href="index.html" data-icon="delete">Cancel</a>
  <h1>Edit Contact</h1>
  <a href="index.html" data-icon="check">Save</a>
</div>
```



Cabecera con botones

Los botones automáticamente adoptan el mismo estilo que la barra que los contiene, pero tal y como hemos visto antes, podemos modificar este diseño para mostrar otro diseño a los usuarios.

```
<div data-role="header" data-position="inline">
  <a href="index.html" data-icon="delete">Cancel</a>
  <h1>Edit Contact</h1>
  <a href="index.html" data-icon="check" data-theme="b">Save</a>
</div>
```



Cabecera con botones

Pero, ¿qué pasa si queremos controlar la posición donde se pinta el botón? Pues que la gente de jQuery Mobile ya ha pensado en esto y simplemente añadiendo el atributo `class` con los valores `ui-btn-left` o `ui-btn-right` al enlace en cuestión podremos indicar donde queremos que aparezca el botón.

```
<div data-role="header" data-position="inline">
  <h1>Page Title</h1>
  <a href="index.html" data-icon="gear"
  class="ui-btn-right">Options</a>
</div>
```



Controlando la posición de los botones

Además de estos botones, jQuery Mobile también pone a disposición de los desarrolladores la posibilidad de que automáticamente se cree un botón para volver atrás en las páginas de nuestras aplicaciones. Sin embargo, por defecto esta opción está desactivada. Para activarla únicamente debemos especificar en aquellas páginas donde queramos insertar automáticamente este botón el atributo `data-add-back-btn="true"`

Esto colocará un botón a la izquierda del título que permitirá al usuario volver atrás. Sin embargo, este texto aparecerá en inglés, cosa que no siempre será lo deseado. Vamos a poder modificar el texto que aparece indicando el atributo `data-back-btn-text="Atrás"` en la página en cuestión. De igual forma también podremos modificar el tema por defecto de este botón Atrás con el atributo `data-back-btn-theme="e"`.

Pero además de esta forma, también podemos crear nosotros mismos nuestros botones para volver atrás simplemente especificando el atributo `data-rel="back"` a un enlace en cuestión. Además, recuerda que también tenemos la posibilidad de mostrar una transición inversa con el atributo `data-direction="reverse"`.

### 5.3.3. Pies de página

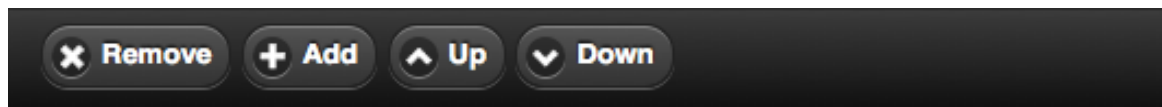
Ahora que ya conocemos como podemos modificar el comportamiento de las cabeceras con jQuery Mobile, pasemos a ver en profundidad los pies de página, que básicamente tienen la misma estructura que las cabeceras con la salvedad, claro está, del atributo `data-role="footer"`.

```
<div data-role="footer">
  <h4>Pie de página</h4>
</div>
```

Estructuralmente, los pies de páginas son muy parecidos a las cabeceras, con la salvedad que en los pies de página, jQuery Mobile no añade automáticamente esos botones que veíamos anteriormente a ambos lados del título, con lo que si queremos mostrar botones, los vamos a tener que pintar nosotros mismos.

Cualquier enlace válido añadido en el pie de página podemos convertirlo automáticamente en un botón en nuestra aplicación. Para ello debemos utilizar el atributo `data-role="button"`. Por defecto, jQuery Mobile no añade ningún tipo de espaciado entre los botones y los laterales del navegador, con lo que si queremos que no aparezcan demasiado pegados a esos laterales, podemos utilizar el atributo `class="ui-bar"`.

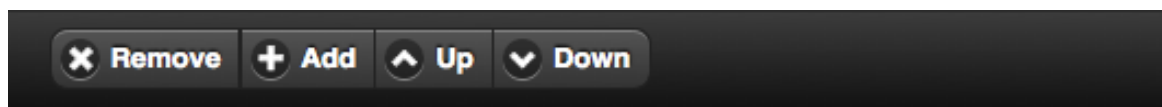
```
<div data-role="footer" class="ui-bar">
  <a href="index.html" data-role="button"
data-icon="delete">Remove</a>
  <a href="index.html" data-role="button" data-icon="plus">Add</a>
  <a href="index.html" data-role="button" data-icon="arrow-u">Up</a>
  <a href="index.html" data-role="button"
data-icon="arrow-d">Down</a>
</div>
```



Pie de página

Incluso podemos agrupar los botones con los atributos `data-role="controlgroup"` y `data-type="horizontal"`.

```
<div data-role="footer" class="ui-bar">
  <div data-role="controlgroup" data-type="horizontal">
    <a href="index.html" data-role="button"
data-icon="delete">Remove</a>
    <a href="index.html" data-role="button"
data-icon="plus">Add</a>
    <a href="index.html" data-role="button"
data-icon="arrow-u">Up</a>
    <a href="index.html" data-role="button"
data-icon="arrow-d">Down</a>
  </div>
</div>
```



Pie de página

### 5.3.4. Barras de navegación

jQuery Mobile tiene también una característica que permite añadir barras de navegación en nuestras aplicaciones. Estas barras permiten hasta cinco botones con la posibilidad incluso de añadir iconos y normalmente se sitúan dentro de la cabecera o del pie de página.

Una barra de navegación no es más que una lista desordenada de enlaces que se encuentra dentro de elemento con el atributo `data-role="navbar"`. Si queremos marcar una determinada opción de esta barra de navegación como activa podemos especificar el atributo `class="ui-btn-active"` en el enlace.

```
<div data-role="footer">
  <div data-role="navbar">
    <ul>
      <li><a href="a.html"
class="ui-btn-active">One</a></li>
      <li><a href="b.html">Two</a></li>
    </ul>
  </div><!-- /navbar -->
</div><!-- /footer -->
```



Pie de página

A medida que vayamos aumentando el número de botones en la barra de navegación,

jQuery Mobile se encargará automáticamente de posicionarlos correctamente en el navegador, tal y como vemos en las siguientes imágenes.



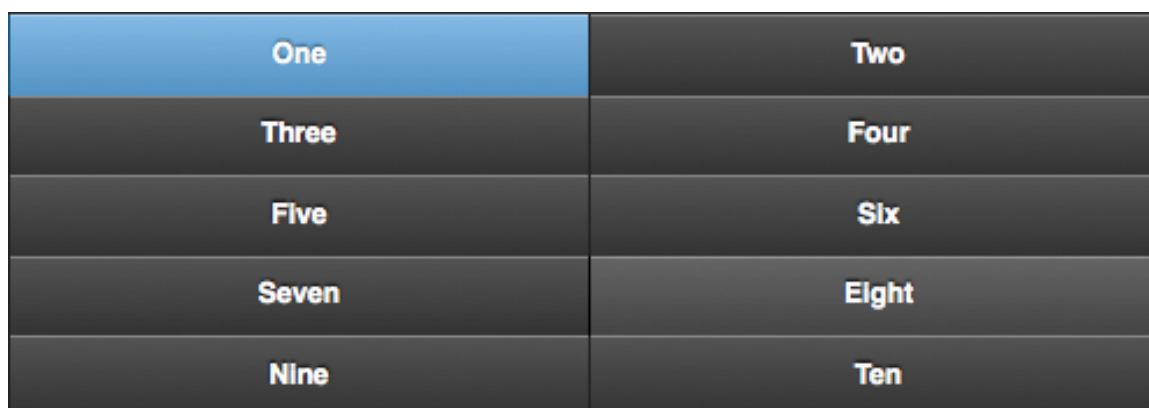
Pie de página con 3 botones



Pie de página con 4 botones



Pie de página con 5 botones



Pie de página con muchos botones

Comentar también que los botones de nuestras barras de navegación pueden ir acompañados de iconos que mejoren la experiencia del usuario final. Estos iconos se pueden añadir a los enlaces mediante el atributo `data-icon` especificándole un valor de entre los siguientes:

- arrow-l
- arrow-r
- arrow-u
- arrow-d
- delete
- plus
- minus
- check
- gear
- refresh
- forward

- back
- grid
- star
- alert
- info
- home
- search

Por último, comentar también que estos iconos aparecen por defecto a la izquierda del texto, pero si queremos colocarlos en cualquier otro lugar podemos utilizar el atributo `data-iconpos` con cualquiera de estos valores:

- left
- right
- top
- bottom

## 5.4. Formateo de contenido

El contenido de las páginas de una aplicación desarrollada con jQuery Mobile es totalmente abierto, aunque como siempre, siguiendo una serie de convenciones, el framework nos ayudará muchísimo en nuestra tarea. En esta sección veremos algunos aspectos interesantes como son los paneles desplegados y los diseños para múltiples columnas que nos facilitarán el formateo de contenido para aplicaciones móviles.

### 5.4.1. HTML básico

Lo más importante de jQuery Mobile es que para formatear contenido no tenemos más que aplicar los conocimientos del lenguaje HTML. Por su parte, el framework se encargará de modificar la apariencia de nuestras aplicaciones.

Por defecto, jQuery Mobile utiliza los estilos y tamaños estándar de HTML, tal y como se muestran en los siguientes ejemplos:

```
<h1>Cabecera H1</h1>
<h2>Cabecera H2</h2>
<h3>Cabecera H3</h3>
<h4>Cabecera H4</h4>
<h5>Cabecera H5</h5>
<h6>Cabecera H6</h6>
```

# Cabecera H1

## Cabecera H2

### Cabecera H3

#### Cabecera H4

##### Cabecera H5

###### Cabecera H6

Cabeceras

```

<ol>
  <li>Lista desordenada item 1</li>
  <li>Lista desordenada item 2</li>
  <li>Lista desordenada item 3</li>
</ol>

```

1. Lista desordenada item 1
2. Lista desordenada item 2
3. Lista desordenada item 3

Listas desordenadas

```

<table>
  <thead>
    <tr>
      <th>ISBN</th>
      <th>Título</th>
      <th>Autor</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>843992688X</td>
      <td>La colmena</td>
      <td>Camilo José Cela Trulock</td>
    </tr>
    <tr>
      <td>0936388110</td>
      <td>La galatea</td>
      <td>Miguel de Cervantes Saavedra</td>
    </tr>
  </tbody>
</table>

```



```

        <tr>
            <td>8437624045</td>
            <td>La dragontea</td>
            <td>Félix Lope de Vega y Carpio</td>
        </tr>
    </tbody>
</table>

```

ISBN	Título	Autor
843992688X	La colmena	Camilo José Cela Trulock
0936388110	La galatea	Miguel de Cervantes Saavedra
8437624045	La dragontea	Félix Lope de Vega y Carpio

Tablas

Como puedes observar, la apariencia de la mayoría de los componentes HTML no difieren prácticamente en nada de lo que ya conoces del desarrollo de aplicaciones web de *escritorio*.

Sin embargo, a continuación vamos a ver algunos componentes HTML que enmarcados dentro de una web desarrollada con jQuery Mobile facilitará la labor de estructuración de la información en una aplicación web para móviles.

#### 5.4.2. Diseños por columnas

Aunque el diseño por columnas no es muy habitual verlo en aplicaciones web para móviles debido a la propia idiosincrasia de los dispositivos móviles y su amplitud, en algunas ocasiones este tipo de diseños se hace necesario para aprovechar al máximo esta amplitud.

jQuery Mobile tiene como convención para este tipo de diseños una clase llamada *ui-grid* y que básicamente utiliza una serie de diseños CSS para generar un diseño por columnas, en el que se permiten hasta un máximo de 5 columnas.

En función del número de columnas que necesitemos en nuestros diseños, la capa contenedora debe tener el atributo `class` a uno de estos valores:

1. 2 columnas: `ui-grid-a`
2. 3 columnas: `ui-grid-b`
3. 4 columnas: `ui-grid-c`
4. 5 columnas: `ui-grid-d`

Veamos el siguiente código y analicémoslo:

```

<div class="ui-grid-a">
    <div class="ui-block-a"><div class="ui-bar ui-bar-a">Block
A</div></div>
    <div class="ui-block-b"><div class="ui-bar ui-bar-b">Block
B</div></div>
</div>

```

```

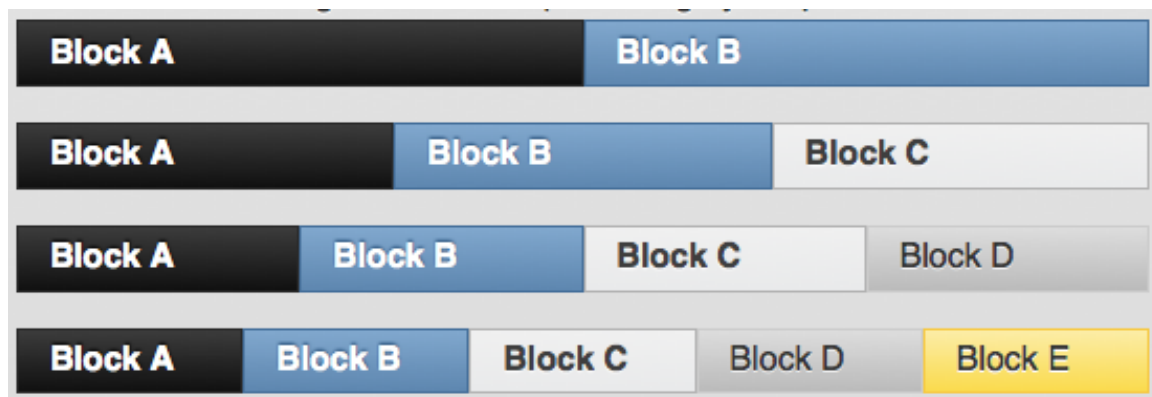
<br/>
<div class="ui-grid-b">
    <div class="ui-block-a"><div class="ui-bar ui-bar-a">Block
A</div></div>
    <div class="ui-block-b"><div class="ui-bar ui-bar-b">Block
B</div></div>
    <div class="ui-block-c"><div class="ui-bar ui-bar-c">Block
C</div></div>
</div>
<br/>

<div class="ui-grid-c">
    <div class="ui-block-a"><div class="ui-bar ui-bar-a">Block
A</div></div>
    <div class="ui-block-b"><div class="ui-bar ui-bar-b">Block
B</div></div>
    <div class="ui-block-c"><div class="ui-bar ui-bar-c">Block
C</div></div>
    <div class="ui-block-d"><div class="ui-bar ui-bar-d">Block
D</div></div>
</div>
<br/>

<div class="ui-grid-d">
    <div class="ui-block-a"><div class="ui-bar ui-bar-a">Block
A</div></div>
    <div class="ui-block-b"><div class="ui-bar ui-bar-b">Block
B</div></div>
    <div class="ui-block-c"><div class="ui-bar ui-bar-c">Block
C</div></div>
    <div class="ui-block-d"><div class="ui-bar ui-bar-d">Block
D</div></div>
    <div class="ui-block-e"><div class="ui-bar ui-bar-e">Block
E</div></div>
</div>

```

Este código HTML produciría la siguiente salida:



Diseño por columnas

Como vemos en la imagen, cada bloque está identificado por el atributo `class="ui-grid"` diferente en función del número de columnas. Posteriormente, cada elemento debe estar a su vez contenido por una capa con el atributo `class="ui-block"` que indicará la posición del contenido en el diseño por columnas. Por último, tener en cuenta que el diseño del contenido mostrado, dependerá del atributo `class="ui-bar"`

especificado, tal y como se muestra en la imagen.

Por último comentar también que la idea de los diseños por columnas es organizar en filas una serie de elementos. En ocasiones, nos puede interesar agrupar en un diseño de tres columnas a nueve elementos, con lo que jQuery Mobile metería tres elementos por cada fila tal y como vemos en el siguiente ejemplo.

```
<div class="ui-grid-c">
  <div class="ui-block-a"><div class="ui-bar ui-bar-e">Block
1</div></div>
  <div class="ui-block-b"><div class="ui-bar ui-bar-e">Block
2</div></div>
  <div class="ui-block-c"><div class="ui-bar ui-bar-e">Block
3</div></div>
  <div class="ui-block-a"><div class="ui-bar ui-bar-e">Block
4</div></div>
  <div class="ui-block-b"><div class="ui-bar ui-bar-e">Block
5</div></div>
  <div class="ui-block-c"><div class="ui-bar ui-bar-e">Block
6</div></div>
  <div class="ui-block-a"><div class="ui-bar ui-bar-e">Block
7</div></div>
  <div class="ui-block-b"><div class="ui-bar ui-bar-e">Block
8</div></div>
  <div class="ui-block-c"><div class="ui-bar ui-bar-e">Block
9</div></div>
</div>
```

Block 1	Block 2	Block 3
Block 4	Block 5	Block 6
Block 7	Block 8	Block 9

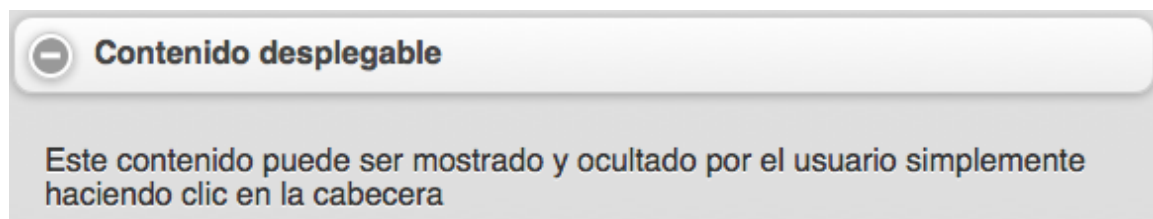
Multiples columnas y filas

### 5.4.3. Contenido desplegable

jQuery Mobile permite incluso agrupar contenido que se mostrará en forma de desplegable cuando el usuario haga clic sobre el mismo. El formato de este tipo de contenidos es muy sencillo y simplemente debemos utilizar el atributo `data-role="collapsible"` seguido de un elemento de encabezado, como por ejemplo `<h3/>` y el contenido a mostrar. jQuery Mobile se encargará incluso de pintar un botón para que el usuario detecte que puede hacer clic sobre éste, tal y como vemos en el siguiente ejemplo.

```
<div data-role="collapsible">
  <h3>Contenido desplegable</h3>
  <p>
    Este contenido puede ser mostrado y
    ocultado por el usuario simplemente haciendo
    clic en la cabecera
  </p>
```

```
</div>
```



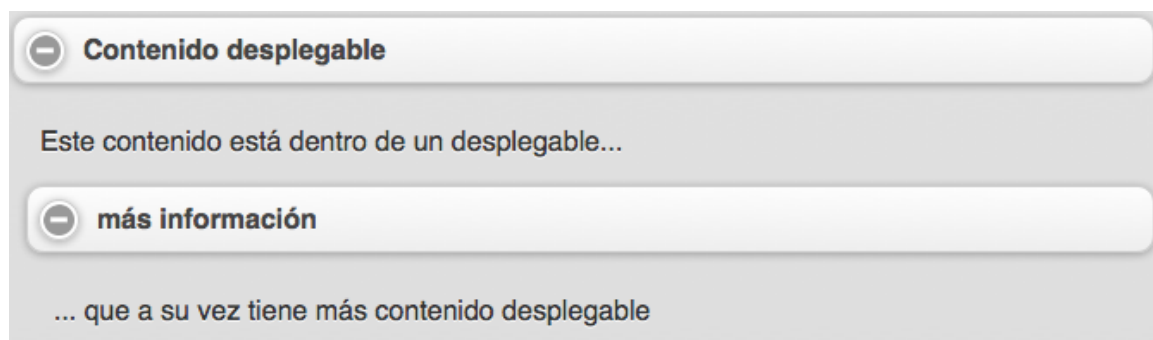
Contenido desplegable

Como puedes observar, este contenido aparece desplegado al cargar la página. Si quieres que el mismo aparezca contraído debes añadir el atributo `data-collapsed="true"` de la siguiente forma:

```
<div data-role="collapsible" data-collapsed="true"></div>
```

Comentar por último que el contenido de estos desplegables puede ir desde un simple párrafo hasta incluso un formulario y que incluso vamos a poder insertar un contenido desplegable dentro de otro, tal y como vemos en este ejemplo.

```
<div data-role="collapsible">
  <h3>Contenido desplegable</h3>
  <p>Este contenido está dentro de un desplegable...</p>
  <div data-role="collapsible">
    <h4>más información</h4>
    <p>... que a su vez tiene más contenido desplegable</p>
  </div>
</div>
```



Contenido desplegable anidado

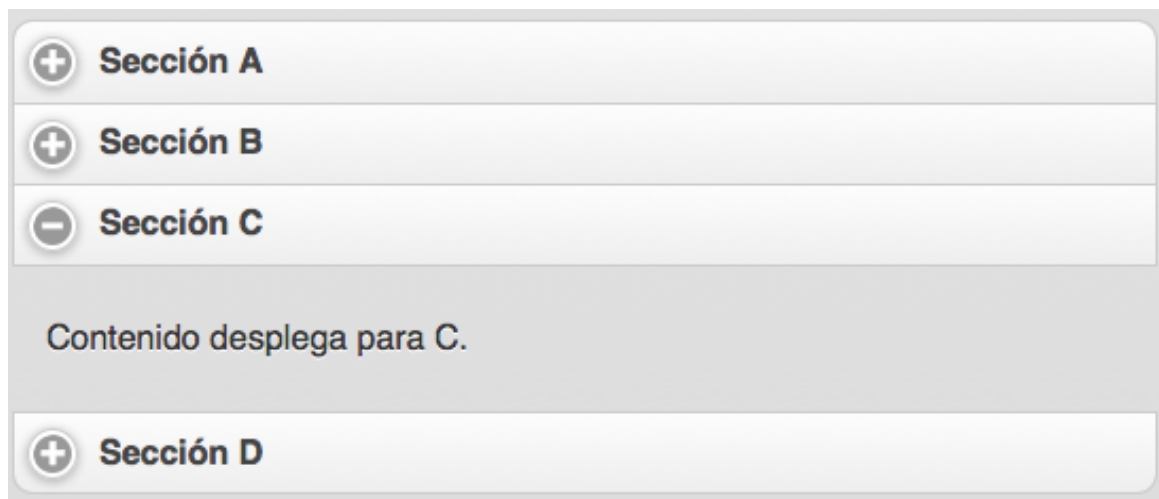
#### 5.4.4. Contenidos desplegables agrupados (acordeones)

Los acordeones no son más que un conjunto de contenidos desplegables de tal forma que al hacer clic sobre uno de ellos, el resto se ocultarán automáticamente, con lo que nunca podrá haber más de uno elemento desplegado al mismo tiempo.

La sintaxis de estos acordeones es prácticamente la misma que veíamos anteriormente salvo que ahora debemos añadir un elemento que agrupará a todos estos contenidos

desplegables y al cual le debemos asignar el atributo `data-role="collapsible-set"`. Si queremos mostrar de inicio alguno de estos desplegables, debemos asignar el atributo `data-collapsed="false"`.

```
<div data-role="collapsible-set">
  <div data-role="collapsible" data-collapsed="true">
    <h3>Sección A</h3>
    <p>Contenido desplega para A.</p>
  </div>
  <div data-role="collapsible" data-collapsed="true">
    <h3>Sección B</h3>
    <p>Contenido desplega para B.</p>
  </div>
  <div data-role="collapsible" data-collapsed="false">
    <h3>Sección C</h3>
    <p>Contenido desplega para C.</p>
  </div>
  <div data-role="collapsible" data-collapsed="true">
    <h3>Sección D</h3>
    <p>Contenido desplega para D.</p>
  </div>
</div>
```



Grupo de elementos desplegables

## 6. Introducción a jQuery Mobile - Ejercicios

En la parte dedicada a los ejercicios de jQuery Mobile vamos a desarrollar una parte muy reducida de una aplicación destinada a la gestión de una biblioteca de un centro educativo. Esta biblioteca básicamente contará por supuesto con *libros* y con una serie de tipos de *usuario* que podrán hacer realizar una serie de *operaciones*. En otras partes de este curso de especialista, desarrollaremos más ampliamente esta aplicación, pero en esta parte dedicada a la programación web con jQuery Mobile nos quedaremos en algo muy sencillo como es la interfaz de usuario desde el punto de vista del alumno

### 6.1. Ejercicio 1 - Inicio de la aplicación

En este primer ejercicio vamos a empezar por preparar la interfaz de usuario de lo que será nuestra aplicación y a definir el comportamiento de determinadas pantallas de la misma. Empecemos creando la página que será la portada de nuestra aplicación. Esta página debe contener una cabecera indicando el nombre de nuestra aplicación que llamaremos iBiblioteca.

A continuación, vamos a definir lo que será el contenido de esa página de inicio. Aquí vamos a redactar un pequeño texto formateado explicando brevemente el funcionamiento de la aplicación. Que no se te olvide indicar también quien ha desarrollado esta aplicación.

**Aviso:**

Las páginas de inicio en el mundo de las aplicaciones web para móvil no son nada deseables ya que lo más recomendable es que el usuario pueda directamente interactuar con la aplicación sin necesidad de aterrizar en una página de inicio que siempre muestra la misma información.

Por último, crear también un pie de página con una barra de navegación que nos servirá para tener siempre disponible una serie de opciones en cada página de nuestra aplicación. En esta primera página de la aplicación las opciones que vamos a añadir serán las siguientes:

- Botón para configurar una serie de ajustes en nuestra aplicación
- Botón para mostrar una pequeña ayuda de lo que nuestra aplicación puede hacer

No olvides poner unos iconos de acuerdo a la funcionalidad de cada botón y asegúrate también que esta barra de herramientas está siempre visible por el usuario.

El archivo html creado en este ejercicio se debe llamar *index.html* y debe estar en la raíz de un directorio llamado ibiblioteca.

### 6.2. Ejercicio 2 - Página del autor

En el ejercicio anterior hemos indicado en la página de bienvenida a la aplicación quien es el autor de la misma, sin embargo, no hemos añadido más información al mismo. En este ejercicio vamos a crear un nuevo documento html con toda la información del autor y en él practicaremos como crear una estructura de navegación sencilla para el usuario para que pueda moverse por la aplicación sin problemas.

Este nuevo documento relativo al autor tendrá más de una página y en cada una de ellas añadiremos unos datos diferentes. En una primera página incluiremos datos como el nombre completo y otros como el correo electrónico o los teléfonos de contacto. Incluso podemos añadir también una foto de tamaño reducido.

En una segunda página añadiremos información relativa a nuestro puesto actual (programador, analista, arquitecto, etc) y un pequeño listado de los puestos ocupados recientemente, cada uno de ellos con un enlace que nos llevará a una tercera página donde se explicará el cargo ocupado en ese puesto.

Ten en cuenta que siempre debemos volver atrás en todas las pantallas de nuestra aplicación. Recuerda también especificar una transición idónea para estos enlaces.

El archivo html creado en este ejercicio se debe llamar *author.html* y debe estar en la raíz de un directorio llamado *ibiblioteca*. Para la fotografía del autor, crea un directorio llamado *images* y ponla en la raíz de este directorio.

### 6.3. Ejercicio 3 - Listado de libros

En este ejercicio vamos a preparar una página en la que mostrar todos los libros de nuestra biblioteca y para ello utilizaremos los *acordeones*. En estos acordeones presentaremos en primer lugar el título del libro y una vez hagamos clic en ellos, mostraremos otras opciones de los libros como son el autor, el isbn y la editorial.

En el siguiente [enlace podéis descargaros](#) tanto un xml con libros de ejemplo como sus correspondientes imágenes de las portadas de los libros.

En cada libro, además daremos la posibilidad al usuario para que realice una reserva del mismo y marcarlo como favorito. Utiliza un tema diferente en estos enlaces para resaltar las opciones. Añade por último una opción a cada libro para que se pueda ver la portada del libro.

El archivo html creado en este ejercicio se debe llamar *books.html* y debe estar en la raíz de un directorio llamado *ibiblioteca*.

## 7. Aspectos avanzados de jQuery Mobile

Una vez vistos los aspectos más básicos de jQuery Mobile, en esta sesión vamos a profundizar en aspectos algo más avanzados y empezaremos viendo como podemos introducir listas de elementos (algo tan habitual en las aplicaciones web para móviles). Seguiremos analizando todos los elementos de formulario con las peculiaridades que se incluyen en jQuery Mobile. Por último, veremos algunos ejemplos para consumir servicios REST en nuestras aplicaciones web para móviles.

### 7.1. Listados de elementos

Uno de los aspectos más importantes en cualquier aplicación web para móviles es la forma en la que se muestran los listados. Debemos tener siempre en cuenta el espacio limitado que tenemos en una pantalla de un dispositivo móvil, con lo que estos listados deben aprovechar al máximo este espacio limitado.

Los listados de elementos nos servirán entre otras cosas para mostrar datos, para crear un sistema de navegación, para mostrar listas de resultados e incluso para crear sistemas de entrada de información. jQuery Mobile tiene una gran variedad de tipos de listas que cubrirán la mayoría de nuestras necesidades. Pasemos a ver estos tipos de listas y todas sus características.

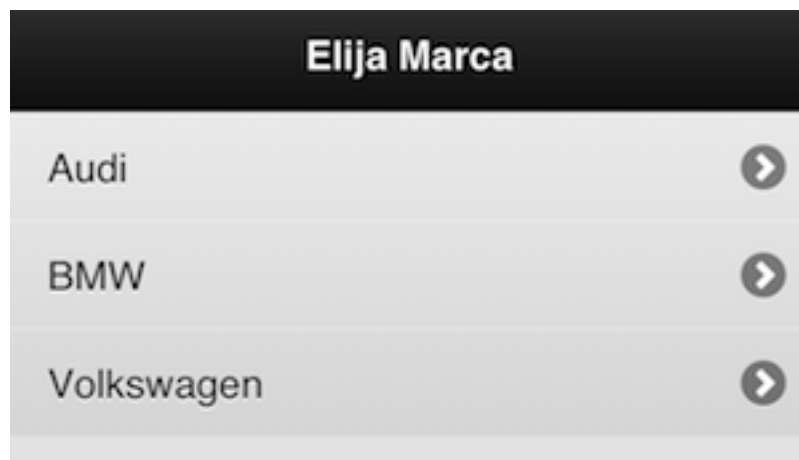
#### 7.1.1. Listados básicos con enlaces

Un listado en jQuery Mobile no es más que una lista desordenada en la que sus elementos son enlaces y que tiene el atributo `data-role="listview"`. Cuando jQuery Mobile detecte este elemento del DOM lo transformará en una lista adaptada a los dispositivos móviles y le añadirá incluso una flecha a la parte derecha para indicar al usuario que en ese elemento es posible hacer clic.

Cuando el usuario haga clic en el enlace, éste se cargará vía AJAX mostrando una transición de página. Veamos un poco de código:

```
<ul data-role="listview" data-theme="g">
  <li><a href="audi.html">Audi</a></li>
  <li><a href="bmw.html">BMW</a></li>
  <li><a href="volkswagen.html">Volkswagen</a></li>
</ul>
```



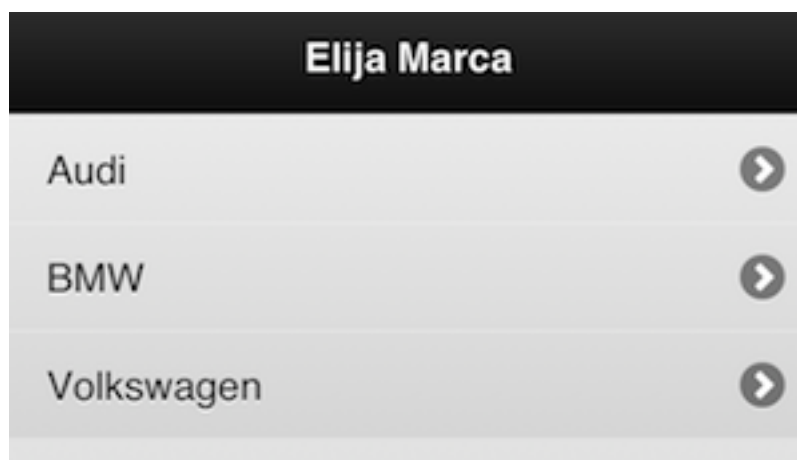


Listados básicos

### 7.1.2. Listados anidados

El concepto de listados anidados es tan simple como incluir una lista dentro de otra. En estos casos, jQuery Mobile se encargará de mostrar las listas de forma separada. Esto es, en primer lugar se mostrará la lista principal y al pinchar en cada uno de los elementos se mostrará la sublista incluida en ese elemento. Veamos un ejemplo:

```
<ul data-role="listview" data-theme="g">
  <li>
    Audi
    <ul data-role="listview">
      <li>A1</li>
      <li>A2</li>
      <li>A3</li>
    </ul>
  </li>
  <li>
    BMW
    <ul data-role="listview">
      <li>Serie 1</li>
      <li>Serie 2</li>
      <li>Serie 3</li>
    </ul>
  </li>
  <li>
    Volkswagen
    <ul>
      <li>Golf</li>
      <li>Passat</li>
      <li>Touran</li>
    </ul>
  </li>
</ul>
```



Listados básicos

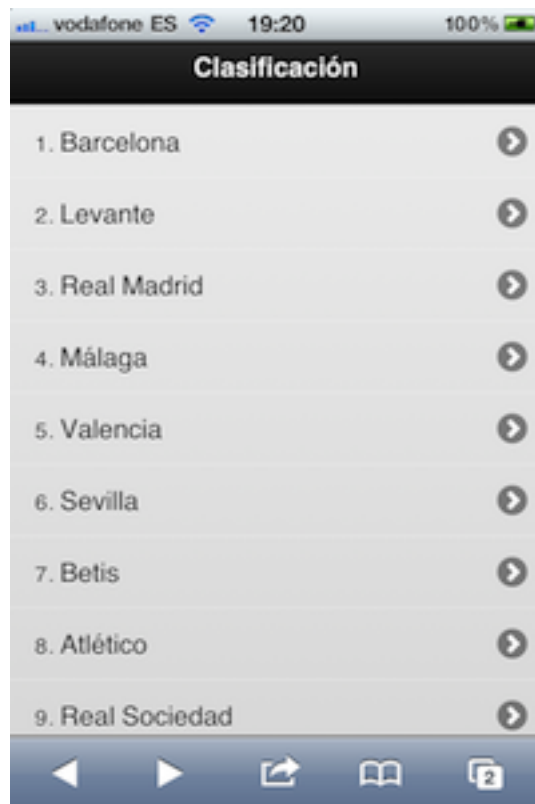


Listados anidados

### 7.1.3. Listados numerados

Hasta ahora siempre hemos visto los listados no numerados, sin embargo, en jQuery Mobile también es posible utilizar los listados numerados. Un típico caso de listado numerado puede ser el resultado de una búsqueda o para indicar por la clasificación de un torneo cualquiera.

```
<ol data-role="listview" data-theme="g">
  <li><a href="barca.html">Barcelona</a></li>
  <li><a href="levantado.html">Levante</a></li>
  <li><a href="realmadrid.html">Real Madrid</a></li>
  <li><a href="malaga.html">Málaga</a></li>
  <li><a href="valencia.html">Valencia</a></li>
  <li><a href="sevilla.html">Sevilla</a></li>
  <li><a href="betis.html">Betis</a></li>
  <li><a href="atletico.html">Atlético</a></li>
  <li><a href="rsociedad.html">Real Sociedad</a></li>
  <li><a href="mallorca.html">Mallorca</a></li>
</ol>
```



Listados numéricos

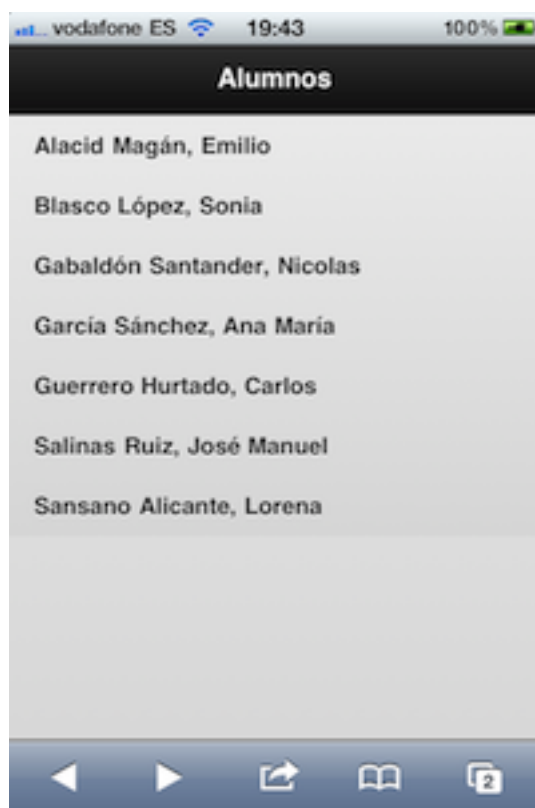
#### 7.1.4. Listados de solo lectura

Por supuesto, no siempre necesitaremos que un elemento de una lista haga una determinada acción y simplemente vamos a querer mostrar determinada información en este listado. En este tipo de listados vamos a poder tanto listas ordenadas como listas desordenadas.

##### 7.1.4.1. Listados simples

En este tipo de listados no hay posibilidad de hacer clic sobre los elementos de la lista e incluso el tamaño del texto es considerablemente más pequeño que en las listas con enlaces. Imagina por ejemplo un simple listado de alumnos.

```
<ul data-role="listview" data-theme="g">
  <li>Alacid Magán, Emilio</li>
  <li>Blasco López, Sonia</li>
  <li>Gabaldón Santander, Nicolas</li>
  <li>García Sánchez, Ana María</li>
  <li>Guerrero Hurtado, Carlos</li>
  <li>Salinas Ruiz, José Manuel</li>
  <li>Sansano Alicante, Lorena</li>
</ul>
```

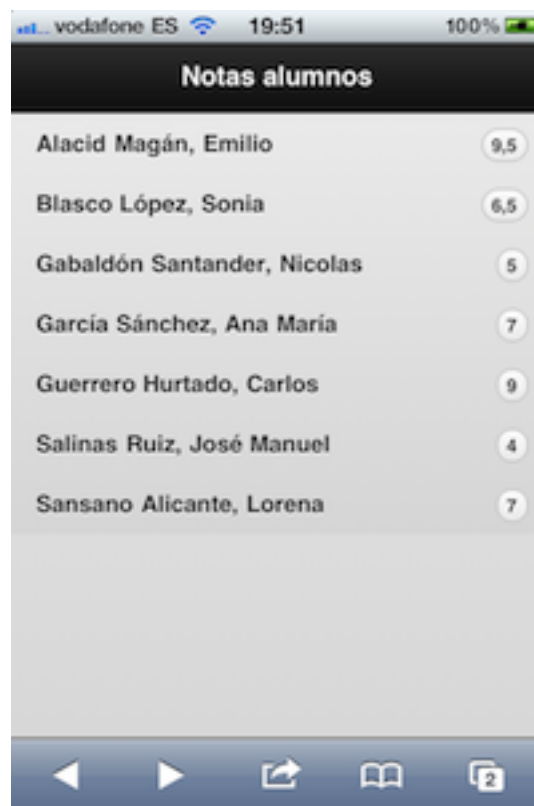


Listas básicas de solo lectura

#### 7.1.4.2. Listados simples con información numérica

En ocasiones, estos listados de solo lectura proporcionan algo de información numérica. Continuando con el ejemplo anterior de los alumnos podríamos tener una nota asociada a los mismos. Para introducir esta información numérica únicamente debemos añadir un elemento de tipo `div` con el atributo `class="ui-li-count"`.

```
<ul data-role="listview" data-theme="g">
  <li>Alacid Magán, Emilio <span class="ui-li-count">9,5</span></li>
  <li>Blasco López, Sonia <span class="ui-li-count">6,5</span></li>
  <li>Gabaldón Santander, Nicolas <span
class="ui-li-count">5</span></li>
  <li>García Sánchez, Ana María <span
class="ui-li-count">7</span></li>
  <li>Guerrero Hurtado, Carlos <span
class="ui-li-count">9</span></li>
  <li>Salinas Ruiz, José Manuel <span
class="ui-li-count">4</span></li>
  <li>Sansano Alicante, Lorena <span
class="ui-li-count">7</span></li>
</ul>
```



Listas básicas de solo lectura con información numérica

#### 7.1.4.3. Listados formateados

Por supuesto, no tenemos porque utilizar elementos de una sola línea sino que éstos pueden estar formados por más de una línea indicando información complementaria al elemento.

```
<ul data-role="listview">
  <li>
    <h3>Alacid, Emilio</h3>
    <p><strong>Un fenómeno en las matemáticas</strong></p>
    <p>Comentar el incidente de los pinceles</p>
    <p class="ui-li-aside"><strong>17:10</strong></p>
  </li>
  <li>
    <h3>Blasco, Sonia</h3>
    <p><strong>Atiende mucho en clase</strong></p>
    <p>En ocasiones viene sin los deberes hechos</p>
    <p class="ui-li-aside"><strong>17:35</strong></p>
  </li>
  <li>
    <h3>Gabaldón, Nicolas</h3>
    <p><strong>Le encanta el medio ambiente</strong></p>
    <p>Se despista con una mosca</p>
    <p class="ui-li-aside"><strong>18:00</strong></p>
  </li>
</ul>
```

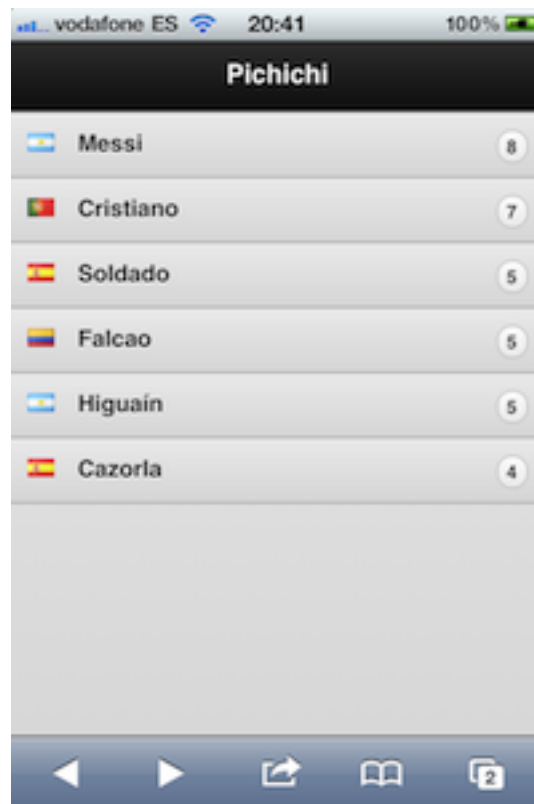


Listas básicas de solo lectura con información formateada

#### 7.1.4.4. Listados con imágenes

Por supuesto, en los listados también vamos a poder añadir imágenes como si fuera iconos para facilitar la comprensión por parte del usuario. Simplemente debemos añadir la imagen que queramos mostrar e indicarle el atributo `class="ui-li-icon"`.

```
<ul data-role="listview">
  <li>
    Messi <span class="ui-li-count">8</span>
  </li>
  <li>
    Cristiano <span class="ui-li-count">7</span>
  </li>
  <li>
    Soldado <span class="ui-li-count">5</span>
  </li>
  <li>
    Falcao <span class="ui-li-count">5</span>
  </li>
  <li>
    Higuaín <span class="ui-li-count">5</span>
  </li>
  <li>
    Cazorla <span class="ui-li-count">4</span>
  </li>
</ul>
```



Listas básicas de solo lectura con iconos

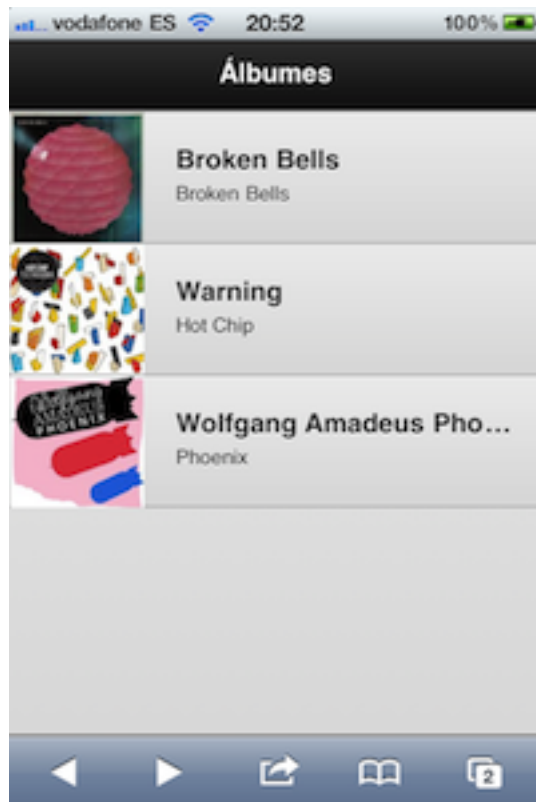
[Aquí tienes los iconos de las imágenes](#)

#### 7.1.4.5. Listados con thumbnails

Otro tipo de listado similar al de las imágenes sería aquellos listados que se presentan con una imagen de tipo *thumbnail* asociada al texto. Piensa por ejemplo un listado de álbumes musicales.

```
<ul data-role="listview">
  <li>
    
    <h3>Broken Bells</h3>
    <p>Broken Bells</p>
  </li>
  <li>
    
    <h3>Warning</h3>
    <p>Hot Chip</p>
  </li>
  <li>
    
    <h3>Wolfgang Amadeus Phoenix</h3>
    <p>Phoenix</p>
  </li>
</ul>
```

```
</ul>
```



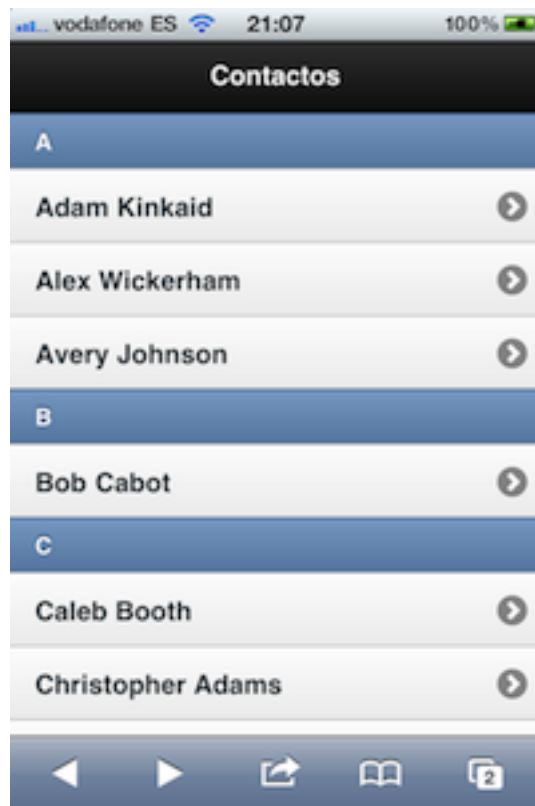
Listas básicas de solo lectura con iconos

### 7.1.5. Listados divididos y agrupados

Otra posibilidad de los listados es mostrarlos agrupados por ejemplo por orden alfabético. Estos listados se pueden implementar fácilmente simplemente indicando en el elemento divisor el atributo `data-role="list-divider"` tal y como vemos en el siguiente ejemplo.

```
<ul data-role="listview">
  <li data-role="list-divider">A</li>
  <li><a href="index.html">Adam Kinkaid</a></li>
  <li><a href="index.html">Alex Wickerham</a></li>
  <li><a href="index.html">Avery Johnson</a></li>
  <li data-role="list-divider">B</li>
  <li><a href="index.html">Bob Cabot</a></li>
  <li data-role="list-divider">C</li>
  <li><a href="index.html">Caleb Booth</a></li>
  <li><a href="index.html">Christopher Adams</a></li>
  <li><a href="index.html">Culver James</a></li>
</ul>
```





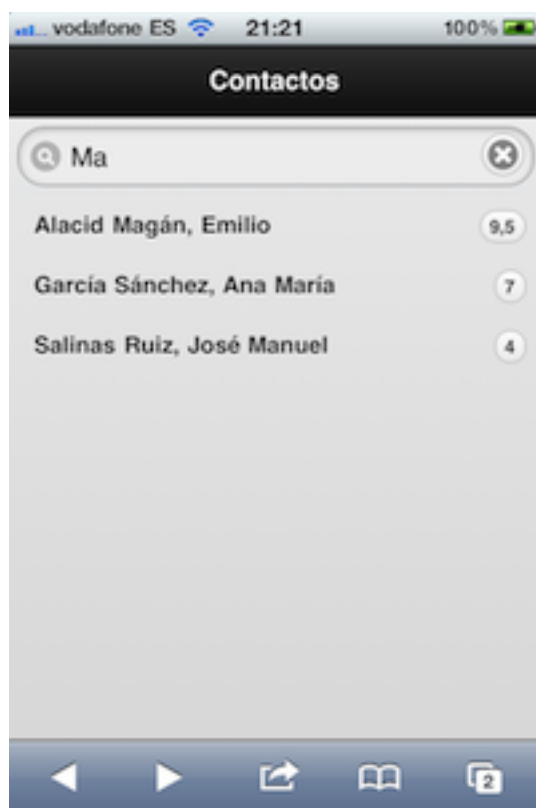
Listas agrupadas

### 7.1.6. Listados con filtros

jQuery Mobile además incorpora la posibilidad de implementar fácilmente la búsqueda de textos en las listas de nuestras aplicaciones. Esto se puede conseguir simplemente indicando el atributo `data-filter="true"` a la lista en cuestión.

Con esto veremos que jQuery Mobile añadirá una caja de texto para la búsqueda de texto en los elementos de nuestra lista que nos servirá para filtrar únicamente aquellos elementos que contengan el texto buscado. Esta caja de texto vendrá completada con un texto por defecto que podremos cambiar modificando el atributo `data-filter-placeholder="Filtro de búsqueda"`.

```
<ul data-role="listview" data-filter="true"
data-filter-placeholder="Filtro de búsqueda">
  <li>Alacid Magán, Emilio <span class="ui-li-count">9,5</span></li>
  <li>Blasco López, Sonia <span class="ui-li-count">6,5</span></li>
  <li>Gabaldón Santander, Nicolas <span
class="ui-li-count">5</span></li>
  <li>García Sánchez, Ana María <span class="ui-li-count">7</span></li>
  <li>Guerrero Hurtado, Carlos <span class="ui-li-count">9</span></li>
  <li>Salinas Ruiz, José Manuel <span class="ui-li-count">4</span></li>
  <li>Sansano Alicante, Lorena <span class="ui-li-count">7</span></li>
</ul>
```



Listas con filtros

## 7.2. Formularios

Sin duda, uno de los aspectos más importantes de cualquier aplicación web ya esté dirigida a un entorno móvil o de escritorio es la interacción con el usuario mediante formularios.

En jQuery Mobile, todos los elementos de formulario siguen el estándar de HTML añadiéndole algunas características para hacerlos más atractivos sobre todo a nivel visual y de usabilidad en un dispositivo móvil.

En aquellos navegadores que no soporten las transformaciones oportunas, estos elementos se verán sin ningún problema pero con el diseño habitual de los mismos.

### 7.2.1. Conceptos básicos

Por supuesto, para crear un formulario debemos utilizar, como se hace en HTML habitualmente, la etiqueta `form` seguido de los atributos `action` para indicar quien se encargará de procesar esa petición y `method` que nos servirá para señalar como queremos pasar esos datos (habitualmente GET o POST).

```
<form action="form.php" method="post">
  ...
</form>
```

Por suerte, todo lo que ya conocemos a la hora de crear formularios en HTML nos va a servir cuando creemos formularios con jQuery Mobile. Sin embargo, debemos tener en cuenta que todos los elementos de un formulario deben especificar el atributo `id`. Este atributo habitualmente debe ser único en cada página HTML, sin embargo, con jQuery Mobile este atributo no puede repetirse a lo largo de toda la aplicación ya que jQuery Mobile trabaja modificando el DOM y no es posible tener dos elementos cargados con el mismo identificador.

Además, es muy importante que se utilice la etiqueta `label` con el atributo `for` especificando el elemento de formulario que queremos etiquetar.

jQuery Mobile siempre trabaja de la misma forma. Cuando el usuario carga una determinada página, se analiza todo el DOM y se realizan una serie de sustituciones. Por ejemplo, cuando utilizamos un elemento de formulario de tipo `select`, jQuery Mobile utiliza un plugin llamado *selectmenu* y realiza las sustituciones indicadas en este plugin.

Esto es lo que hace jQuery Mobile por defecto, sin embargo nosotros podemos indicarle a un determinado elemento de formulario que no queremos que se realice ninguna sustitución simplemente añadiéndole el atributo `data-role="none"`, tal y como vemos en el siguiente ejemplo.

```
<label for="marcacoche">Marca</label>
<select name="marcacoche" id="marcacoche" data-role="none">
  <option value="audi" >Audi</option>
  <option value="bmw" >BMW</option>
  <option value="volkswagen" >Volkswagen</option>
</select>
```

Algo también muy interesante es como se organizan los diversos elementos de un formulario y es que jQuery Mobile trata de posicionar todos estos elementos para aprovechar al máximo el ancho de un dispositivo móvil.

Por ejemplo, en aquellos dispositivos estrechos, los elementos `label` se sitúan justo encima del elemento de formulario al que etiquetan. Sin embargo, en los dispositivos más anchos, estas etiquetas aparecen a la izquierda de su elemento de formulario.

Por último, comentar también que para mejorar la experiencia del usuario con los formularios, es aconsejable utilizar elementos de tipo `div` o `fieldset` con el atributo `data-role="fieldcontain"` para envolver todos los elementos del formulario.

### 7.2.2. Elementos de formularios

Veamos a continuación detalladamente todos los elementos de formulario y sus características específicas en jQuery Mobile.

### 7.2.2.1. Cajas de texto

Las cajas de texto de una sola línea se insertan como si fuera HTML puro y duro y jQuery Mobile se encargará de hacerlos más atractivos y fáciles de utilizar para un usuario con un dispositivo móvil, tal y como vemos en el siguiente ejemplo.

```
<div data-role="fieldcontain">
  <label for="name">Nombre</label>
  <input type="text" name="name" id="name" value="" />
</div>
```


 A screenshot of a mobile application interface. At the top is a dark header bar with the word 'Contacto' in white. Below the header is a light gray background. A label 'Nombre' is positioned above a single-line text input field. The input field has rounded corners and a subtle shadow.

Cajas de texto

Además, en jQuery Mobile también podemos utilizar cualquiera de los nuevos tipos introducidos en HTML5 (password, email, tel, number, url, etc).

```
<div data-role="fieldcontain">
  <label for="email">Correo electrónico</label>
  <input type="email" name="email" id="email" value="" />
  <label for="password">Contraseña</label>
  <input type="password" name="password" id="password" value="" />
  <label for="age">Edad</label>
  <input type="number" name="age" id="age" value="" />
  <label for="url">Url</label>
  <input type="url" name="url" id="url" value="" />
  <label for="telephone">Teléfono</label>
  <input type="tel" name="telephone" id="telephone" value="" />
</div>
```

A mobile application form titled "Contacto" with a dark header. Below the header, there are five input fields, each with a label above it: "Correo electrónico", "Contraseña", "Edad", "Url", and "Teléfono". The fields are simple rounded rectangles with a light gray background and a thin border.

Más opciones de las cajas de texto

Si compruebas este ejemplo, verás que al intentar editar cada una de las cajas de texto el método de entrada será diferente.

También podemos utilizar las cajas de texto multilínea `textarea` que también siguen el mismo formato que en HTML y es jQuery Mobile quien se encargará de ajustar la altura para evitar tener que pintar un scroll innecesario a medida que el usuario va escribiendo.

```
<div data-role="fieldcontain">
  <label for="description">Descripción</label>
  <textarea name="description" id="description"></textarea>
</div>
```

Por último, en HTML5 se ha añadido un nuevo tipo llamado `search`, que como su nombre indica nos servirá para introducir cajas de texto que se utilizarán para implementar búsquedas en nuestra aplicación.

```
<div data-role="fieldcontain">
  <label for="search">Buscar</label>
  <input type="search" name="search" id="search" placeholder="texto a
  buscar" value="" />
</div>
```



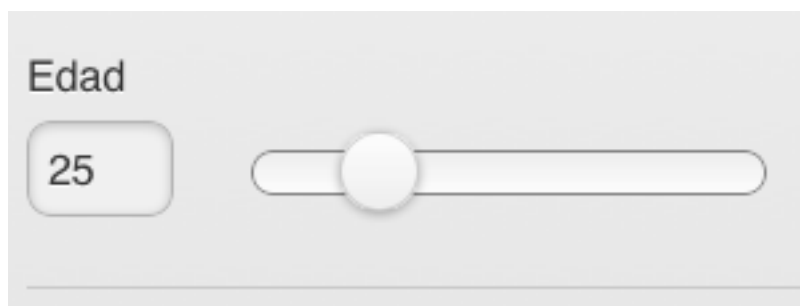
Cajas de búsqueda de texto

### 7.2.2.2. Deslizadores

Uno de los elementos introducidos en la especificación del HTML5 ha sido la posibilidad de utilizar el elemento de formulario de tipo `range`. El usuario podrá deslizar fácilmente este elemento y al mismo irá tomando un valor diferente para cada posición.

En estos elementos debemos especificar los atributos `min` y `max` que servirán para indicar un valor mínimo y un máximo y por otro lado, también podemos indicarle un valor actual con el atributo `value`.

```
<div data-role="fieldcontain">
  <label for="age">Edad</label>
  <input type="range" name="age" id="age" value="25" min="0" max="100"
/>
</div>
```



Deslizador

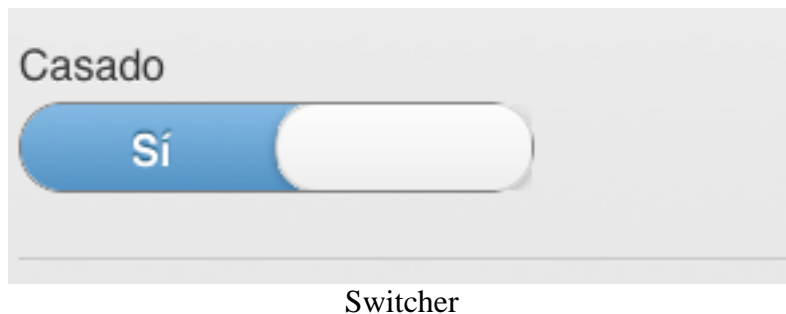
Como vemos en la imagen, a la izquierda del deslizador aparece una caja de texto de tal forma que el usuario podrá utilizar éste o bien ser más preciso escribiendo el texto directamente, lo que se verá automáticamente reflejado en el deslizador.

### 7.2.2.3. Switchers

En jQuery Mobile tenemos también la posibilidad de añadir un elemento de formulario nuevo que nos permitirá indicar al usuario que debe elegir entre dos valores, como pueden ser *on/off* o *true/false*. Este elemento se mostrará de tal forma que el usuario podrá con sus dedos cambiar el estado del mismo fácilmente.

Estos elementos se definen como si fuera un elemento de tipo `select` con dos opciones. Además, el `select` debe tener el atributo `data-role="slider"`. El primer elemento del `select` será tratado como el estado *on* o *true*, mientras que el segundo será *off* o *false*.

```
<div data-role="fieldcontain">
  <label for="casado">Casado</label>
  <select name="casado" id="casado" data-role="slider">
    <option value="no">No</option>
    <option value="yes">Sí</option>
  </select>
</div>
```



#### 7.2.2.4. Elementos de tipo radio

Este tipo de elementos se utilizan para proporcionar al usuario una serie de elementos de los cuales éste debe seleccionar solamente uno. En las aplicaciones de escritorio este tipo de elementos no están optimizados para el uso en dispositivos móviles, pero con jQuery Mobile éstos se nos mostrarán mucho más accesibles.

Para crear un conjunto de elementos de tipo radio simplemente debemos añadir la etiqueta `input` con el atributo `type="radio"` y su correspondiente `label`.

Es aconsejable además que todos los elementos de tipo radio queden envueltos en las etiquetas `fieldset` y `legend`. Esta última actuará como título del elemento.

Por último, la etiqueta `fieldset` debe quedar envuelta a su vez por una etiqueta `div` con el atributo `data-role="controlgroup"`, tal y como vemos en el siguiente ejemplo.

```
<div data-role="fieldcontain">
  <fieldset data-role="controlgroup">
    <legend>Marca de coche</legend>
    <input type="radio" name="radio-choice-1"
id="radio-choice-1" value="audi" checked="checked" />
    <label for="radio-choice-1">Audi</label>

    <input type="radio" name="radio-choice-1"
id="radio-choice-2" value="bmw" />
    <label for="radio-choice-2">BMW</label>

    <input type="radio" name="radio-choice-1"
id="radio-choice-3" value="seat" />
    <label for="radio-choice-3">Seat</label>

    <input type="radio" name="radio-choice-1"
```

```
id="radio-choice-4" value="volkswagen" />
    <label for="radio-choice-4">Volkswagen</label>
  </fieldset>
</div>
```

Marca de coche

- ☒ Audi
- ☐ BMW
- ☐ Seat
- ☐ Volkswagen

Radio vertical

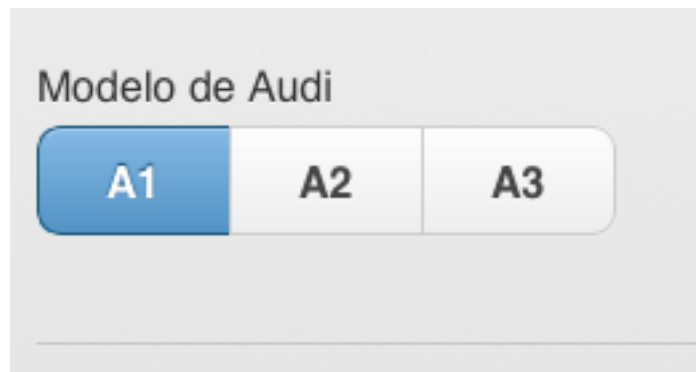
En el ejemplo anterior, el grupo de elementos aparece de forma vertical, pero también es posible que aparezcan horizontalmente simplemente especificando el atributo `data-type="horizontal"` a la etiqueta `fieldset`.

```
<div data-role="fieldcontain">
  <fieldset data-role="controlgroup" data-type="horizontal">
    <legend>Modelo de Audi</legend>
    <input type="radio" name="radio-choice-1"
id="radio-choice-1" value="a1" checked="checked" />
    <label for="radio-choice-1">A1</label>

    <input type="radio" name="radio-choice-1"
id="radio-choice-2" value="a2" />
    <label for="radio-choice-2">A2</label>

    <input type="radio" name="radio-choice-1"
id="radio-choice-3" value="a3" />
    <label for="radio-choice-3">A3</label>
  </fieldset>
</div>
```





Radio horizontal

#### 7.2.2.5. Elementos de tipo checkbox

Este tipo de elementos se utilizan para proporcionar al usuario una serie de opciones de las cuales puede seleccionar más de una. Al igual que los elementos de tipo radio, los elementos de tipo checkbox utilizan la misma sintaxis que en HTML puro y duro y es jQuery Mobile el encargado de realizar las transformaciones oportunas para adaptarlas a un entorno móvil.

Para añadir uno de estos elementos debemos utilizar la etiqueta `input` con el atributo `type="checkbox"` y su correspondiente etiqueta `label` con el atributo `for` correctamente asociado al identificador del checkbox.

Por último y al igual que sucedía con los elementos de tipo radio, la etiqueta `fieldset` debe quedar envuelta a su vez por una etiqueta `div` con el atributo `data-role="controlgroup"`, tal y como vemos en el siguiente ejemplo.

```
<div data-role="fieldcontain">
  <fieldset data-role="controlgroup">
    <legend>De acuerdo con los términos del contrato:</legend>
    <input type="checkbox" name="checkbox-1" id="checkbox-1"
class="custom" />
    <label for="checkbox-1">Sí, estoy de acuerdo</label>
  </fieldset>
</div>
```

**Contrato**

De acuerdo con los términos del contrato:

☒ **Sí, estoy de acuerdo**

Checkbox horizontal

Por defecto, los elementos de tipo checkbox aparecerán agrupados de forma vertical, tal y como se muestra en el siguiente ejemplo.

```
<div data-role="fieldcontain">
  <fieldset data-role="controlgroup">
    <legend>Indíquenos sus hobbies</legend>
    <input type="checkbox" name="musica" id="musica"/>
    <label for="musica">Música</label>
    <input type="checkbox" name="deporte" id="deporte"/>
    <label for="deporte">Deportes</label>
    <input type="checkbox" name="television" id="television"/>
    <label for="television">Televisión</label>
    <input type="checkbox" name="cine" id="cine"/>
    <label for="cine">Cine</label>
  </fieldset>
</div>
```

Registro

Indíquenos sus hobbies

☐ Música

☒ Deportes

☐ Televisión

☒ Cine

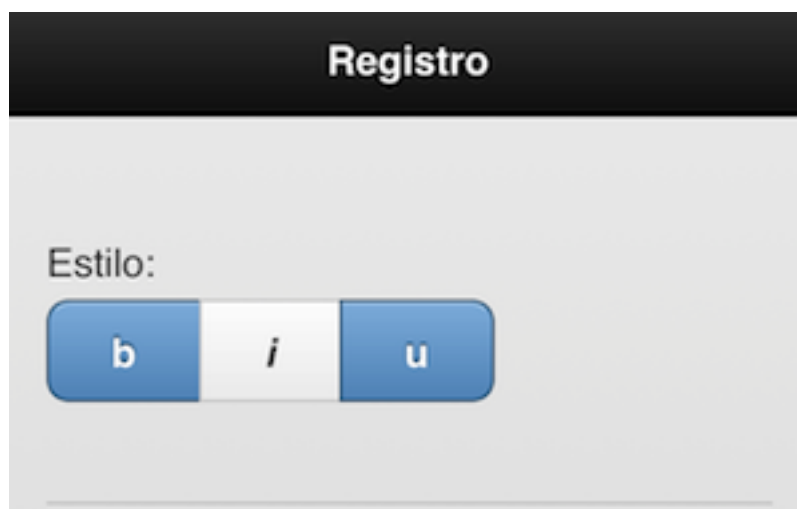
Checkbox vertical

Por último, si en lugar de mostrarlos de forma vertical queremos hacerlo de forma horizontal, podemos añadir el atributo `data-type="horizontal"`.

```
<div data-role="fieldcontain">
  <fieldset data-role="controlgroup" data-type="horizontal">
    <legend>Estilo:</legend>
    <input type="checkbox" name="bold" id="bold"/>
    <label for="bold">b</label>

    <input type="checkbox" name="cursive" id="cursive"/>
    <label for="cursive"><em>i</em></label>

    <input type="checkbox" name="underline" id="underline"/>
    <label for="underline">u</label>
  </fieldset>
</div>
```



Checkbox horizontal

#### 7.2.2.6. Elementos de tipo select

Para terminar con los diferentes tipos de elementos de formulario, vamos a ver el elemento de tipo `select`. Este tipo nos permitirá seleccionar un solo elemento de una lista.

Como los elementos de tipo `radio` y `checkbox`, éstos también tienen la sintaxis típica de HTML y será nuevamente jQuery Mobile quien se encargue de realizar las transformaciones oportunas para mejorar la experiencia del usuario de dispositivos móviles.

Para añadir un elemento de este tipo debemos utilizar la etiqueta `select` con una serie de elementos de tipo `option`. Debemos también relacionar este elemento con una etiqueta de tipo `label`. Además, debemos también agrupar este elemento dentro de un elemento de tipo `div` con el atributo `data-role="fieldcontain"`. Veamos un ejemplo:

```
<div data-role="fieldcontain">
  <label for="tiposuscripcion">Tipo de suscripción:</label>
  <select name="tiposuscripcion" id="tiposuscripcion">
    <option value="diaria">Diaria</option>
    <option value="semanal">Semanal</option>
    <option value="mensual">Mensual</option>
    <option value="anual">Anual</option>
  </select>
</div>
```



Select

Como podemos comprobar en la imagen, este tipo de elementos se mostrarán de forma nativa en los diferentes dispositivos donde carguemos nuestra aplicación. Si queremos modificar este comportamiento y mostrar las opciones siempre de la misma forma y con algo más de estilo propio, podemos utilizar el atributo `data-native-menu="false"` en el elemento `select` obteniendo lo siguiente:



Select no nativo

Si necesitas además en tu aplicación que tus usuarios puedan utilizar la sección múltiple, jQuery Mobile también nos va a facilitar esta labor. Únicamente debemos añadir a la etiqueta `select` el atributo `multiple="multiple"`. Además, también debemos añadir un primer elemento sin valor que se mostrará como una cabecera del `select`.

```
<div data-role="fieldcontain">
  <label for="tiposuscripcion">Tipo de suscripción:</label>
  <select name="tiposuscripcion" id="tiposuscripcion" multiple="multiple"
data-native-menu="false">
    <option>Selecciona opciones</option>
    <option value="diaria">Diaría</option>
    <option value="semanal">Semanal</option>
    <option value="mensual">Mensual</option>
    <option value="anual">Anual</option>
  </select>
</div>
```



Select múltiple

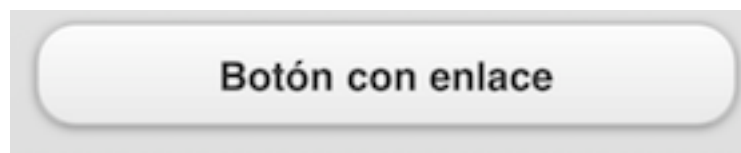
Como vemos en la imagen, jQuery Mobile se encarga de poner un encabezado con la primera opción del `select` que además podremos cerrar con un botón también añadido automáticamente.

### 7.2.3. Botones

Una vez vistos todos los elementos de formulario, pasemos a ver los botones en jQuery Mobile. Los botones con jQuery Mobile se especifican como si fuera HTML pero como siempre, éstos se presentarán de una forma más atractiva para los clientes móviles.

Sin embargo, existe otra posibilidad para pintar botones en nuestras aplicaciones que es especificarlo mediante un enlace con el atributo `data-role="button"`.

```
<a href="index.html" data-role="button">Botón con enlace</a>
```



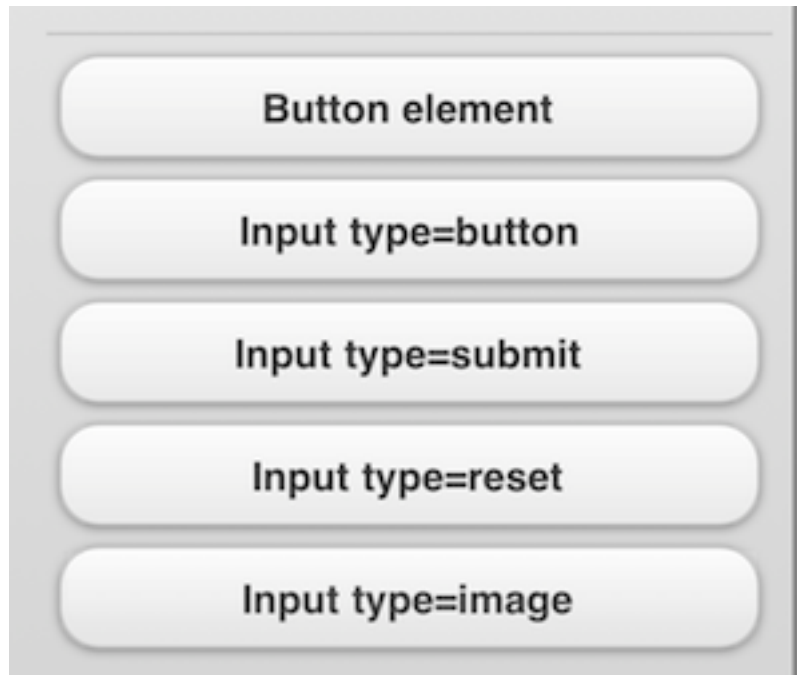
Botón con enlace

Por otro lado, si utilizamos la sintaxis típica para los botones en HTML, tenemos las siguientes representaciones.

```

<button>Elemento button</button>
<input type="button" value="Input type=button" />
<input type="submit" value="Input type=submit" />
<input type="reset" value="Input type=reset" />
<input type="image" value="Input type=image" />

```



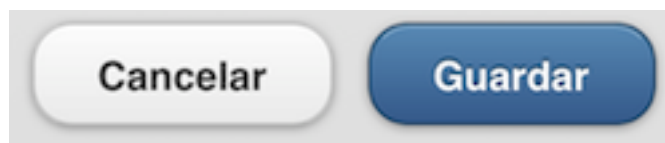
Botón con inputs

Por defecto, los botones se muestra a nivel de bloque, esto es que ocuparán todo el ancho posible de la página. Si queremos pintar más de un botón en una misma línea, debemos utilizar el atributo `data-inline="true"` a cada botón.

```

<a href="index.html" data-role="button" data-inline="true">Cancelar</a>
<a href="index.html" data-role="button" data-inline="true"
data-theme="b">Guardar</a>

```



Botón en línea

Pero además, jQuery Mobile nos permite agrupar botones tanto de forma vertical.

```

<div data-role="controlgroup">
<a href="index.html" data-role="button">Sí</a>
<a href="index.html" data-role="button">No</a>
<a href="index.html" data-role="button">Quizás</a>
</div>

```

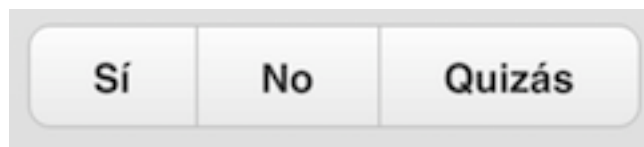




Botones agrupados

Y en el caso que queramos mostrar los botones de forma horizontal, deberemos utilizar el atributo `data-type="horizontal"` al contenedor con el atributo `data-role="controlgroup"`.

```
<div data-role="controlgroup" data-type="horizontal">
<a href="index.html" data-role="button">Sí</a>
<a href="index.html" data-role="button">No</a>
<a href="index.html" data-role="button">Quizás</a>
</div>
```



Botones agrupados horizontalmente

#### 7.2.4. Envío de formularios

Y por último, ahora que ya conocemos ampliamente como podemos preparar todo tipo de formularios con jQuery Mobile, vamos a pasar a ver como se envía la información cumplimentada en estos formularios a un servidor que la procese.

jQuery Mobile por defecto procesa el envío de los formularios mediante llamadas Ajax, creando incluso una transición entre el formulario y la página resultante. Para asegurarnos de que nuestro formulario se procesa correctamente, debemos especificar los atributos `action` y `method`. En caso de que no especifiquemos estos valores, el método pasado será `GET` y el atributo `action` será la misma página que contiene el formulario.

Los formularios incluso aceptan otros parámetros como `data-transition="pop"` y `data-direction="reverse"`. Incluso, si no queremos que el formulario sea procesado vía Ajax, podemos desactivar este comportamiento especificando el atributo `data-ajax="false"`. Además, también podemos especificar el atributo `target="_blank"`.

#### 7.2.5. Consumir servicios REST

Aunque la parte de la programación del servidor queda fuera de los objetivos de este curso, vamos a ver una pequeña introducción a un tema muy interesante como es la consumición de servicios REST.

#### ¿Qué es REST?

Es la abreviatura de *REpresentational State Transfer* y no es más que un conjunto de principios en el que se definen una serie de estándares web y como deben utilizarse éstos.

Básicamente vamos a tener dos formas de consumir estos servicios REST por parte de terceros. Por un lado el archiconocido formato XML y por otro lado el no tan conocido formato JSON (que será el que utilizaremos aquí).

El formato JSON es el acrónimo de *Javascript Object Notation* y se considera como un formato ligero para el intercambio de datos entre aplicaciones web. Este puede ser un ejemplo de archivo json:

```
{ "menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      { "value": "New", "onclick": "CreateNewDoc()" },
      { "value": "Open", "onclick": "OpenDoc()" },
      { "value": "Close", "onclick": "CloseDoc()" }
    ]
  }
}
```

Para consumir servicios REST en formato JSON, jQuery proporciona la llamada \$.getJSON(). El primer parámetro de esta función es la URI del servicio REST y el segundo será la función que se encargará de tramitar esta petición.

Pero para entender mejor como funciona, veamos un ejemplo que recupere todos los libros de nuestra biblioteca. En la dirección <http://server.jtech.ua.es:8080/jbib-rest/resources/libros> tenemos un servicio REST implementado que nos devolverá los libros de una biblioteca. Nuestra tarea será mostrar ese listado de libros. Veamos como quedaría:

```
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="UTF-8">
  <title>Biblioteca</title>
  <meta name="viewport" content="width=device-width,
initial-scale=1">
  <link rel="stylesheet"
href="http://code.jquery.com/mobile/1.0b2/jquery.mobile-1.0b2.min.css" />
  <script type="text/javascript"
src="http://code.jquery.com/jquery-1.6.2.min.js"></script>
  <script type="text/javascript"
src="http://code.jquery.com/mobile/1.0b2/jquery.mobile-1.0b2.min.js"></script>
  <script text="text/javascript">
$.getJSON('http://server.jtech.ua.es:8080/jbib-rest/resources/libros',
function(data) {
```

```

        var items = [];

        $.each(data['libro'], function(key, val){
            items.push('<li data-theme="c" class="ui-btn
ui-btn-icon-right ui-li ui-btn-up-c">' +
                                '<div class="ui-btn-inner
ui-li"><div class="ui-btn-text">' +
                                ' <a rel="external"
href="librodetalle.php?method=post&url='+ val['link']['@uri'] +' "
class="ui-link-inherit">' + val['@titulo'] +
                                ' </a></div></div>'
+
                                '<span class="ui-icon
ui-icon-arrow-r"/></li>');
        });

        $('<ul/>', {
            'data-role': 'listview',
            'data-filter': 'true',
            'class': 'ui-listview',
            html: items.join('')
        }).appendTo('#listadoLibros');
    });
</script>
</head>
<body>
<div data-role="page" id="vistaLibros" data-theme="b">
    <div data-role="header" data-position="fixed">
        <h2>Biblioteca</h2>
    </div>
    <div data-role="content" id="listadoLibros">
    </div>
</div>
</body>
</html>

```

Si analizamos un poco el código, veremos como en primer lugar se obtienen todos los libros a partir del servicio REST pasado por parámetro (<http://server.jtech.ua.es:8080/jbib-rest/resources/libros>) y una vez con todos los libros, los vamos añadiendo de forma dinámica en un array (`items`). Una vez ya tenemos todos los libros cargados en el array, lo que hacemos es añadirlo al elemento del DOM con identificador `listadoLibros`.

Sin embargo, si intentamos cargar el ejemplo anterior, veremos como algo no funcionará como es debido. Este se produce porque, por motivos de seguridad, es imposible hacer peticiones javascript a dominios que no nos pertenezcan. Una solución a este problema es utilizar un proxy en nuestro propio servidor que se encargue de procesar dicha petición.

```

<?php
if ($_GET['url']== "")
    $url =
"http://server.jtech.ua.es:8080/jbib-rest/resources/libros";
else
    $url = $_GET['url'];

$ch = curl_init($url);

switch (strtolower($_GET['method'])){
    case "post":curl_setopt($ch, CURLOPT_POST,1);
                curl_setopt($ch,

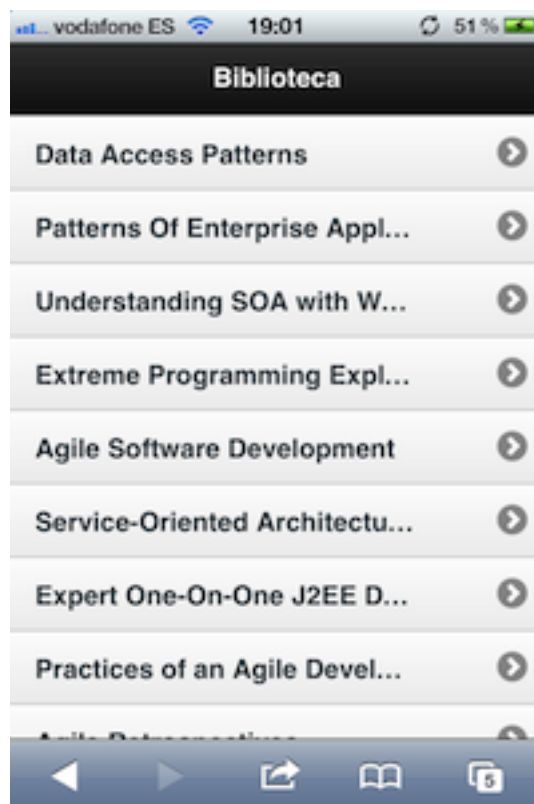
```

```

CURLOPT_USERPWD, "profesor:profesor");
        break;
    case "delete":
    case "put": curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'PUT');
                curl_setopt($ch,
CURLOPT_USERPWD, "profesor:profesor");
        break;
    }
    curl_exec($ch);
    curl_close($ch);
?>

```

Gracias a este proceso intermedio, ya podremos realizar llamadas al servicio REST que comentábamos simplemente cambiando el primer parámetro de la función `$.getJSON()` por el valor `curl.php`.



Biblioteca con servicios REST

## 8. Aspectos avanzados de jQuery Mobile - Ejercicios

### 8.1. Ejercicio 1: Listado de libros mejorado

En el último ejercicio de la sesión anterior preparábamos un listado de libros que ahora vamos a mejorar. Sin perder el listado anterior, crearemos una nueva forma de listar los libros de la biblioteca siguiendo el modelo de hemos visto de listados básicos.

En este nuevo listado únicamente mostraremos el título y será una vez que el usuario haga clic o toque el elemento en cuestión cuando cargaremos los datos de dicho libro en una nueva página. Prueba también a utilizar la característica que permite buscar elementos de una lista desordenada que hemos visto en la sección de *Listados con filtros*.

Esta nueva página donde se mostrarán los datos del libro seleccionado será un listado simple de solo lectura en donde se mostrarán los datos relativos al autor, el año de edición, el isbn y la editorial.

Agregar por último un pie de página cuando el usuario está viendo los detalles de un libro de tal forma que pueda realizar la reserva del libro en cuestión y además pueda marcarlo como favorito. Añadir también una opción para que el usuario pueda ver la portada del libro en forma de cuadro de diálogo (visto en la sesión 3).

El archivo html creado en este ejercicio se debe llamar *author.html* y debe estar en la raíz de un directorio llamado *ibiblioteca*.

### 8.2. Ejercicio 2: Registro de usuarios

En nuestra aplicación biblioteca, una parte de la misma serán los alumnos y serán ellos mismos quienes se registren en la aplicación rellenando un sencillo formulario que vamos a diseñar en este ejercicio. Veamos que datos vamos a solicitarles.

En primer lugar, necesitaremos los datos típicos de cualquier registro, estos son nombre y apellidos, correo electrónico y una contraseña. Por otro lado, también le pediremos al alumno que nos indique que curso está estudiando para lo que tendrá que elegir una opción de entre las siguientes opciones:

- 1º ESO
- 2º ESO
- 3º ESO
- 4º ESO
- 1º BAT
- 2º BAT

Por último, el alumno deberá marcar que ha leído las condiciones del registro que podrá leer gracias a un enlace que cargará estas condiciones en forma de cuadro de diálogo.

Puedes descargar este archivo desde [este enlace](#).

### 8.3. Ejercicio 3: Consumiendo servicios REST

---

En la parte de teoría veíamos como podíamos generar un listado con los libros a partir de un servicio REST que nos devuelve un archivo en formato JSON. Si accedemos a la dirección del servicio REST desde un navegador cualquiera podremos ver como en lugar de generar un archivo en formato JSON obtendremos el mismo contenido pero en formato XML, quizás más legible desde nuestro punto de vista.

Si echamos un vistazo al <http://server.jtech.ua.es:8080/jbib-rest/resources/libros> comprobaremos como cada libro tiene una url para poder acceder a la información del mismo. Concretamente, el archivo que se encarga de mostrar la información de cada libro es *librodetalle.php*.

En este ejercicio completaremos la información del libro a partir de la información del servicio REST concreto para cada libro. Estos detalles serán los siguientes:

- Título
- Autor
- ISBN
- Páginas
- Imagen de la portada

Además, y en función de las opciones recibidas de cada libro en el servicio REST, colocaremos unos botones para cada una de estas opciones.

## 9. Introducción a Sencha Touch

### 9.1. Introducción

Sencha Touch es un framework para el desarrollo de aplicaciones móviles centrado en WebKit. Fue el primer framework basado en HTML5 y JavaScript, además utiliza CSS3 para realizar animaciones. La apariencia de las aplicaciones desarrolladas es similar al de las aplicaciones nativas en Android, BlackBerry e iOS. Sencha Touch está disponible tanto en versión con licencia comercial como con licencia Open Source GPL v3.



Una de las principales ventajas de Sencha Touch es la cantidad de controles IU o elementos de interfaz que incluye, todos ellos muy fáciles de usar y personalizar. Ha sido diseñado específicamente para dispositivos táctiles por lo que incluye una amplia gama de eventos táctiles o *gestures*, que comúnmente son usados en dispositivos táctiles. Además de los eventos estándares como *touchstart* o *touchend*, han añadido una extensa lista de eventos como *tap*, *double tap*, *tap and hold*, *swipe*, *rotate* o *drag and drop*.



#### Interoperabilidad con PhoneGap

Sencha Touch funciona perfectamente junto a *PhoneGap*, por lo que puede ser usado para distribuir nuestras aplicaciones en la *App Store* o en *Android Marketplace*. Se basa en el

uso de un mecanismo que empotra nuestra aplicación en una shell nativa de la forma más sencilla posible. Además, gracias a *PhoneGap* podemos hacer uso de la API nativa del dispositivo para acceder a la lista de contactos, la cámara y muchas otras opciones directamente desde JavaScript.

## Integración de datos

Al igual que con *ExtJS*, Sencha Touch implementa el patrón de diseño MVC en el lado del cliente y nos ofrece una API rica y poderosa para manejar flujos de datos desde una increíble variedad de fuentes. Podemos leer datos directamente a través de AJAX, JSON, YQL o la nueva capacidad *local storage* de HTML5. Podemos enlazar esos datos a elementos específicos de nuestras vistas, y utilizar los datos sin conexión gracias a los almacenes locales.

## 9.2. Sencha Touch vs. JQuery Mobile

---

A continuación se enumeran las principales diferencias entre Sencha Touch y JQuery Mobile:

Sencha Touch:

- Tiene una curva de aprendizaje mucho mayor y necesita una mayor comprensión del lenguaje de programación JavaScript, pero gracias a esto proporciona una API mucho más potente.
- Dispone de un mayor número de controles para la interfaz de usuario, así como efectos de transición y animaciones entre páginas, mucho más personalizables.
- Más rápido en mayor número de dispositivos móviles (en Android a partir de la versión 2.1). El comportamiento y velocidad de Sencha Touch es mucho mejor que el de otros frameworks, a excepción de el tiempo de carga inicial, pues JQuery Mobile pesa menos.
- Al estar basado en ExtJS (usan el mismo núcleo), es muy robusto y potente, además de ser un framework muy probado y usado (también debido a que fue uno de los primeros en aparecer).
- Al igual que en ExtJS, y en comparación con JQuery Mobile, se escribe mucho. Esto podría ser tomado como un pro y como un contra. Es bueno porque indica una mayor potencia de configuración y personalización, pero por contra conlleva más tiempo de desarrollo y de aprendizaje.

JQuery Mobile:

- Muy sencillo de aprender y de implementar aplicaciones.
- Es necesario escribir muy poco código (y casi no se usa JavaScript) para lograr aplicaciones móviles muy interesantes. En lugar de orientarse a la programación JavaScript, JQuery Mobile se centra en usar etiquetas HTML con atributos definidos por el framework.
- No dispone de muchos controles para el diseño de la interfaz.
- Actualmente sigue en versión beta.



- Tiene una ejecución más lenta en los dispositivos móviles. Se espera una mejora sustancial cuando salga de fase beta.
- Al estar basado en un framework muy desarrollado, como es JQuery, funciona correctamente en un mayor número de dispositivos móviles y de navegadores, como Symbian, Android, iOS, Blackberry, Window Phone 7 o WebOS.

Ambos frameworks son buenas opciones para el desarrollo de aplicaciones móviles. Los dos utilizan HTML5, JavaScript e integran la tecnología AJAX. La decisión dependerá de las necesidades de la aplicación a desarrollar. En principio, Sencha Touch es más apropiado para aplicaciones grandes, que necesiten de mayor personalización o configuración y que vayan a hacer un mayor uso del lenguaje de programación JavaScript. JQuery Mobile se suele utilizar para aplicaciones en las que se necesite una interfaz de usuario que conecte directamente con un servidor y que haga un menor uso de JavaScript.

### 9.3. Instalar Sencha Touch

---

En primer lugar descargamos el SDK de Sencha Touch desde su página Web “<http://www.sencha.com/products/touch/download/>”. Tendremos que elegir si queremos usar la licencia comercial o la Open-source, pero en ambos casos el contenido a descargar será el mismo. Se nos descargará un fichero comprimido con ZIP (llamado algo como “sencha-touch-VERSION.zip”), que descomprimiremos en una carpeta de nuestro servidor Web que renombraremos a “touch”.

También es posible guardar el SDK de Sencha Touch en una carpeta diferente y posteriormente crear un enlace simbólico (llamado “touch”) hasta esta carpeta desde cada uno de nuestros proyectos Web (ver siguiente apartado).

Una vez instalado podremos comprobar su correcto funcionamiento accediendo con nuestro navegador a la dirección “<http://localhost/touch>”, con lo que veremos el contenido de ejemplo que viene con el SDK de Sencha Touch:



Sencha Touch solo funciona con navegadores basados en WebKit, como son: Safari, Google Chrome, Epiphany, Maxthon o Midori. Si lo probamos en un navegador que no lo soporte, como Firefox o Internet Explorer, solamente veremos una página en blanco o un resultado erróneo. Por lo tanto para probar nuestros proyectos Web tendremos que instalar uno de los navegadores soportados, como Google Chrome (<http://www.google.es/chrome>) o Apple Safari (<http://www.apple.com/es/safari/>).

Aunque la mayoría de webs que podemos hacer con Sencha Touch se podrían ejecutar y visualizar directamente sin necesidad de un servidor Web, sí que será necesario su uso si queremos probar nuestros proyectos utilizando algún emulador o un dispositivo móvil real.

En la sección inicial de “Instalación de un servidor Web” se puede encontrar información sobre la instalación de un emulador móvil o la configuración para el acceso externo mediante un dispositivo móvil real.

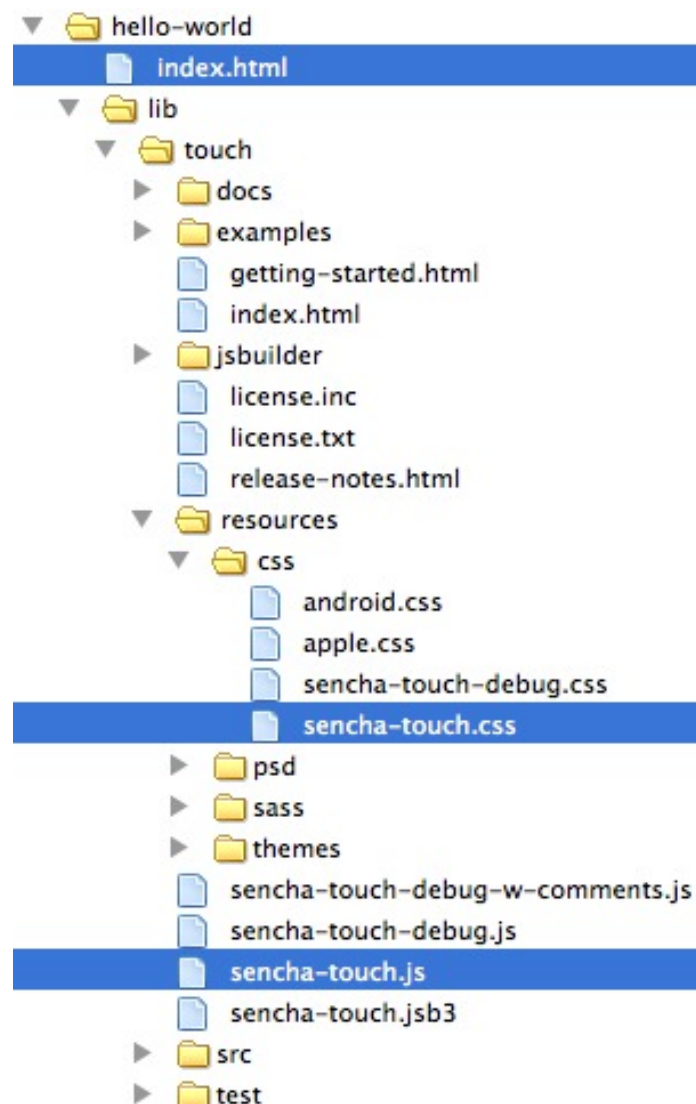
## 9.4. Estructura de carpetas

Para crear un nuevo proyecto en primer lugar tenemos que crear una carpeta dentro de nuestro servidor, por ejemplo “hello-world” como en la imagen inferior. Creamos un fichero vacío dentro de esta carpeta llamado “index.html” que será el fichero de inicio de

nuestro proyecto. Por último tenemos que copiar (o enlazar) el SDK de Sencha Touch dentro de un subdirectorio llamado “lib/touch”. Los pasos serían los siguientes:

```
$ mkdir hello-world // Nuevo proyecto
$ cd hello-world
$ touch index.html
$ mkdir lib
$ ln -s /home/code/web/SenchaSDK/sencha-touch-VERSION ./lib/touch
```

Finalmente la estructura de directorios queda como:



En la imagen se han marcado los ficheros “index.html”, “sencha-touch.css” y “sencha-touch.js”, los cuales corresponden con el fichero inicial del proyecto, la hoja de

estilo a utilizar y la librería JavaScript de Sencha Touch.

Al desplegar nuestra aplicación final no será necesario copiar todo el código de la librería de Sencha Touch, sino solamente los recursos que utilizemos.

## 9.5. Código HTML básico de una aplicación

Las aplicaciones de Sencha Touch se crean como un documento HTML5 que contiene referencias a los recursos de JavaScript y CSS. Nuestro fichero “index.html” debe de contener como mínimo el siguiente código:

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Hello World</title>

  <link href="lib/touch/resources/css/sencha-touch.css" rel="stylesheet"
        type="text/css" />

  <script src="lib/touch/sencha-touch.js" type="text/javascript"></script>
  <script src="app/app.js" type="text/javascript"></script>
</head>
<body>
</body>
</html>
```

A continuación analizaremos por separado cada una de las partes de este código:

```
<!DOCTYPE html>
<html>
...
</html>
```

La primera línea nos indica que este es un documento del tipo HTML5. Las etiquetas de `<html>` y `</html>` indican el inicio y final del documento HTML y deben de contener en su interior todo el resto del código.

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Hello World</title>
  ...
</head>
<body>
</body>
```

Todo documento HTML (y HTML5 también) debe de contener primero una sección de cabecera (`<head>`) y a continuación una sección con el contenido principal o cuerpo del documento (`<body>`). En este caso el cuerpo del documento (`<body>`) se encuentra vacío. Esto se debe a que la librería Sencha Touch crea todo el contenido de la Web, incluidos todos los elementos de la interfaz, mediante código JavaScript.

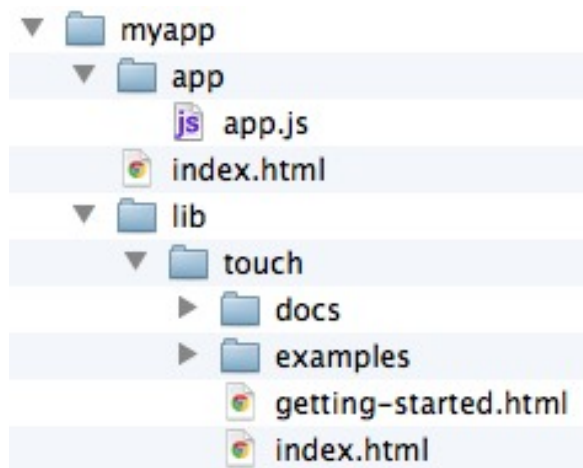
La cabecera del documento (`<head>`) debe de contener como mínimo los metadatos

acerca del tipo de contenido, el conjunto de caracteres usados, y un título que mostrará el navegador en la parte superior de la ventana. Además debe de contener los enlaces a las librerías JavaScript y a la/s hoja/s de estilo usada/s:

```
<link href="lib/touch/resources/css/sencha-touch.css" rel="stylesheet"
type="text/css" />
<script src="lib/touch/sencha-touch.js" type="text/javascript"></script>
<script src="app/app.js" type="text/javascript"></script>
```

La etiqueta `<link/>` indica la localización de la hoja de estilo. Esta hoja de estilo se puede cambiar o colocar en otra dirección diferente. La etiqueta `<script></script>` se utiliza para cargar código JavaScript en nuestra página Web. Primero se carga la librería de Sencha Touch (localizada en “lib/touch/sencha-touch.js”) y a continuación el código de nuestra aplicación (“app/app.js”, que de momento está vacío).

Finalmente la estructura de carpetas de nuestro proyecto Web debería quedar como el de la imagen siguiente:



Ahora ya tenemos cargadas las librerías de Sencha Touch y el código de nuestra aplicación para empezar a trabajar. De momento, si lo visualizamos en un navegador solo veremos una página en blanco.

### Mostrar aviso durante la carga

Mientras que se carga la librería de Sencha Touch podemos mostrar fácilmente un texto o una imagen. Para esto podemos aprovechar el cuerpo del documento (`<body>`) que hasta ahora se encontraba vacío. Todo lo que incluyamos en esta sección se visualizará únicamente durante la carga, posteriormente será ocultado por el contenido de la aplicación. En el siguiente ejemplo se ha creado un cuadro centrado en el que aparece el texto "Cargando Aplicación...".

```
<body>
```

```
<div style="margin:auto; width:220px; padding-top:100px;
font-size:16pt;">
    Cargando aplicación...
</div>
</body>
```

## 9.6. Instanciar una aplicación

Para realizar nuestro primer ejemplo vamos a crear una aplicación que muestre en pantalla el mensaje “Hello World!”, para lo cual abriremos el fichero “app.js” y añadiremos el siguiente código:

```
var App = new Ext.Application({
    name: 'MyApp',
    useLoadMask: true,
    launch: function()
    {
        MyApp.views.viewport = new Ext.Panel({
            fullscreen: true,
            html: 'Hello World!'
        });
    }
});
```

A continuación analizaremos por separado cada una de las partes de este código:

```
var App = new Ext.Application({
    name: 'MyApp',
    launch: function()
    {
        ...
    }
});
```

Con “**new Ext.Application({ ... });**” creamos una nueva instancia de Sencha Touch, es decir, este es el constructor de nuestra aplicación. Entre las llaves “{ }” le pasaremos la lista de opciones de configuración para crear nuestra aplicación. En primer lugar le damos un nombre “**name: 'MyApp'**”, con esto automáticamente se crea una variable global llamada **MyApp** junto con los siguientes *namespaces*:

- **MyApp**
- **MyApp.views**
- **MyApp.controllers**
- **MyApp.models**
- **MyApp.stores**

Estos *namespaces* (o espacios de nombres) nos permitirán acceder a atributos de nuestra aplicación de forma sencilla, los iremos viendo en detalle más adelante.

La función “**launch: function() { }**” solo se ejecuta una vez al cargar la aplicación, y es donde deberemos de colocar el código necesario para definir nuestra interfaz de usuario.

```
MyApp.views.viewport = new Ext.Panel({  
    ...  
});
```

Con “new Ext.Panel({ ....” instanciamos un panel para nuestro contenido y se lo asignamos al “viewport” de nuestra aplicación. El “**viewport**” es la vista principal de nuestra aplicación, dentro de la cual iremos añadiendo el resto del contenido.

```
fullscreen: true,  
html: 'Hello World!'
```

Lo único que hacemos es indicarle que debe ocupar toda la pantalla (fullscreen: true) y el código HTML que tiene que contener (html: 'Hello World!'). Al poner “fullscreen: true” también activamos la opción “monitorOrientation”, para que se tengan en cuenta los eventos de cambio de orientación.

Con esto ya hemos creado nuestra primera aplicación, un panel que ocupa toda la pantalla con el texto “Hello World!”.

**Nota:**

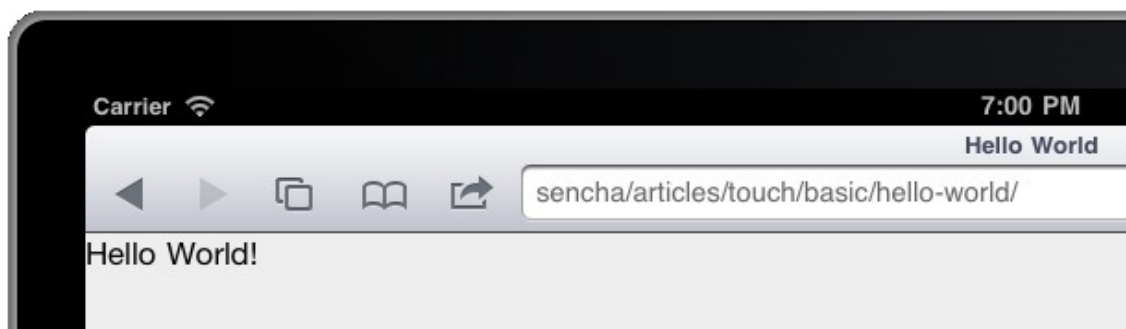
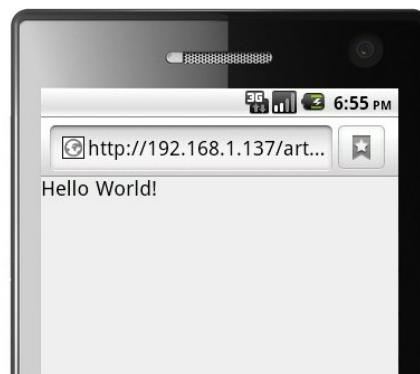
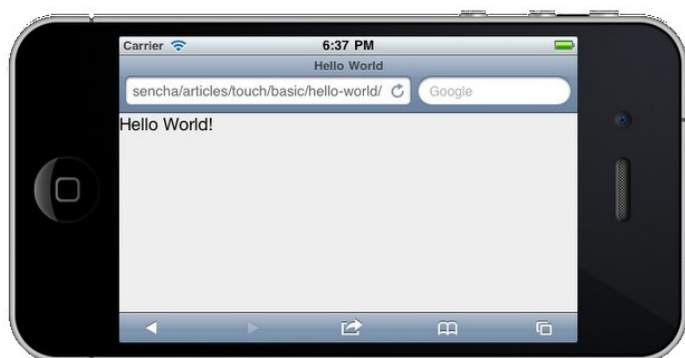
Si queremos podemos definir la función launch como una función independiente. Dentro del código del panel simplemente tendremos que poner el nombre de la función a llamar, por ejemplo: launch: crearViewport (sin poner los paréntesis de función). Y luego de forma independiente definiríamos la función function crearViewport() {} con el resto del código. Esta es una buena práctica para modularizar nuestro código.

## 9.7. Comprobando los resultados

Para comprobar el código de nuestra aplicación podemos abrirlo en un navegador compatible con WebKit, como Chrome o Safari. Si abrimos el ejemplo de la sección anterior deberíamos obtener algo como:



Si nuestro código está en un servidor también podemos comprobar el resultado utilizando un emulador de móvil, como en las imágenes inferiores:



Estos emuladores son parte del IDE de Xcode y del SDK de Android. Para poder utilizarlos necesitaremos tener nuestro código en un servidor Web. Para más información consultar la sección inicial “Instalación de un servidor Web” y Emuladores.



## 9.8. Todo en un único fichero

Para aplicaciones sencillas también es posible trabajar sin un servidor Web. Para esto enlazaremos las librerías de Sencha Touch directamente desde su SDK online (asumiendo que tenemos conexión a Internet) e incluiremos nuestro código JavaScript dentro del mismo fichero “index.html”, entre las etiquetas `<script></script>` y dentro de la sección `<head>`, como en el siguiente fragmento:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello World</title>
  <link
href="http://cdn.sencha.io/touch/1.1.0/resources/css/sencha-touch.css"
                                rel="stylesheet"
type="text/css" />
  <script src="http://cdn.sencha.io/touch/1.1.0/sencha-touch.js"
type="text/javascript"></script>

  <script type="text/javascript">
    // Nuestro código JavaScript
  </script>
</head>
<body>
</body>
</html>
```

Este fichero lo podremos colocar en cualquier carpeta de nuestro ordenador y abrir directamente con un navegador. No es la mejor forma de programar, pero nos sirve para probar cosas de forma rápida con un solo fichero.

## 9.9. Paneles y Layouts

Como ya hemos comentado, en Sencha Touch el contenido se distribuye en **paneles**, los cuales actúan de contenedores de otros elementos y nos ayudan a distribuir el contenido. En primer lugar deberemos definir el panel asignado para la vista principal, el “**viewport**”, dentro del cual iremos añadiendo el resto de contenido y paneles.

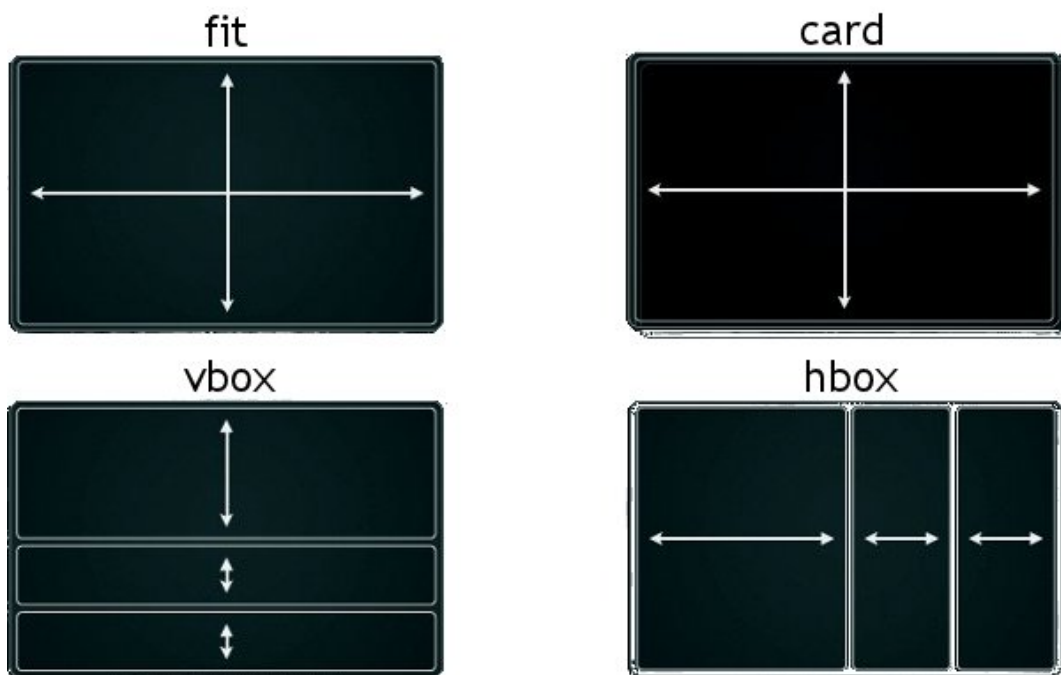
```
MyApp.views.viewport = new Ext.Panel({
  fullscreen: true,
  layout: 'fit'
});
```

Este panel lo hemos definido para que ocupe toda la pantalla (`fullscreen: true`) y para que los elementos se distribuyan en él ocupando todo el espacio disponible (`layout: 'fit'`).

Los **layouts** se aplican a los paneles y especifican como se distribuyen los objetos que estén dentro de ellos. Los layouts que incluye Sencha Touch son:

- **fit**: los elementos se distribuyen consecutivamente ocupando todo el espacio disponible.

- **card**: los elementos se colocan de forma superpuesta, uno encima de otro. Este elemento se utiliza para colocar paneles intercambiables, solo se visualizará uno a la vez, y el resto se mostrarán al realizar alguna acción.
- **vbox**: los elementos se distribuyen de forma vertical uno debajo de otro.
- **hbox**: los elementos se distribuyen de forma horizontal.



Con “**new Ext.Panel({ ... })**” creamos una instancia para un panel, dentro de este constructor se incluirán los atributos necesarios para describir su contenido. Ya hemos visto los atributos “**fullscreen: true**” y “**html: ...**”, ahora veremos que atributos tenemos que usar para definir el layout e incluir otros paneles.

Con “**layout: 'valor'**” definimos el tipo de disposición de un panel. Y usando el atributo “**items: [ valores ]**” podemos añadir elementos. La forma de referenciar estos elementos es a través del espacio de nombres: “**NombreAplicación.view.NombrePanel**” o a través de su identificador (como veremos más adelante). En el siguiente código se crean dos paneles (panelSuperior y panelInferior) que después se añaden al panel principal usando la disposición “**vbox**”:

```
MyApp.views.panelSuperior = new Ext.Panel({
    html: 'Panel Superior'
});
MyApp.views.panelInferior = new Ext.Panel({
    html: 'Panel Inferior'
});
MyApp.views.viewport = new Ext.Panel({
    fullscreen: true,
    layout: 'vbox',
    items: [
```

```

    MyApp.views.panelSuperior,
    MyApp.views.panelInferior
  ]
});

```

### Items de un panel

La propiedad “items” permite añadir elementos de diferentes formas:

- Especificar un elemento o una lista de elementos utilizando su nombre:

```

items: [elemento]
items: [elemento1, elemento2]

```

- Definir un elemento o una lista de elementos en línea:

```

items: {}
items: [{...}, {...}]

```

Al crear un elemento en línea, además de especificar el resto de sus propiedades, también tendremos que definir su tipo (xtype), de la forma:

```

items: { xtype: 'toolbar', dock: 'top' }

```

En todos los casos deberemos especificar un layout apropiado.

## 9.10. Identificadores

En todos los componentes de Sencha Touch podemos definir un identificador (id) mediante el cual posteriormente podremos hacer referencia a ese elemento. La forma de definirlo, por ejemplo, para un panel es la siguiente:

```

var panelSuperior = new Ext.Panel({
  id: 'panelSuperior',
  html: 'Panel Superior'
});

```

Posteriormente desde otro elemento podremos referirnos a este panel como 'panelSuperior', por ejemplo, para añadirlo como un ítem en otro panel:

```

MyApp.views.viewport = new Ext.Panel({
  fullscreen: true,
  layout: 'card',
  items: [ 'panelSuperior' ]

  // También podríamos haber usado su nombre de variable, de la forma:
  // items: [ panelSuperior ]
});

```

Como hemos dicho, este identificador podemos usarlo con todos los elementos: botones, barras, etc. Es una buena práctica definirlo para todos los elementos que creamos que vayamos a referenciar posteriormente. En este documento, por simplicidad, no lo incluiremos en todos los ejemplos, solamente cuando sea necesario. Cuando aparezca una

referencia a otro elemento por su nombre de variable, también sería posible hacerlo por su identificador. En algunos casos solo será posible hacerlo por su nombre de identificador, como veremos más adelante.

## 9.11. Toolbars

Añadir barras de herramientas a un panel se realiza de forma muy similar a la forma de añadir paneles a otro panel. En primer lugar tenemos que crear la barra de herramientas, para esto utilizamos el constructor “**new Ext.Toolbar({ ... })**”. Dentro de este constructor incluiremos los atributos necesarios para describir su contenido.

Con “**doc: 'top'**” o “**doc: 'bottom'**” indicamos que la barra se coloque en la parte superior o en la parte inferior. Con el atributo “**title: 'texto'**” podemos indicar un texto que se colocará en el centro de la barra.

Para añadir estas barras a un panel utilizamos el atributo “**dockedItems: [ ... ]**”, referenciando cada barra a través del espacio de nombres (“NombreAplicación.view.NombreBarra”) o a través de su identificador.

En el siguiente ejemplo se crean dos barras de herramientas (topToolbar y bottomToolbar) y después se añaden a un panel usando para una el nombre de la variable y para otra su identificador:

```
MyApp.views.topToolbar = new Ext.Toolbar({
  dock: 'top',
  title: 'Top Toolbar'
});
var bottomToolbar = new Ext.Toolbar({
  id: 'bottomToolbar',
  dock: 'bottom',
  title: 'Bottom Toolbar'
});
MyApp.views.viewport = new Ext.Panel({
  fullscreen: true,
  layout: 'fit',
  html: 'Contenido central',
  dockedItems: [
    MyApp.views.topToolbar,           // Nos referimos al elemento a través
    de su variable                    // Usamos su identificador
    'bottomToolbar'
  ]
});
```

Con lo que obtendríamos un resultado similar a:



Un atributo opcional que podemos usar es “**ui: valor**” para cambiar la apariencia de la barra. Por defecto toma el valor “dark”, pero también podemos usar “light” que aplicará unos colores más claros.

## 9.12. Docked items

Como ya hemos visto en la sección anterior, **dockedItems** se utiliza para especificar uno o más elementos que serán anclados a una parte del panel. Los elementos que se suelen anclar usando esta propiedad son: **Toolbar** y **TabBar** (como veremos más adelante). Y las posiciones que podemos usar son: **top**, **bottom**, **left**, **right**.

Los elementos se pueden añadir definiéndolos en línea (hay que tener en cuenta que para este caso tenemos que añadir su *xtype*):

```
dockedItems: [  
  {  
    xtype: 'toolbar',  
    dock: 'top',  
    title: 'barra superior'  
  }, {  
    xtype: 'toolbar',  
    dock: 'bottom',
```

```

    title: 'barra inferior'
  }
];

```

Pero por cuestiones de claridad será mejor crear una definición separada y añadirlos posteriormente, como hemos visto en la sección anterior:

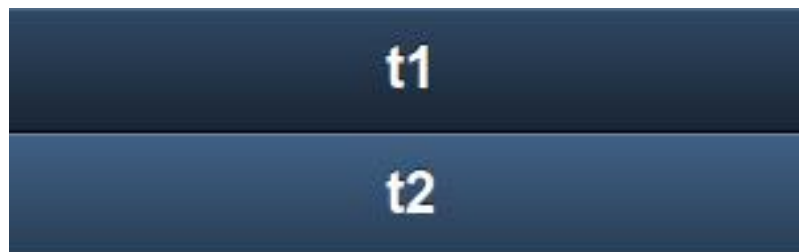
```

var toolbar1 = new Ext.Toolbar({ dock: 'top', title: 't1' });
var toolbar2 = new Ext.Toolbar({ dock: 'top', title: 't2' });

MyApp.views.viewport = new Ext.Panel({
  ...
  dockedItems: [ toolbar1, toolbar2 ]
});

```

En este ejemplo las dos barras de herramientas se colocan en la parte superior del panel. Las barras se irán colocando una debajo de otra según el orden en el que se añaden.



También es posible manipular los elementos anclados mediante las funciones:

- `addDocked( item/array )`: ancla un nuevo elemento al objeto desde el que se referencia, por ejemplo: `panel.addDocked( bottomToolbar )`;
- `removeDocked( item )`: elimina un elemento anclado.
- `getDockedItems()`: devuelve un array con los elementos anclados.

### 9.13. Botones

Los botones se añaden a las barras de herramientas (Toolbar) mediante su propiedad “items”. Simplemente tendremos que definir el tipo de botón (ui) y el texto a mostrar (text). En el siguiente ejemplo se crea una barra de herramientas con un botón y posteriormente se añade al panel principal.

```

var topToolbar = new Ext.Toolbar({
  dock: 'top',
  title: 'mi barra',
  items: [
    { ui: 'action', text: 'Nuevo' }
  ]
});

MyApp.views.viewport = new Ext.Panel({
  fullscreen: true,
  layout: 'fit',
  dockedItems: [ topToolbar ]
});

```

```
} ) ;
```

Podemos usar siete tipos predefinidos de botones, estos son:

- ui: 'back'
- ui: 'forward'
- ui: 'normal'
- ui: 'round'
- ui: 'action'
- ui: 'confirm'
- ui: 'decline'



Además podemos usar los modificadores “-small” y “-round” sobre los tipos “action”, “confirm” y “decline” para obtener botones más pequeños o redondeados:



Si no seleccionamos un tipo (ui), por defecto nos aparecerá el botón tipo “normal”.

Si queremos variar el **ancho** de un botón podemos utilizar la propiedad “width: '200px'” en píxeles o “width: '95%'” indicando porcentajes.

## Iconos

También podemos usar algunos iconos predefinidos, indicando el nombre del icono mediante la propiedad “iconCls: 'nombre’” y activando “iconMask: true” para que el icono se visualice correctamente, de la forma:

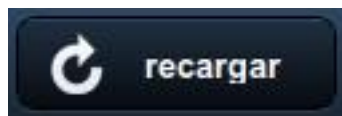
```
var button = new Ext.Button({
    iconCls: 'action',
    iconMask: true
});
```

Los iconos que podemos utilizar son:

 action	 add	 arrow_down
		

arrow_left	arrow_right	arrow_up
		
compose	delete	organize
		
refresh	reply	search
		
settings	star	trash
		
maps	locate	home

Si además usamos la propiedad “text: 'texto'” al definir el botón, el texto aparecerá a la derecha del icono:



Opcionalmente podemos aplicar colores a estos iconos, aplicándole los tipos (ui) de 'action', 'decline' o 'confirm', obteniendo:



### Imágenes externas

Si queremos usar una imagen externa tenemos que aplicar al botón un estilo CSS. El botón lo definimos de forma normal, pero utilizamos su propiedad `cls` para indicarle el nombre del estilo:

```
items: [ { ui: 'normal', cls: 'btnAyuda' } ]
```

El estilo "btnAyuda" lo tendremos que definir indicando la imagen de fondo a usar junto con el ancho y el alto del botón. El tamaño de la imagen que usemos deberá de coincidir con el tamaño aplicado al botón para que no se vea recortado. El tamaño habitual de un botón es de 45x35 píxeles. Además es muy importante añadir la propiedad `!important` al

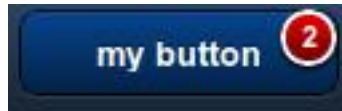


cargar la imagen de fondo. Esto es debido a que Sencha Touch sobrescribe algunos estilos, y tenemos que aplicar esta propiedad para que prevalezca el nuestro:

```
.btnAyuda {
    background: url(imgs/ayuda.png) !important;
    width: 45px;
    height: 35px;
}
```

## Badge

De forma muy sencilla podemos añadir una insignia distintiva a los botones para destacar alguna información. Para esto utilizamos la propiedad “badgeText: '2'”, que daría como resultado:



## Alineaciones

Por defecto los botones salen alineados a la izquierda. Para crear otras alineaciones utilizaremos un espaciador “{ xtype: 'spacer' }”. En el siguiente código podemos ver diferentes ejemplos de alineaciones:

```
// Alineación derecha
items: [
    { xtype: 'spacer' },
    { ui: 'normal', text: 'Botón' }
]
// Alineación centrada
items: [
    { xtype: 'spacer' },
    { ui: 'normal', text: 'Botón' },
    { xtype: 'spacer' }
]
```

## Acciones

Para añadir acciones a los botones tenemos que definir su propiedad “**handler**”, a la cual le asignaremos una función. Esta función la podemos definir en línea, de la forma `handler: function () { ... }`, o creando una función independiente para separar mejor el código, como en el ejemplo:

```
function botonPresionado(btn, evt) {
    alert("Botón '" + btn.text + "' presionado.");
}

var topToolbar = new Ext.Toolbar({
    items: [
        {
            ui: 'normal',
            text: 'Botón 1',
            handler: botonPresionado
        }
    ]
});
```

```

        },
        {
            ui: 'action',
            text: 'Botón 2',
            handler: function(btn, evt) {
                alert("Botón '" + btn.text + "'
presionado.");
            }
        }
    ]
});

```

## 9.14. Transiciones de cambio de vista

En este apartado vamos a ver como cambiar entre diferentes paneles. Nuestra aplicación tendrá una vista principal y al apretar sobre algún botón, cambiaremos a una vista o panel diferente. Para hacer esto lo más importante es utilizar un panel base no visible (asignado al “viewport”), el cual contendrá como “items” los paneles entre los que queremos cambiar. Además tenemos que establecer el layout del panel base a “layout: 'card'”, quedando el código de nuestro “viewport” de la forma:

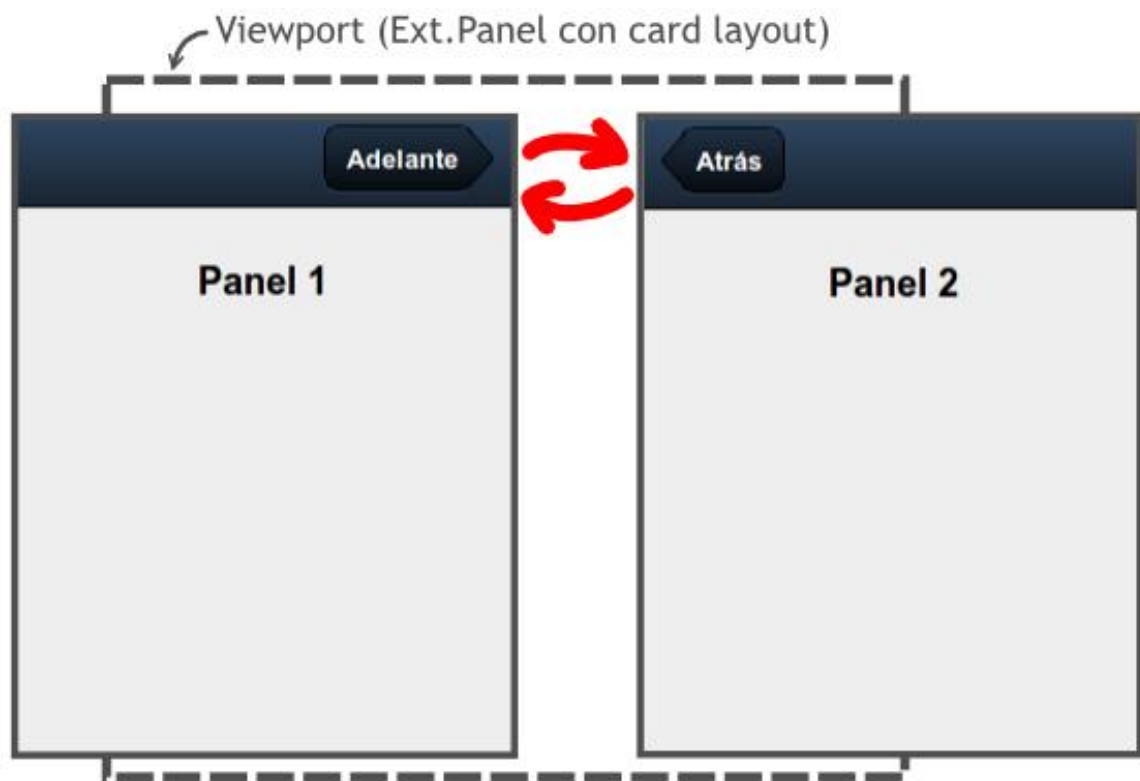
```

MyApp.views.viewport = new Ext.Panel({
    fullscreen: true,
    layout: 'card',
    items: [
        'panel1',
        'panel2'
    ]
});

```

El panel que se verá al principio es el primero que se añade a la lista de ítems, quedando el otro (o los otros) ocultos. También es muy importante fijarse que estamos **referenciando los paneles por su identificador**, y no directamente por la variable, de no hacerlo así se podrían producir comportamientos inesperados en las animaciones.

En la siguiente imagen se puede ver un esquema del intercambio de paneles (en nuestro ejemplo con dos paneles). El panel asignado al “viewport” queda como un contenedor invisible por detrás, y mediante un botón podemos pasar de un panel a otro:



A continuación se incluye el código para el “panel1”. Un simple panel con una barra de herramientas en la parte superior que contiene un botón. Lo más importante aquí es la función “handler” del botón, en la cual llamamos a `MyApp.views.viewport.setActiveItem` para cambiar al “panel2” utilizando una animación “slide”. El código para el “panel2” sería exactamente igual, pero intercambiando los identificadores.

```
var panell = new Ext.Panel({
    id: 'panell',
    fullscreen: true,
    layout: 'fit',
    html: 'Panel 1',
    dockedItems: [
        {
            xtype: 'toolbar',
            items: [
                {
                    ui: 'forward',
                    text: 'Adelante',
                    handler: function() {
                        MyApp.views.viewport.setActiveItem(
                            'panel2',
                            {type: 'slide', direction:
                                'up', duration: 2000});
                    }
                }
            ] // end items
        }
    ]
});
```

```
    } // end dockedItems
  });
```

### Método setActiveItem

La función `setActiveItem( panelID, animación )` solo está definida para los paneles que tienen un layout tipo “card”. Permite cambiar entre el panel activo o visible por otro panel indicado utilizando una animación. El primer argumento de esta función es el identificador del panel que queremos colocar en primer plano. Y el segundo argumento es el tipo de animación utilizada para intercambiar los paneles. Los tipos de animaciones que podemos utilizar son:

- *fade*: difumina el panel actual, fundiéndolo con el panel de destino, hasta completar la transición.
- *pop*: realiza una especie de animación 3D. Escala el panel actual minimizándolo hasta ocultarlo, mientras que aumenta el tamaño del panel a visualizar.
- *slide*: realiza un desplazamiento para intercambiar un panel por otro, podemos indicar una dirección: left, right, up, down (por ejemplo: `direction: 'left'`).
- *flip*: realiza una animación 3D para intercambiar los paneles.
- *cube*: realiza una animación 3D para intercambiar los paneles.
- *wipe*: realiza un barrido para el intercambio. No funciona correctamente.

Para todos ellos podemos definir una duración en milisegundos (por ejemplo “`duration: 2000`”). Las animaciones “flip”, “cube” y “wipe” no funcionan correctamente en Android.

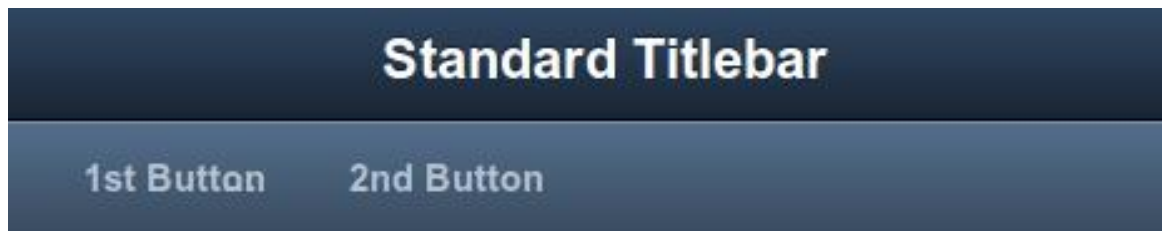
## 9.15. TabBar

Los TabBar son muy similares a las barras de herramientas, las cuales también se añaden a un panel anclándolas mediante “`dockedItems`”. La apariencia visual es diferente, con una superficie más plana (con menos sombra que las ToolBar). Se usan para crear botones de texto más grandes (o áreas pulsables), los cuales no tendrán aspecto de botón sino que formarán parte de la barra. Por esta razón los botones no admiten la propiedad “`ui`”, pero sí que la podemos usar para cambiar la apariencia de la barra (`ui: 'dark'` o `ui: 'light'`). Además estas barras tampoco soportan el atributo “`title`”.

En el siguiente ejemplo se ha añadido un TabBar en la parte de abajo de un ToolBar (solo se incluye el código del TabBar):

```
var bar = new Ext.TabBar({
  dock : 'top',
  ui   : 'light',
  items: [
    { text: '1st Button' },
    { text: '2nd Button' }
  ]
});
```

Con lo que obtendríamos un resultado similar al de la siguiente imagen:



También podemos usar la propiedad “**width**” para definir el ancho de los “tabs” en píxeles o en porcentaje, por ejemplo: “width: '50%'”.

Para añadir **acciones** a los tabs tenemos que definir su propiedad “handler”, asignándole una función, de la forma:

```
handler: function() {
    alert("Tab presionado");
}
```

## 9.16. Carousel

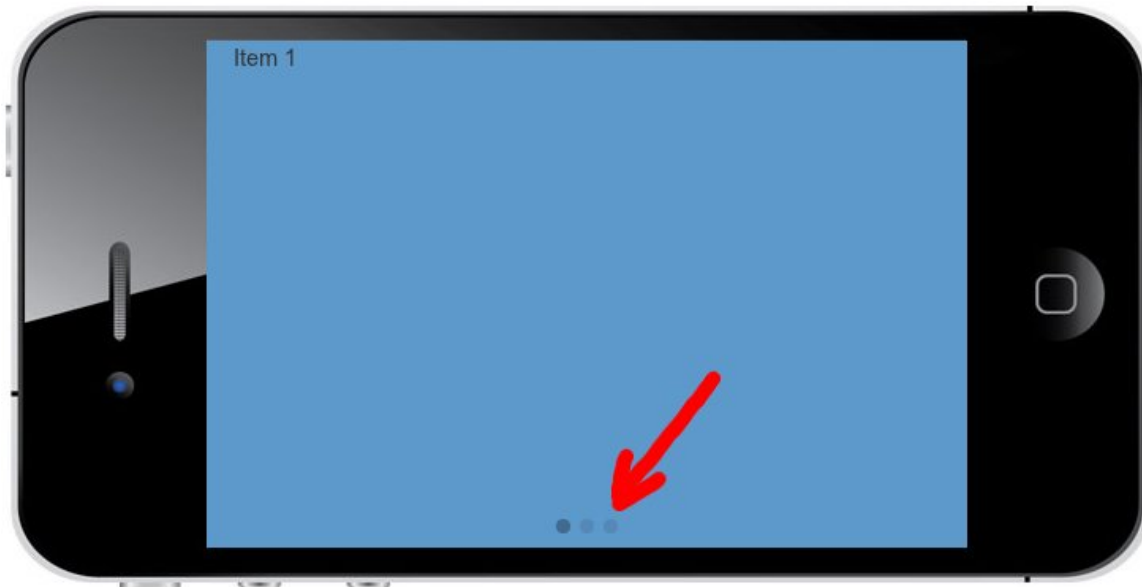
El Carousel es un contenedor de paneles que nos permite cambiar entre ellos simplemente arrastrando el dedo. Solo se muestra un panel en cada momento, además de un pequeño indicador con puntos que muestra el número de paneles disponibles. Es muy sencillo configurarlo, en la sección `items` tenemos que definir cada uno de los paneles, además si queremos que se utilicen los estilos HTML básicos tenemos que activar la opción `defaults: { styleHtmlContent: true }`, como en el siguiente ejemplo:

```
var panelCarousel = new Ext.Carousel({
    fullscreen: true,

    defaults: {
        styleHtmlContent: true
    },

    items: [
        {
            html : 'Item 1',
            style: 'background-color: #5E99CC'
        },
        {
            html : 'Item 2',
            style: 'background-color: #759E60'
        },
        {
            html : 'Item 3'
        }
    ]
});
```

Con lo que obtendríamos un resultado como el siguiente:



Una opción interesante de configuración es la orientación del panel, que básicamente lo que hace es cambiar la posición de los puntos y la dirección de movimiento de los paneles. Para configurarlo usamos la propiedad `direction: 'vertical'` (por defecto) o `direction: 'horizontal'`.

### 9.17. MessageBox

Esta clase nos permite generar mensajes emergentes de tres tipos: alertas, confirmación y de campo de texto:

#### ALERTAS

Muestra un mensaje de aviso con un solo botón OK, como podemos ver en la imagen siguiente



Para crear una ventana de aviso usamos el constructor `Ext.Msg.alert()`, el primer parámetro es el título de la ventana, el segundo parámetro es el mensaje de aviso que

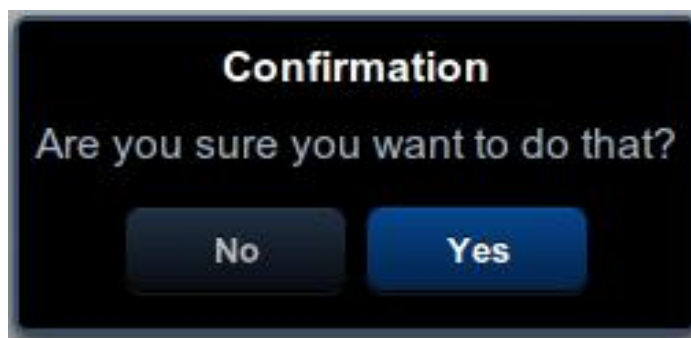
aparecerá en el centro de la ventana, y el último es la función *callback* que se llamará una vez cerrada la ventana.

```
Ext.Msg.alert('Titulo', 'Mensaje de aviso', Ext.emptyFn);
```

En este caso usamos la función vacía `Ext.emptyFn` para que no se ejecute nada. En su lugar podríamos haber puesto directamente el nombre de una función.

## CONFIRM

Este mensaje de aviso nos da la opción de aceptar o rechazar, como podemos ver en la siguiente imagen:



En este caso utilizamos el constructor `Ext.Msg.confirm()`, los parámetros serán los mismos: título, mensaje y función. En este caso sí que nos interesa indicar el nombre de una función para poder comprobar si se ha pulsado el OK. La función *callback* recibirá un único parámetro cuyo valor será el texto del botón pulsado.

```
function myFunction(btn)
{
    if( btn == "yes" )
        alert( "Confirmed!" );
}

Ext.Msg.confirm( "Confirmation", "Are you sure you want to do that?",
myFunction );
```

## PROMPT

El mensaje de campo de texto sirve para solicitar un dato al usuario, consiste en una pequeña ventana con un campo de texto que se puede aceptar o rechazar:



Utilizamos el constructor `Ext.Msg.prompt()` con los parámetros: título, mensaje y función. En este caso la función *callback* recibirá dos parámetros, el botón pulsado y el texto introducido.

```
Ext.Msg.prompt('Name', 'Please enter your name:', function(btn, text)
{
    alert( btn + ' ' + text );
});
```



## 10. Ejercicios - Introducción a Sencha Touch

En la sección de ejercicios de Sencha Touch vamos a practicar creando una pequeña aplicación móvil que nos permita gestionar un listado de notas. La pantalla principal contendrá el listado de notas, con botones que nos permitirán crear nuevas notas y modificarlas. Al crear una nueva nota nos aparecerá un formulario en el que podremos introducir los datos y añadirlos al listado. Este mismo formulario también se utilizará para modificar las notas existentes. Además desde la pantalla principal podremos abrir la ayuda con información sobre la aplicación y los autores.

Si necesitas ayuda puedes descargar la plantilla para los ejercicios [sesion05-ejercicios.zip](http://sencha.com/examples/touch/2.0.0/sesion05-ejercicios.zip). Al descomprimirlo aparecerán tres carpetas para cada uno de los ejercicios.

### 10.1. Ejercicio 1 - Estructura de la aplicación y creación de paneles

En este primer ejercicio vamos a empezar creando los ficheros que vamos a necesitar: *index.html*, *app.js* y *app.css*. Por simplicidad los vamos a colocar en el mismo directorio. El contenido de cada uno de los ficheros debe de ser el siguiente:

En *index.html* colocaremos el código necesario para cargar la librería y la hoja de estilo de Sencha Touch, junto con nuestro fichero de estilo *app.css* y nuestro fichero de código *app.js* (ver sección "Código HTML básico de una aplicación" del manual). Además, en el cuerpo del documento, añadiremos una capa con el texto "Cargando Mis Notas..." y le asignaremos el estilo "loading" que después definiremos en la hoja de estilo.

En *app.css* de momento solo vamos a definir el estilo de la clase *loading*, que lo definiremos como un texto de 16pt, en color gris, centrado en la página y a 100px del borde superior.

En *app.js* es donde vamos a crear nuestra aplicación y los paneles que necesitamos. Como ya hemos explicado vamos a tener un panel para la lista de notas, otro para editar las notas y un tercero para mostrar la ayuda. Además vamos a necesitar un cuarto panel que se utilizará como contenedor. A continuación se explican los pasos que debemos seguir:

1. En primer lugar creamos la instancia de la aplicación a la que pondremos como nombre *MisNotas*. Aquí deberemos definir también la función *launch*, a la que asignaremos una función llamada *crearViewPort*. Para ayuda consultar la sección "Instanciar una aplicación".
2. Definimos la función *crearViewPort* como una función independiente, dentro de la cual vamos a crear el panel principal y asignarlo al *viewport*. Este panel ocupará toda la pantalla y tendrá un *layout* tipo *card*. Además en su sección *items* tenemos que incluir los tres paneles de la aplicación usando su identificador, a los que llamaremos 'panelContenedorLista', 'panelFormulario' y 'panelAyuda'.
3. Los paneles 'panelContenedorLista' y 'panelFormulario' los definiremos de la misma forma, un panel, con el identificador apropiado, un *layout* tipo *fit* y un texto *html* de

prueba que utilizaremos para comprobar que el panel se visualiza correctamente.

4. Para el panel de ayuda vamos a utilizar un *Carousel*. De momento solo lo crearemos con el identificador 'panelAyuda' y dos paneles (*items*) que contengan respectivamente en su sección de código HTML los textos "panel ayuda 1" y "panel ayuda 2".

Por último deberemos comprobar que todo funciona correctamente. Para comprobar cada uno de los paneles recordad que tenemos que cambiar su orden en la sección *items* del viewport, pues solo se visualizará el primero de ellos.

## 10.2. Ejercicio 2 - Barras de herramientas y contenido de la ayuda

Este ejercicio continua con el anterior, al cual vamos a añadir las barras de herramientas necesarias para cada panel y el contenido del panel de ayuda.

Empezaremos añadiendo las barras de herramientas, las cuales las instanciaremos como un objeto separado y después las añadiremos a los paneles usando su identificador:

- En el panel 'panelContenedorLista' añadiremos una barra en la parte superior con el título "Mis Notas" y el identificador 'panelListaToolbar'.
- En el panel 'panelFormulario' añadiremos dos barras de herramientas. Una en la parte superior con el título "Editar nota" y el identificador 'panelFormularioTopToolbar'. Y una segunda barra en la parte inferior con el identificador 'panelFormularioBottomToolbar' y sin ningún título.
- Por último para el panel 'panelAyuda' añadiremos una barra en su parte superior con el título "Ayuda" y el identificador 'panelAyudaToolbar'.

A continuación vamos a añadir el contenido del panel ayuda, el cual lo habíamos creado como un Carousel. Para separar mejor nuestro código, el contenido HTML de cada uno de los paneles del Carousel lo definiremos en una variable independiente (llamadas `var htmlAyuda1` y `var htmlAyuda2`), que posteriormente asignaremos al elemento `html` del panel correspondiente. Recordad que para concatenar cadenas tendremos que usar el símbolo más (+).

Para el HTML del primer panel del Carousel definiremos una capa a la que asignaremos el estilo "ayuda". Dentro de esta capa colocaremos el título "Mis Notas" (de tipo H1), seguido por un par de párrafos con el texto "Aplicación Web para la gestión de notas realizada con **Sencha Touch**. Especialista Universitario en Desarrollo de Aplicaciones para Dispositivos Móviles". Y por último abajo colocaremos una imagen con un ancho de 100 píxeles (subcarpeta "imgs" de la plantilla).

Para el segundo panel utilizaremos también una capa de la clase "ayuda", en la que colocaremos el titular "Autor", seguido por unos párrafos con los datos del autor.

Para dar formato a este texto tendremos que modificar el fichero *app.css* añadiendo los siguientes estilos:

- *ayuda*: la clase ayuda la definiremos con un ancho del 100% y un texto centrado.

- *ayuda h1*: la etiqueta H1 (cuando esté dentro de la clase "ayuda") la definiremos con un color azul oscuro (navy), un tamaño de 18 puntos, y el estilo "uppercase" para que aparezca siempre en mayúsculas.
- *ayuda p*: la etiqueta P la definiremos para que tenga color gris (gray).

### 10.3. Ejercicio 3 - Botones y Transiciones

En este último ejercicio vamos a añadir los botones de cada una de las barras de herramientas. Además también incluiremos las acciones de transición que nos permitirán movernos entre paneles. Recordad que para añadir botones a una barra de herramientas tenemos que aplicarles el `layout: 'hbox'`, y que los botones irán definidos en su sección `items`. Además todas las transiciones que apliquemos serán del tipo 'slide' y con una duración de 1 segundo (1000ms).

Por simplicidad empezaremos definiendo el botón del panel 'panelAyuda'. En su barra de herramientas vamos a poner un único botón, alineado a la izquierda, en el que colocaremos el icono predefinido de tipo 'home'. Además al presionar este botón realizaremos una transición hacia arriba para cambiar al panel 'panelContenedorLista'. Para más información ver la sección "Transiciones de cambio de vista".

En el panel principal ('panelContenedorLista') vamos a colocar dos botones en su barra de herramientas. El primer botón estará alineado a la izquierda y contendrá una imagen externa (subcarpeta "imgs" de la plantilla). Para añadir esta imagen tendremos que asignar un estilo al botón, llamado "btnAyuda" (ver sección "Botones"). Este estilo lo definiremos en nuestra hoja de estilo "app.css" para que cargue la imagen indicada (recordad usar `!important`) y que tenga un ancho de 45 píxeles y un alto de 35 píxeles. Para este botón definiremos una transición hacia abajo que cargue el panel 'panelAyuda'.

El segundo botón del panel 'panelContenedorLista' estará alineado a la derecha, con el tipo 'action' y el texto "Nueva Nota". Este botón cargará el panel 'panelFormulario' con una transición hacia la izquierda.

Por último en el panel 'panelFormulario' teníamos dos barras de herramientas. En la barra superior colocaremos dos botones. El primero de ellos será similar al panel de ayuda, un botón alineado a la izquierda, con el icono predefinido de tipo 'home'. El segundo botón de esta barra estará alineado a la derecha, será de tipo 'action' con el texto "Guardar". El tercer botón estará en la barra inferior alineado a la derecha y contendrá el icono predefinido de tipo 'trash'. **Los tres botones** de este panel realizarán la misma transición hacia la derecha para cambiar al panel 'panelContenedorLista'.

## 11. Aspectos avanzados de Sencha Touch

### 11.1. Data Model

Los modelos de datos nos permiten representar objetos, junto con sus validaciones de formato y las relaciones que estos tienen con otros objetos. Por comparación podríamos pensar que un Data Model es como una clase con la definición de los datos que la componen y funciones para poder validar y trabajar con esos datos.

Para usarlos tenemos que crear una instancia de “**Ext.regModel**”, en la que primero se indica el nombre del modelo o identificador (en el ejemplo siguiente sería “Users”), y a continuación los campos que lo componen (fields):

```
Ext.regModel('Users', {
  idProperty: 'id',
  fields: [
    { name: 'id', type: 'int' },
    { name: 'usuario', type: 'string' },
    { name: 'nombre', type: 'string' },
    { name: 'genero', type: 'string' },
    { name: 'activo', type: 'boolean', defaultValue: true },
    { name: 'fechaRegistro', type: 'date', dateFormat: 'c' },
  ]
});
```

También es importante indicar el campo que se va a utilizar como identificador único (idProperty: 'id'). Es recomendable indicarlo siempre porque para algunos tipos de almacenamiento es necesario.

Como se puede ver en el ejemplo, los campos de la sección fields se definen primero por su nombre (name) y después por su tipo (type), que puede ser básicamente de cinco tipos: string, int, float, boolean, date. Además podemos usar algunos atributos opcionales, como el valor por defecto (defaultValue) o el tipo de fecha (dateFormat, ver [“http://docs.sencha.com/touch/1-1/#/api/Date”](http://docs.sencha.com/touch/1-1/#/api/Date)).

### Validaciones

Los modelos de datos incluyen soporte para realizar validaciones, las cuales las deberemos de incluir dentro de la misma clase regModel, a continuación del campo “fields”. Continuando con el ejemplo anterior:

```
fields: [ ... ],
validations: [
  { field: 'id', type: 'presence' },
  { field: 'usuario', type: 'exclusion', list: ['Admin', 'Administrador'] },
  { field: 'usuario', type: 'format', matcher: /([a-z]+)[0-9]{2,3}/ },
  { field: 'nombre', type: 'length', min: 2 },
  { field: 'genero', type: 'inclusion', list: ['Masculino', 'Femenino'] }
]
```

Otra opción interesante es cambiar el mensaje de error que se produciría si una validación no es correcta, esto lo podemos hacer definiendo el valor de la propiedad *message*:

```
{ field: 'titulo', type: 'presence', message: 'Por favor, introduzca un título' }
```

Cuando trabajemos con un formulario y queramos comprobar estas validaciones lo tendremos que hacer manualmente llamando a la función `validate()`, esta opción la veremos más en detalle en la sección de formularios.

### Crear de un registro

Para crear registros utilizamos el constructor “`Ext.ModelMgr.create`”:

```
var instance = Ext.ModelMgr.create({
  id: 1,
  usuario: 'javierGallego',
  nombre: 'Antonio Javier Gallego Sánchez',
  genero: 'Male',
  fechaRegistro: new Date()
}, 'Users');
```

### Más información

Para más información sobre validaciones: “<http://docs.sencha.com/touch/1-1/#!/api/Ext.data.validations>”.

Los modelos de datos también soportan la creación de relaciones con otros modelos de datos, del tipo “has-many” y “belongs-to”. Para más información consultar la API: “<http://docs.sencha.com/touch/1-1/#!/api/Ext.data.Model>”.

## 11.2. Data Store

Los almacenes de datos se utilizan para encapsular los datos (de un modelo dado) en la caché del cliente. Estos datos se cargan y escriben utilizando un proxy. Además disponen de funciones para ordenar, filtrar y consultar los datos. Crear un almacén de datos es fácil, simplemente tenemos que indicarle el modelo de datos a usar y el proxy para cargar y guardar los datos. Dependiendo del proxy que utilicemos podremos almacenar los datos de forma local o de forma remota:

### Almacenamiento en local

Para este tipo de almacenamiento, el proxy utiliza la nueva funcionalidad de HTML5 de almacenamiento en local. Esta opción es muy útil para almacenar información sin la necesidad de utilizar un servidor. Sin embargo solo podremos guardar pares clave-valor, no soporta objetos complejos como JSON. Al usar almacenamiento en local es muy importante que el modelo de datos tenga un “id” único, que por defecto tendrá ese nombre de campo (id); en caso de utilizar otro lo podríamos indicar mediante la

propiedad “idProperty: 'myID’”.

Para indicar que el proxy ha de utilizar el almacenamiento en local simplemente tendremos que indicar el tipo (*localStorage*) y un identificador (usado como clave para guardar los datos). En el siguiente ejemplo se utiliza el modelo de datos “Users” que hemos creado en la sección anterior:

```
var myStore = new Ext.data.Store({
    model: 'Users',
    autoLoad: true,
    proxy: {
        type: 'localStorage',
        id: 'notes-app-localstore'
    }
});
```

Además hemos añadido la propiedad `autoLoad: true`, muy importante para que se carguen los datos al inicio. Si no lo hiciéramos así el Store inicialmente estaría vacío.

Una forma alternativa de instanciar un Store es: `Ext.regStore( 'myStore', { ... } )`; de forma similar a como lo hacíamos con los Data Models.

### Almacenamiento en remoto

Para configurar un proxy para que utilice **JSON** lo podemos hacer de la forma:

```
var myStore = new Ext.data.Store({
    model: 'Users',
    autoLoad: true,
    proxy: {
        type: 'ajax',
        url : '/users.json',
        reader: {
            type: 'json',
            root: 'users'
        }
    }
});
```

En este ejemplo el proxy utiliza AJAX para cargar los datos desde la dirección “./users.json”. La propiedad “reader” indica al proxy como debe decodificar la respuesta que obtiene del servidor a través de la consulta AJAX. En este caso espera que devuelva un objeto JSON que contiene un array de objetos bajo la clave 'users'. Para más información sobre JSON: <http://docs.sencha.com/touch/1-1/#!/api/Ext.data.JsonReader>

### Añadir datos

Es muy sencillo añadir datos directamente a un Store, solo tenemos insertarlos como un array a través de la propiedad “data”. Suponiendo que el modelo “Users” solo tuviera dos campos (firstName, lastName), podríamos añadir datos de la forma:

```
var myStore = new Ext.data.Store({
    model: 'Users',
```

```

data : [
    {firstName: 'Ed',    lastName: 'Spencer'},
    {firstName: 'Tommy', lastName: 'Maintz'},
    {firstName: 'Aaron', lastName: 'Conran'},
    {firstName: 'Jamie', lastName: 'Avins'}
];

```

O también podemos añadir datos posteriormente llamando a la función “**add**” del objeto:

```

myStore.add({firstName: 'Ed',    lastName: 'Spencer'},
            {firstName: 'Tommy', lastName: 'Maintz'});

```

### Ordenar y Filtrar elementos

Para ordenar y filtrar los datos usamos las propiedades “sorters” y “filters”. En el siguiente ejemplo se ordenan los datos de forma descendente por nombre de usuario (también podría ser ASC) y se realiza un filtrado por género (los filtros también admiten expresiones regulares).

```

var myStore = new Ext.data.Store({
    model: 'Users',
    sorters: [
        { property: 'usuario', direction: 'DESC' }
    ],
    filters: [
        { property: 'genero', value: 'Femenino' }
    ]
});

```

### Buscar registros

En algunos casos antes de añadir un registro será necesario comprobar si el registro está repetido. Para esto podemos utilizar el método `findRecord()` del *Store*. En el ejemplo se compara el campo *id* de los datos del *Store*, con el campo *id* del registro a añadir:

```

if (myStore.findRecord('id', registro.data.id) === null)
{
    myStore.add( registro );
}

```

Otra opción para buscar registros es la función `find(campo, valor)` la cual devuelve el índice del registro encontrado, y posteriormente podríamos llamar a `getAt(index)` para obtener los datos.

### Eliminar registros

Para eliminar un registro de un *Store* usaremos la función `remove(registro)`, por ejemplo:

```

myStore.remove( registro );

```

Es recomendable comprobar si existe el registro a eliminar, para esto usaremos la función

`findRecord()`. Normalmente el *Store* estará asignado a algún panel que nos permita ver los datos (como un listado). Si quisiésemos eliminar un registro de este listado, primero tendríamos que obtener el *Store* usado, a continuación comprobar si existe el registro y si es así eliminarlo, y por último sincronizar los datos para que se visualicen, de la forma:

```
var store = miListado.getStore();

if( store.findRecord('id', registro.data.id) )
{
    store.remove( registro );
}

store.sync();
miListado.refresh();
```

### Más información

Para más información sobre los almacenes de datos consultar: [“http://docs.sencha.com/touch/1-1/#!/api/Ext.data.Store”](http://docs.sencha.com/touch/1-1/#!/api/Ext.data.Store).

## 11.3. Plantillas

Las plantillas se utilizan para describir la disposición y la apariencia visual de los datos de nuestra aplicación. Nos proporcionan funcionalidad avanzada para poder procesarlos y darles formato, como: auto-procesado de arrays, condiciones, operaciones matemáticas, ejecución de código en línea, variables especiales, funciones, etc.

### Auto-procesado de arrays

Para procesar un array se utiliza la etiqueta `<tpl for="variable">plantilla</tpl>`, teniendo en cuenta que:

- Si el valor especificado es un array se realizará un bucle por cada uno de sus elementos, repitiendo el código de la plantilla para cada elemento.
- Con `<tpl for=".">...</tpl>` se ejecuta un bucle a partir del nodo raíz.
- Con `<tpl for="foo">...</tpl>` se ejecuta el bucle a partir del nodo “foo”.
- Con `<tpl for="foo.bar">...</tpl>` se ejecuta el bucle a partir del nodo “foo.bar”
- Podemos usar la variable especial `{#}` para obtener el índice actual del array.

Si por ejemplo tenemos el siguiente objeto de datos:

```
var myData = {
    name: 'Tommy Maintz',
    drinks: ['Agua', 'Café', 'Leche'],
    kids: [
        { name: 'Tomás', age:3 },
        { name: 'Mateo', age:2 },
        { name: 'Salomón', age:0 }
    ]
};
```

Podríamos mostrar un listado con el contenido de “data.kids” de las siguientes formas:



```
var myTpl = new Ext.XTemplate(
    '<tpl for=".">',
    '<p>{#}. {name}</p>',
    '</tpl>' );
myTpl.overwrite(panel.body, myData.kids);

// Otra opción:
var myTpl = new Ext.XTemplate(
    '<tpl for="kids">',
    '<p>{#}. {name}</p>',
    '</tpl>' );
myTpl.overwrite(panel.body, myData);
```

Si el array solo contiene valores (en nuestro objeto de ejemplo, el array “drinks”), podemos usar la variable especial { . } dentro del bucle para obtener el valor actual:

```
var myTpl = new Ext.XTemplate(
    '<tpl for="drinks">',
    '<p>{#}. {.></p>',
    '</tpl>' );
myTpl.overwrite(panel.body, myData);
```

## Condiciones

Para introducir condiciones en las plantillas se utiliza la etiqueta '`<tpl if="condicion"> plantilla </tpl>`'. Pero hemos de tener en cuenta que: si utilizamos las “comillas”, deberemos escribirlas codificadas (&quot;), y que no existe el operador “else”, si lo necesitamos deberemos de utilizar un condición “if” adicional.

Ejemplos:

```
<tpl if="age > 1 && age < 10">Child</tpl>
<tpl if="age >= 10 && age < 18">Teenager</tpl>
<tpl if="name == &quot;Tommy&quot;">Hello!</tpl>
```

## Visualización

Para renderizar el contenido de una plantilla sobre un panel (u otro elemento que lo soporte, como veremos más adelante), podemos usar la función “`tpl.overwrite()`” que ya hemos usado en los ejemplos anteriores. O usar la propiedades “tpl” junto con “data”, de la forma:

```
MyApp.views.viewport = new Ext.Panel({
    data: myData,
    tpl: myTpl
});
```

## Más información

Para más información sobre las plantillas (operaciones matemáticas, ejecución de código en línea, variables especiales, funciones, etc.) consultar directamente la API de Sencha Touch: <http://docs.sencha.com/ext-js/4-0/#/api/Ext.XTemplate>

## 11.4. Data Views

Los Data Views nos permiten mostrar datos de forma personalizada, mediante el uso de plantillas y opciones de formato. Principalmente se utilizan para mostrar datos provenientes de un Store y aplicarles formato utilizando las plantillas “XTemplate”, como hemos visto en la sección anterior. Además también proporcionan mecanismos para gestionar eventos como: click, doubleclick, mouseover, mouseout, etc., así como para permitir seleccionar los elementos mostrados (por medio de un itemSelector).

Continuando con los ejemplos anteriores, vamos a crear un DataView para mostrar el contenido de “myStore” por medio de plantilla “myTpl”:

```
var myDataView = new Ext.DataView({
    store: myStore,
    itemConfig: {
        tpl: myTpl
    },
    fullscreen: true
});
```

Esta “vista de datos” tendríamos que añadirla a un panel para poder visualizarlo en nuestra aplicación:

```
MyApp.views.viewport = new Ext.Panel({
    items: [ myDataView ]
});
```

## 11.5. Listados

Permiten mostrar datos usando una plantilla por defecto de tipo lista. Estos datos se obtienen directamente de un “store” (Ext.data.Store) y se mostrarán uno a uno en forma de listado según la plantilla definida en “itemTpl”. Además incorpora funcionalidades para gestionar eventos como: itemtap, itemdoubletap, containertap, etc.

Utilizarlo es muy simple, solo tendremos que definir el “store” que queremos utilizar, y la plantilla para cada uno de los elementos con “itemTpl”:

```
var myList = new Ext.List({
    store: myStore,
    itemTpl: '{firstName} ' +
            '{lastName}'
});
```

Es muy importante diferenciar “itemTpl” de la propiedad “tpl” que ya habíamos visto (en las que usábamos los XTemplate). En “itemTpl” se procesa cada elemento del listado individualmente, y además, utilizaremos como separador para el salto de línea el símbolo de unión “+” y no la coma “,”.

Una vez creado el listado solo nos faltará añadirlo a un panel para poder visualizarlo:

```
MyApp.views.viewport = new Ext.Panel({
    items: [ myList ]
});
```

En el “store” debemos de utilizar la propiedad “sorters” para ordenar el listado, pues sino nos aparecerá desordenado. Por ejemplo, podríamos indicar (en el “store”) que se ordene por el apellido “sorters: 'lastName’”.

### Obtener datos de la lista

Para obtener el almacén de datos asociado a un listado utilizamos la propiedad `getStore`:

```
var notesStore = myList.getStore();
```

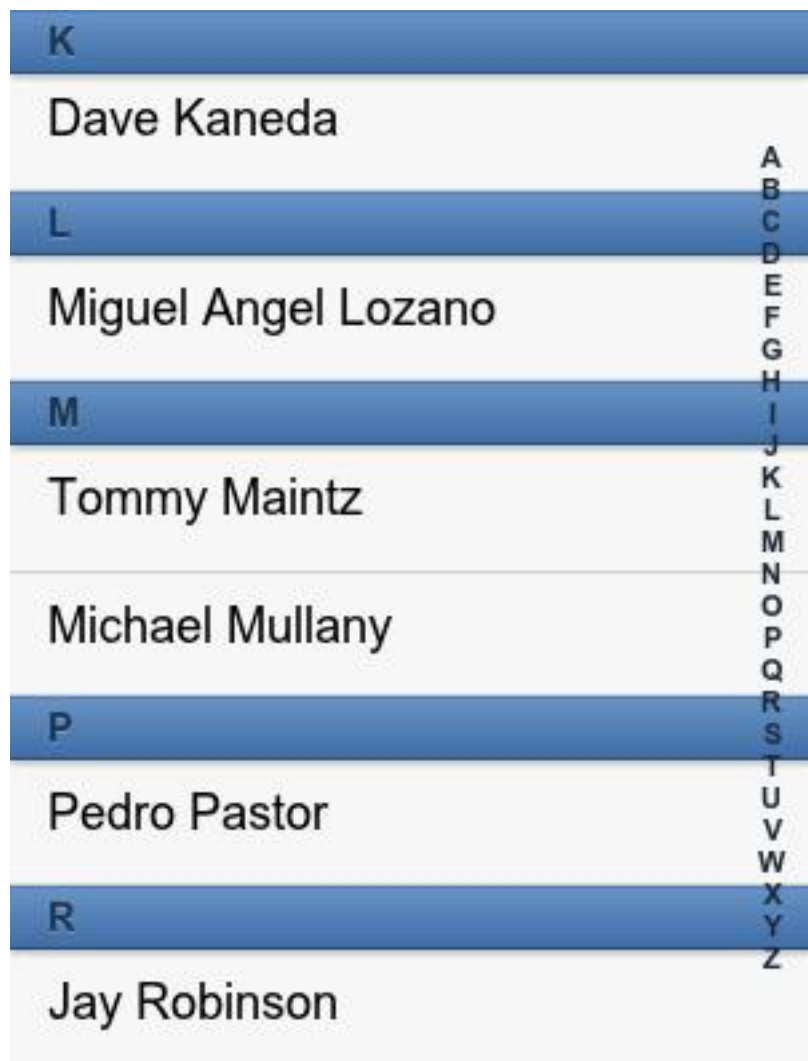
### Actualizar datos

Si modificamos el almacén de datos asociado con el listado tendremos que actualizarlo para que se visualicen correctamente los nuevos datos en el listado. En primer lugar llamaremos al método `sync()` del Store para sincronizar los cambios. A continuación, si es necesario, ordenamos los datos (pues el registro se habrá añadido al final). En el ejemplo se ordenan de forma descendente por fecha. Por último llamamos al método `refresh()` del listado para actualizar la vista.

```
notesStore.sync();
notesStore.sort([{ property: 'date', direction: 'DESC' }]);
myList.refresh();
```

### Agrupar elementos

Una propiedad muy útil que nos ofrecen los listados es la posibilidad de agrupar los elementos (como podemos ver en la imagen inferior). Para esto activaremos la propiedad “grouped: true” y opcionalmente podremos indicar que se muestre una barra lateral de navegación “indexBar: true”.



Pero para que esta propiedad funcione dentro del “store” tenemos que indicar la forma de agrupar los elementos. Tenemos dos opciones:

- `groupField: 'campo'` - para agrupar por un campo (por ejemplo: elementos de género masculino y femenino).
- `getGroupString: function(instance) {...}` - para agrupar usando una función. Esta opción es mucho más avanzada y nos permitirá agrupar, por ejemplo, usando la primera letra del apellido (como se muestra en la imagen de ejemplo).

Para obtener el resultado de la imagen de ejemplo, el código quedaría:

```
var myStore = new Ext.data.Store({
  model: 'Users',
  sorters: 'apellido',
  getGroupString: function(instance) {
```

```

        return instance.get('apellido')[0];
    }
});

var myList = new Ext.List({
    store: myStore,
    grouped : true,
    indexBar: true,
    itemTpl: '{nombre} {apellido}'
});

```

## Acciones

Para añadir acciones al presionar sobre un elemento del listado tenemos varias opciones:

- **itemtap**: permite realizar una acción al presionar sobre un elemento de la barra. Este evento lo debemos definir dentro de la sección “listeners” de nuestro Ext.List, de la forma:

```

listeners: {
    itemtap: function(record, index) { alert( "tap on" + index ); }
}

```

Donde el parámetro *record* representa el objeto sobre el que se ha pulsado. Este valor lo podríamos aprovechar para cargarlo directamente en un formulario o realizar alguna operación con él.

- **itemdoubletap**: permite realizar una acción al presionar dos veces consecutivas sobre un elemento. Este evento lo debemos definir dentro de la sección “listeners” de nuestro Ext.List, de la forma:

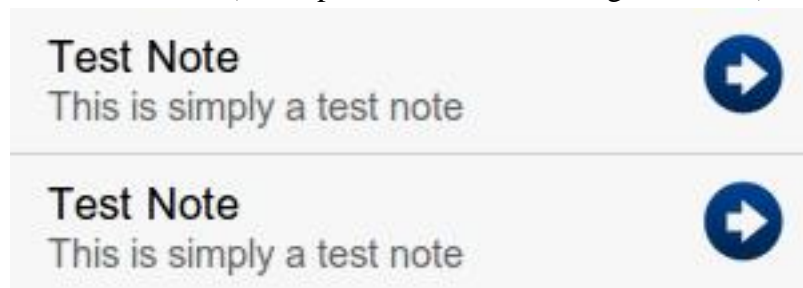
```

listeners: {
    itemdoubletap: function(record, index){ alert( "doubletap on " + index ); }
}

```

Donde el parámetro *record* representa el objeto sobre el que se ha pulsado. Este valor lo podríamos aprovechar para cargarlo directamente en un formulario o realizar alguna operación con él.

- **onItemDisclosure**: Boolean / Función - esta propiedad admite varios valores. Si le indicamos el valor booleano “true” simplemente añadirá un icono con una flecha a la derecha de cada elemento (como podemos ver en la imagen inferior).



En lugar de un valor booleano, podemos indicarle una función. En este caso se añadirá también el icono en cada elemento, y además ejecutará la función cada vez

que se presiones sobre dicho icono. Solo se capturará cuando se presione sobre el icono, no sobre toda la barra (como en el caso de `itemtap`). Un ejemplo de código:

```
onItemDisclosure: function (record) { alert( "item disclosure" ); }
```

Donde el parámetro *record* representa el objeto sobre el que se ha pulsado. Este valor lo podríamos aprovechar para cargarlo directamente en un formulario o realizar alguna operación con él.

Por ejemplo, en el siguiente código, al pulsar sobre un elemento de la lista se cargan los datos del elemento pulsado en un formulario (como veremos más adelante), y se cambia la vista para visualizar ese panel.

```
onItemDisclosure: function( record )
{
    MyApp.views.myFormPanel.load( record );
    MyApp.views.viewport.setActiveItem(
        'myFormPanel',
        { type: 'slide', direction: 'left' } );
}
```

## 11.6. Formularios

Para crear formularios utilizamos el constructor “`Ext.form.FormPanel`”, que se comporta exactamente igual que un panel, pero en el que podemos añadir fácilmente campos insertándolos en el array “`items`”:

```
var noteEditor = new Ext.form.FormPanel({
    id: 'noteEditor',
    items: [
        {
            xtype: 'textfield',
            name: 'title',
            label: 'Title',
            required: true
        },
        {
            xtype: 'textareafield',
            name: 'narrative',
            label: 'Narrative'
        }
    ]
});
```

### Tipos de campos

Para todos los campos podemos especificar un nombre “`name`”, una etiqueta “`label`” y si es requerido “`required:true`” (esta propiedad solo es visual, añade un asterisco (\*) en el nombre del campo, pero no realiza ninguna validación). El nombre se utiliza para cargar y enviar los datos del formulario (como veremos más adelante), y la etiqueta se mostrará visualmente en la parte izquierda de cada campo. El valor de todos los campos se encuentra en su atributo “`value`”, podemos utilizarlo para especificar un valor inicial.

Los principales tipos de campos que podemos utilizar son los siguientes (indicados según su nombre “xtype”):

- **textfield**: campo de texto (el código utilizado lo podemos ver al principio de esta sección):

Title*	Campo de prueba
--------	-----------------

- **textareafield**: área de texto (el código utilizado lo podemos ver al principio de esta sección):

Narrative	Área de texto
-----------	---------------

- **passwordfield**: campo de texto para introducir contraseñas. El código es igual que para un textfield pero cambiando el valor de “xtype: 'passwordfield'” :

Password:	.....
-----------	-------

- **urlfield**: campo de texto para direcciones Web, incluye validación de URL correcta:

Website	http://www.ua.es
---------	------------------

- **emailfield**: campo de texto para introducir e-mails, incluye validación automática:

Email	jgallego@dlsi.ua.es
-------	---------------------

- **togglefield**: permite seleccionar entre dos valores (0 ó 1). Por defecto se encuentra desactivado, para activarlo por defecto tenemos que añadir “value:1” a la definición del campo:

Activado:	
-----------	---

- **numberfield**: campo numérico, permite introducir el número manualmente o mediante las flechas laterales. Inicialmente no contiene ningún valor, podemos definir un valor inicial mediante la propiedad “value: 20”:



- **spinnerfield**: campo numérico, permite introducir el número manualmente o mediante los botones laterales. Inicialmente su valor es 0, podemos definir un valor inicial mediante la propiedad “value: 20”. También podemos definir un valor mínimo “minValue: 0”, un valor máximo “maxValue: 100”, el incremento “incrementValue: 2” y si se permiten ciclos “cycle: true”.



- **sliderfield**: campo numérico modificable mediante una barra o slider. Inicialmente su valor es 0, podemos definir un valor inicial mediante la propiedad “value: 50”. También podemos definir un valor mínimo “minValue: 0”, un valor máximo “maxValue: 100” y el incremento “incrementValue: 2”.



- **datepickerfield**: campo para seleccionar fechas. Al pulsar sobre el campo aparece una ventana en la que podemos seleccionar fácilmente una fecha. Podemos indicarle una fecha inicial utilizando “value: {year: 1989, day: 1, month: 5}”:



The screenshot shows a mobile application interface with a form titled "Birthday". The date "05/01/1989" is displayed at the top. Below the title bar, there are "Cancel" and "Done" buttons. The main content area displays a date picker with three rows: "April", "May", and "June". The "May" row is highlighted in blue, and the date "1" is selected, resulting in the date "1989" being displayed on the right.

- **fieldset:** Este elemento en realidad no es un campo de datos, sino un contenedor. No añade ninguna funcionalidad, simplemente poner un título (opcional), y agrupar elementos similares, de la forma:

```
items: [{
  xtype: 'fieldset',
  title: 'About Me',
  items: [
    { xtype: 'textfield', name: 'firstName', label: 'First Name' },
    { xtype: 'textfield', name: 'lastName', label: 'Last Name' }
  ]
}]
```

Con lo que obtendríamos un resultado similar a:

The screenshot shows a mobile application interface with a form titled "About Me". The form contains two text input fields. The first field is labeled "First Name" and contains the text "Antonio Javier". The second field is labeled "Last Name" and contains the text "Gallego Sánchez".

- **selectfield:** campo desplegable para seleccionar entre una lista de valores. Los valores disponibles se indican mediante la propiedad "options" como un array:

```
items:[{
  xtype : 'selectfield',
  label : 'Select',
```

```

options: [
  {text: 'First Option', value: 'first'},
  {text: 'Second Option', value: 'second'},
  {text: 'Third Option', value: 'third'}
]
}]

```

Con lo que obtendríamos un resultados como el siguiente:



- **checkboxfield:** el campo checkbox nos permite elegir uno o varios elementos de una lista. Cada campo de la lista se tiene que declarar como un item independiente, y todos ellos deben de tener el mismo nombre “name” para poder ser agrupados (muy importante para poder recoger los datos correctamente). Además podemos utilizar la propiedad “checked: true” para que aparezcan marcados inicialmente:

```

items: [
  {
    xtype: 'checkboxfield',
    name: 'check_color',
    value: 'red',
    label: 'Red',
    checked: true
  }, {
    xtype: 'checkboxfield',
    name: 'check_color',
    value: 'green',
    label: 'Green'
  }, {
    xtype: 'checkboxfield',
    name: 'check_color',
    value: 'blue',
    label: 'Blue'
  }
]

```

Con lo que obtendríamos un resultado como el siguiente:

Select - Colores	
Red	<input checked="" type="checkbox"/>
Green	<input checked="" type="checkbox"/>
Blue	<input type="checkbox"/>

- **radiofield**: el campo de tipo “radio” nos permite elegir **solo un elemento** de una lista. Cada campo de la lista se tiene que declarar como un ítem independiente, y todos ellos deben de tener el mismo nombre “name” para poder ser agrupados (muy importante para poder recoger los datos correctamente). Además podemos utilizar la propiedad “checked: true” en uno de ellos para que aparezca marcado inicialmente:

```
items: [
  {
    xtype: 'radiofield',
    name: 'radio_color',
    value: 'red',
    label: 'Red',
    checked: true
  }, {
    xtype: 'radiofield',
    name: 'radio_color',
    value: 'green',
    label: 'Green'
  }, {
    xtype: 'radiofield',
    name: 'radio_color',
    value: 'blue',
    label: 'Blue'
  }
]
```

Con lo que obtendríamos un resultado similar a:

Radio - Colores	
Red	<input checked="" type="checkbox"/>
Green	<input type="checkbox"/>
Blue	<input type="checkbox"/>

También podemos instanciar los campos de un formulario de forma independiente utilizando su constructor, por ejemplo para el campo de “textfield” sería “Ext.form.Text”, o para el campo “togglefield” sería “Ext.form.Toggle”. En general el constructor tendrá el mismo nombre que su tipo “xtype” pero quitando el sufijo “field”.

### Cargar datos en un formulario

Para insertar datos en un formulario utilizamos el método “load( data )” de la clase “FormPanel”. Por ejemplo, en el formulario que hemos creado al principio de esta sección:

```
noteEditor.load( note );
```

Este método recibe como parámetro una instancia de un modelo de datos (ver sección Data Model), y cargará solamente los campos cuyos nombre coincidan. En el ejemplo del inicio de esta sección tenemos dos campos cuyo nombre son “name: 'title'” y “name: 'text'”, si cargamos una instancia de un modelo de datos como el siguiente, solamente cargaría los dos campos que coinciden.

```
Ext.regModel('Note', {
    idProperty: 'id',
    fields: [
        { name: 'id', type: 'int' },
        { name: 'date', type: 'date', dateFormat: 'c' },
        { name: 'title', type: 'string' },
        { name: 'text', type: 'string' }
    ]
});
```

En la sección de “Data Model” ya hemos visto que podemos usar la función “Ext.ModelMgr.create” para crear instancias de un modelo de datos. Por lo que, por ejemplo, en la función “handler” de un botón, podríamos crear una instancia de este modelo y cargarlo en el formulario, de la forma:

```

{
  text: 'Cargar datos',
  ui: 'action',
  handler: function () {
    var now = new Date();
    var noteId = now.getTime();
    var note = Ext.ModelMgr.create(
      {
        id: noteId,
        date: now,
        title: 'titulo',
        text: 'texto' },
      'Note');
    noteEditor.load( note );
  }
}

```

### Guardar los datos de un formulario

En primer lugar llamaremos al método `getRecord()` del formulario para obtener un objeto con los datos introducidos. A continuación deberemos llamar a la función `updateRecord(objeto)` del mismo formulario para transferir estos valores a su instancia interna del modelo (tenemos que diferenciar entre los datos introducidos en los campos del navegador y el objeto interno de esa instancia). Si no llamásemos a este método la instancia del formulario estaría vacía. Después de esto deberíamos de validar los errores en los datos introducidos (ver siguiente apartado) y por último guardar los datos.

```

var currentNote = noteEditor.getRecord();
noteEditor.updateRecord(currentNote);
// Realizar validaciones (ver siguiente apartado)
// Guardar los datos

```

Si tenemos una instancia del almacén de datos creada (ver sección Data Store) podemos añadir los datos simplemente llamando a la función `add`:

```

notesStore.add(currentNote);

```

Más comúnmente tendremos nuestro almacén de datos asociado con algún elemento que nos permita visualizar los datos (como un listado o un Data View, ver secciones correspondientes). En el ejemplo del listado deberíamos de obtener la instancia del almacén de datos llamando a su método `getStore()` y posteriormente añadir los datos:

```

var notesStore = notesList.getStore();
notesStore.add( currentNote );

```

Opcionalmente podemos comprobar si los datos a añadir están repetidos. Para esto utilizaremos el método `findRecord()` del Store (ver sección Data Store).

```

var notesStore = notesList.getStore();

```

```
if( notesStore.findRecord('id', currentNote.data.id) === null)
{
    notesStore.add( currentNote );
}
```

Una vez añadidos los datos, y por terminar el ejemplo del listado, tendremos que sincronizar el Store, ordenarlo (si fuese necesario) y por último actualizar la vista del listado:

```
notesStore.sync();
notesStore.sort([{ property: 'date', direction: 'DESC'}]);
notesList.refresh();
```

### Comprobar validaciones

Para comprobar las validaciones de un formulario lo tendremos que hacer de forma manual llamando a la función `validate()`. La cual comprobará que se cumplen todas las validaciones que hayamos creado para el modelo de datos. Si continuamos con el ejemplo del modelo de datos “Note” podríamos añadir que los campos `id`, `title` y `narrative` son requeridos.

```
Ext.regModel('Note', {
    idProperty: 'id',
    fields: [
        { name: 'id', type: 'int' },
        { name: 'date', type: 'date', dateFormat: 'c' },
        { name: 'title', type: 'string' },
        { name: 'narrative', type: 'string' }
    ],
    validations: [
        { type: 'presence', field: 'id' },
        { type: 'presence', field: 'title',
            message: 'Introduzca un título para esta nota' },
        { type: 'presence', field: 'narrative',
            message: 'Introduzca un texto para esta nota' }
    ]
});
```

En primer lugar, para validar un formulario tenemos que cargar los datos introducidos en el formulario (con la función `getRecord()`) y a continuación actualizar la instancia del formulario con esos datos. Si no lo hacemos así al obtener los datos del formulario aparecería vacío.

Para validar utilizamos la función `validate()`, la cual devuelve un objeto tipo `Errors`. Podemos usar la función `isValid()` del objeto `Errors` para comprobar si ha habido algún error. En este caso tendremos que mostrar un aviso y no realizar ninguna acción más. Para mostrar el aviso usaremos un `MessageBox` (ver sección correspondiente). Además, dado que pueden haber varios errores (guardados en el array `items` del objeto `Errors`), vamos a iterar por los elementos de este array usando `Ext.each()` y concatenarlos:

```
var currentNote = noteEditor.getRecord();
```

```
noteEditor.updateRecord(currentNote);
var errors = currentNote.validate();
if (!errors.isValid())
{
    var message="";
    Ext.each(errors.items,function(rec,i){
        message += rec.message+"
";
    });
    Ext.Msg.alert("Validate", message, Ext.emptyFn);
    return; // Terminamos si hay errores
}
// Almacenar datos...
```

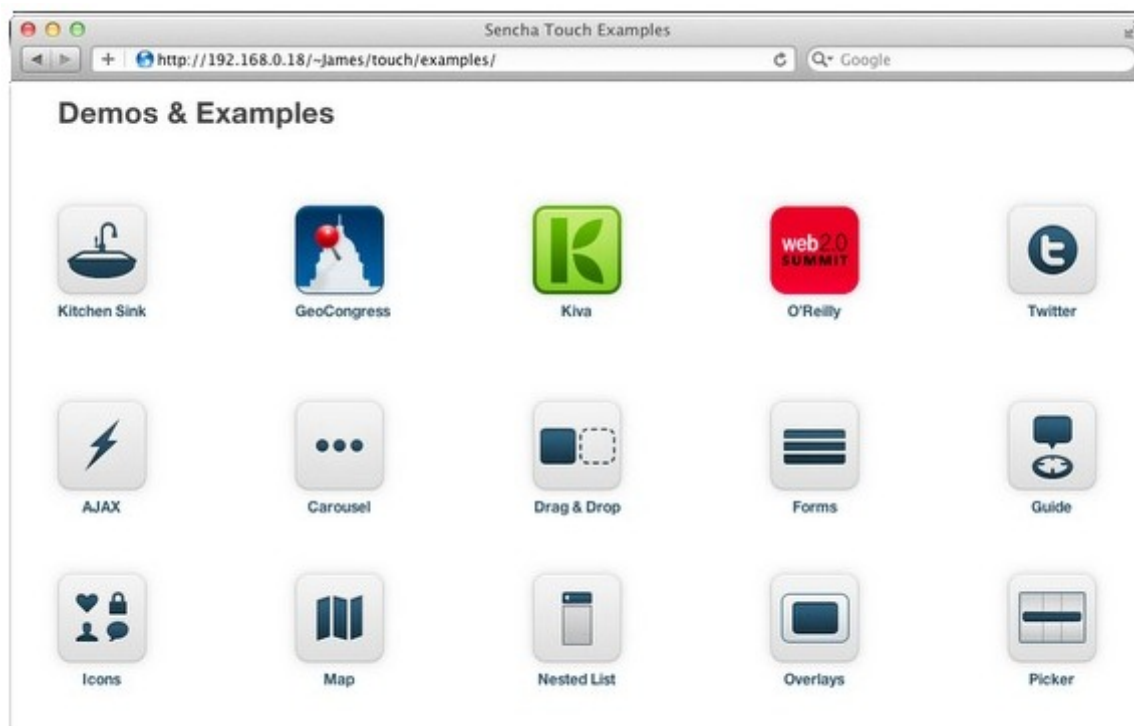
También podríamos haber creado un bucle para iterar entre los elementos del array, o haber llamado a la función `errors.getByField('title')[0].message` para obtener directamente el mensaje de error de un campo en concreto.

## 11.7. Más información

Podemos consultar principalmente tres fuentes de información cuando tengamos alguna duda:

- Los **tutoriales y la sección de FAQ** en la página Web de Sencha Touch:  
<http://www.sencha.com/>
- La **documentación API** Online:
  - <http://docs.sencha.com/touch/1-1/>
  - <http://www.sencha.com/learn/touch/>
  - También disponible de forma local accediendo en la dirección “/docs” de tu SDK, por ejemplo:  
<http://localhost/touch/docs/>
- Los **foros** en la página web de Sencha Touch:  
<http://www.sencha.com/forum/forumdisplay.php?56-Sencha-Touch-Forums>

Además en la carpeta “touch/examples” podemos encontrar aplicaciones de ejemplo. Estos ejemplos también están disponibles de forma online en “<http://localhost/touch/examples/>”:



En GitHub también podemos encontrar numerosas aplicaciones de ejemplo bien documentadas, estas las podemos encontrar en la dirección “<https://github.com/senchalearn/>”.



## 12. Ejercicios - Aspectos avanzados de Sencha Touch

En esta sesión vamos a continuar con el ejercicio del editor de notas de la sesión anterior. En primer lugar añadiremos los elementos necesarios para poder guardar las notas y visualizarlas en un listado. También crearemos el formulario y los botones necesarios para crear, editar y borrar notas.

Si necesitas ayuda puedes descargar la plantilla para los ejercicios [sesion06-ejercicios.zip](#). Al descomprimirlo aparecerán tres carpetas para cada uno de los ejercicios.

### 12.1. Ejercicio 1 - Modelo de datos y Listado

En este primer ejercicio vamos a crear el modelo de datos a utilizar y el almacén donde se van a guardar. Además crearemos una primera versión del listado con datos de prueba.

Nuestro modelo de datos (con identificador 'modeloNotas') tendrá cuatro campos: 'id' de tipo 'int', 'date' de tipo 'date' y formato 'c', 'title' de tipo 'string' y 'text' de tipo 'string'. Además deberemos definir 'id' como identificador único (ver sección "Data Model") y las siguientes validaciones: los campos 'id', 'title' y 'text' serán requeridos, y para los campos 'title' y 'text' modificaremos el mensaje de error por defecto, poniendo "Introduzca un título/texto".

A nuestro almacén de datos le pondremos como identificador 'storeNotas' y le indicaremos que tiene que usar el modelo 'modeloNotas'. Como proxy vamos a usar el almacenamiento en local, indicando como identificador 'misNotas-app-localstore'. Además indicaremos que se ordenen los datos de forma descendente (DESC) por fecha (campo 'date'), y que se carguen los datos del repositorio al inicio (autoLoad: true).

De forma temporal y para poder ver los resultados vamos a insertar datos en el Store, añadiendo las siguientes líneas:

```
data: [
  { id: 1, date: new Date(), title: 'Test 1', text: 'texto de prueba' },
  { id: 2, date: new Date(), title: 'Test 2', text: 'texto de prueba' },
  { id: 3, date: new Date(), title: 'Test 3', text: 'texto de prueba' },
  { id: 4, date: new Date(), title: 'Test 4', text: 'texto de prueba' }
]
```

Por último vamos a crear el listado donde se visualizarán los datos. Le pondremos como identificador 'panelLista' y le diremos que utilice el store 'storeNotas' que hemos definido previamente.

Además le indicaremos en el `itemTpl` que muestre el campo `{title}` dentro de una capa con el estilo "list-item-title", y que el campo `{text}` lo muestre a continuación en otra

capa que use el estilo "list-item-text". Este listado lo tendremos que añadir a nuestro panel 'panelContenedorLista' en su sección `items` para poder visualizarlo.

Ahora tenemos que añadir esos estilos al fichero `app.css`. Para ambas clases definiremos los mismos estilos (ver código siguiente), salvo para el "list-item-text" que añadiremos el color gris al texto.

```
float:left;
width:100%;
font-size:80%;
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
```

## 12.2. Ejercicio 2 - Formulario, Crear y Editar notas

En primer lugar vamos a crear el formulario. Para esto editamos el panel 'panelFormulario' que ya teníamos hecho, y le cambiamos su constructor de panel por uno de tipo formulario (ver sección de formularios). Además tenemos que quitar también los campos `layout: fit` y el texto HTML que teníamos puesto de prueba. En este formulario vamos a utilizar dos campos:

- Un campo de texto con el nombre 'title' (debe de coincidir con el modelo), con la etiqueta "Título:" y activaremos la opción de requerido.
- Un área de texto con nombre 'text', etiqueta "Texto:" y que también sea requerido.

Ahora vamos a modificar la función `handler` del botón "Nueva nota" para que al pulsarlo, **antes de realizar la transición**, cree una nueva nota y la asigne al formulario. Para esto vamos a añadir una llamada a la función `crearNuevaNota()`, la cual crearemos de forma separada.

En la función `crearNuevaNota()` en primer lugar nos guardaremos la fecha actual `var now = new Date();` y a continuación obtendremos el identificador del registro a crear `var noteID = now.getTime();` (con esta función transformamos la fecha en milisegundos, con lo que obtenemos un número que no se repite que podemos usar como ID). A continuación creamos un registro del modelo 'modeloNotas' (ver sección "Data Model") y lo cargamos en nuestro formulario (`panelFormulario.load( note );`).

Por último vamos a añadir la funcionalidad de editar las notas creadas. Para esto vamos hasta el 'panelLista', y definimos su función `onItemDisclosure`. Esta función recoge un parámetro (*record*) que tenemos que cargar en el 'panelFormulario' (`panelFormulario.load( record );`). A continuación realizaremos una transición de tipo 'slide' hacia la izquierda y con una duración de 1 segundo, para mostrar el 'panelFormulario' (ver sección "Listados").

## 12.3. Ejercicio 3 - Guardar y Borrar notas

En este último ejercicio vamos a añadir las acciones de guardar y borrar nota. En las funciones `handler` de los botones "Guardar" y "Borrar" añadiremos llamadas a las funciones `guardarNota()` y `borrarNota()` respectivamente. Las llamadas a estas funciones las deberemos de incluir antes de realizar la transición. A continuación definiremos las acciones a realizar en estas funciones (que estarán definidas de forma separada).

En la función `guardarNota()` realizaremos los siguientes pasos:

1. En primer lugar tendremos que cargar los datos introducidos en el formulario (usaremos la función `getRecord()` sobre la variable que contiene el formulario), y a continuación actualizaremos la instancia del formulario (ver sección "Guardar los datos de un formulario").
2. En segundo lugar comprobaremos si hay algún error de validación y mostraremos los errores (ver apartado "Comprobar validaciones" de la sección "Formularios").
3. Una vez validado el registro obtenido procederemos a guardar los datos. Obtenemos el *Store* usado por el listado y añadimos el registro solo si este no está repetido (función `findRecord()`, ver el apartado "Guardar los datos de un formulario").
4. Por último actualizamos el *Store* (`sync()`), lo reordenamos por fecha, y refrescamos el listado (`refresh()`).

La función `borrarNota()` es más sencilla:

1. Asignamos a variables el registro actual del formulario (`getRecord()`) y el *Store* usado por el listado (`getStore()`).
2. A continuación comprobaremos si existe el registro actual en el *store* (usando su "id") y si es así lo eliminaremos (ver apartado "Eliminar registros" de la sección "Data Store").
3. Por último actualizaremos los datos del *Store* (`sync()`) y refrescamos el listado (`refresh()`).

## 13. Bibliografía

### 13.1. Libros

- [Programming the Mobile Web \(Maximiliano Firtman\) - Ed: O'Reilly](#)
- [Programación en Internet: Clientes Web \(Sergio Luján Mora\) - Ed: Editorial Club Universitario. Libro completo gratuito en pdf](#)
- [HTML5 Mobile Web Development \(Jake Carter\) - Ed: O'Reilly](#)
- [HTML5 and CSS3 Develop with Tomorrow's Standards Today \(Brian P. Hogan\) - Ed: Pragmatic Bookshelf](#)
- [jQuery Mobile: Up and Running \(Maximiliano Firtman\) - Ed: O'Reilly](#)
- [jQuery Mobile \(Jon Reid\) - Ed: O'Reilly](#)
- [Sencha Touch 1.0 Mobile JavaScript Framework \(SSVV Narasimha Rao\) - Ed: Packt Publishing](#)

### 13.2. Enlaces

- [Guía Breve de Web Móvil de la W3C](#)
- [Estándar de HTML del W3C](#)
- [Wikilibros - Lenguaje HTML](#)
- [Referencia detallada de estilos CSS](#)
- [Especificaciones CSS](#)
- [Tutoriales CSS](#)
- [Estándar HTML5 de la W3C](#)
- [Demos de HTML5](#)
- [Tutoriales y ejemplos de HTML5](#)
- [Introducción a CSS3](#)
- [Tutoriales de CSS3 de la W3School](#)
- [Página oficial de jQuery](#)
- [Página oficial de jQuery Mobile](#)
- [Galería de aplicaciones desarrolladas con jQuery Mobile](#)
- [Página oficial de Sencha Touch](#)
- [Documentación API Online de Sencha Touch](#)
- [Tutoriales de Sencha Touch](#)
- [Tutoriales de Sencha Touch en vídeo](#)

### 13.3. Twitter

- [HTML5](#)
- [jQuery Mobile](#)
- [jQuery](#)
- [jQuery Mobile Gallery](#)

- [Sencha Touch](#)

## 13.4. Podcasts

---

- [Podcast oficial de jQuery](#)