Departamento de Telemática

PROYECTO FIN DE CARRERA

Aplicaciones Web Móviles con persistencia, estudio comparado con aplicaciones nativas en iOS y Android y caso práctico de thin client para Wordpress

Autor: David Ortiz Sáez

Tutor: Luis Ángel Galindo Sánchez

Leganés, Septiembre de 2012

Título: Aplicaciones Web Móviles con persistencia, estudio comparado con aplicaciones nativas en iOS y Android y caso práctico de thin client para Wordpress
Autor: David Ortiz Sáez
Director: Luis Ángel Galindo Sánchez

EL TRIBUNAL

Presidente: David Larrabeiti López

Vocal: Florina Almenares Mendoza

Secretario: José María Sierra Cámara

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 21 de Septiembre de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

A mi familia, en especial a mis padres y a mi hermana por toda la paciencia que han tenido conmigo durante estos años y por todo el apoyo que me han dado, espero saber estar a la altura cuando me toque a mí devolver todo este apoyo y este cariño.

Por supuesto a Laura, porque sin ella todo esto habría sido imposible, tantas prácticas, tantas horas en la universidad, tantos momentos juntos y aún así siempre ha estado ahí para apoyarme, aguantarme y darme fuerzas y valor para continuar día tras día. No, sin ella definitivamente no habría sido posible.

A todos los integrantes de Tyven Systems S.L. en especial a Esteban por todos sus consejos y ayudas que me han servido para realizar este proyecto con éxito y para mejorar profesionalmente y como persona.

A mi tutor Luis por darme la oportunidad de realizar este proyecto y por el soporte que me ha proporcionado durante la duración del mismo.

No puedo olvidarme de todos mis amigos que siempre se han interesado por mí, han estado siempre ahí después de las épocas de exámenes sin verme y me han hecho pasar grandes momentos. Los Martín y Rober, Mosi, Mary, Andrés, Dani, Irene, Samu, Sergio, Nadeem, Irene, Alex, Berta, Alberto, Lorena, Camacho, Barrajón… y alguno más que si me he olvidado de él espero que me perdone.

Por último agradecer a todos los compañeros de la universidad que siempre han estado ahí para solucionar dudas, prestarnos apuntes y también por qué no, para pasar un buen rato.

Resumen

Este proyecto consiste en el desarrollo de una página web mediante la tecnología HTML5 y el empleo del framework Dojo Mobile, de modo que la web se asemeje a una aplicación móvil. Además implementaremos un thin client para Wordpress que permita a la aplicación interactuar y descargar contenidos de un servidor ajeno.

Para completar el desarrollo de nuestra aplicación la dotaremos de persistencia. Esto nos servirá para entender por qué HTML5 nos permite el diseño de aplicaciones móviles mientras que anteriormente era impensable.

Además de realizar los aspectos tecnológicos, en este proyecto pretendemos comparar el desarrollo vía HTML5 con el desarrollo de forma nativa. Para ello, tras finalizar el desarrollo, realizaremos una comparación del mismo con el desarrollo realizado de forma nativa con el fin de discernir sobre los puntos fuertes y los débiles que se obtienen de trabajar de este modo, así como aportar nuestra visión sobre las líneas de futuro trabajo que debe seguir HTML5.

Palabras clave: HTML5, Dojo Mobile, Movilforum, Android, iOS, Javascript, Persistencia, CSS, JSON.

Abstract

This Project consists in the development of a HTML5 and Dojo Mobile based web page. By using these technologies the web page will seem like a native application. Furthermore we are going to implement a Wordpress' thin client which will allow the web to interact and download contents from an external server.

In order to complete the development of this project, we will make our application persistent in time. This will be useful to help us to understand why HTML5 allows us to develop mobile applications meanwhile the previous technologies didn't.

In addition to make the technological development, in this project we are also going to compare the development using HTML5 against of the native development. Once the development is finished, we are going to compare our development with a native development to learn about the strengths and weaknesses of HTML5. At the end, we are going to consider the future lines of work of HTML5.

Keywords: HTML5, Dojo Mobile, Movilforum, Android, iOS, Javascript, Persistencia, CSS, JSON.

Índice de contenidos

Índice de figuras

CAPITULO 1. Introducción

1.1 Introducción

Este proyecto ha versado sobre la aplicación de la tecnología, de relativamente reciente aparición, HTML5, para el diseño de páginas web con funcionalidades y aspecto similares a las de las aplicaciones móviles o de escritorio. Surge este proyecto como continuación a la asignatura Estudio Tecnológico [1] cuyo título fue "Estudio sobre el empleo de la tecnología HTML5 para aplicaciones móviles", en él se investigaron las nuevas características que aporta HTML5, la importancia que ha adquirido Javascript en la industria, los patrones de programación, la utilidad de los *frameworks* Javascript así como una comparativa entre el *framework* Dojo frente a Jquery más Backbone.js.

Se pretende con este proyecto ir un paso más allá y poner en práctica todo el contenido teórico adquirido en la asignatura previa así como otros conocimientos que son necesarios a la hora de llevar a cabo la implementación real. Una de estas consideraciones que debemos llevar a cabo es el *responsive design* [2]. *Responsive design* consiste en hacer nuestra web adaptable a todo tipo de dispositivos, ajustando su tamaño, su resolución, etc. Esto como veremos será de vital importancia en nuestro cometido ya que en el mundo móvil abundan dispositivos de todo tipo.

Junto con el *responsive design* debemos tener en cuenta el *graceful degradation*, como se nos informa en [3], es relativamente común acceder a una web y ver un mensaje que nos indica que esa web sólo está disponible para Internet Explorer. En este caso, el usuario en lugar de cambiar su navegador para poder visionar esa página, probablemente no vuelva a intentar acceder a la misma. Esto es debido a que el sitio web en cuestión no ha tenido en cuenta el *graceful degradation*, es decir, hacer nuestra página visible para todo tipo de navegador aunque éste no cumpla con los requisitos necesarios para hacer funcionar completamente la página web. En el caso de que se acceda a nuestra página desde un navegador no compatible al 100% con las funcionalidades de nuestra web, debemos gestionar esto y mostrar al usuario nuestra página web con aquellas funcionalidades que no sean compatibles con su navegador capadas, pero manteniendo las que sí lo sean intactas. La idea de *graceful degradation*

es que un sistema (en este caso una página web) debe continuar funcionando incluso en el caso de que alguna de sus partes falle.

Veremos a lo largo de esta memoria que hemos podido cumplir con todas las especificaciones, no sin antes tener que sortear problemas derivados de la incompatibilidad de algunos dispositivos así como de problemas intrínsecos a la funcionalidad, en ocasiones limitada, que Dojo nos aporta. Durante el resto de este capítulo vamos a realizar una introducción a las características fundamentales de HTML5 y de Dojo.

1.2 HTML5, ¿por qué ahora?

HTML5 es un compendio de tecnologías que han ido desarrollándose durante varios años hasta converger en este punto. Como podemos extraer de [4] y [5] las siglas HTML5 provienen de *HyperText Markup Language* y son la continuación de HTML4, pero debido a que HTML5 ha pasado a ser un reclamo *marketiniano*, las siglas HTML5 engloban también a dos tecnologías sin las que el diseño no sería posible: Javascript y CSS3.

Si echamos la vista atrás en la historia podemos observar lo siguiente:

- 1991 HTML
- 1994 HTML 2
- 1996 CSS 1 + Javascript
- 1997 HTML 4
- 1998 CSS 2
- 2000 XHTML 1
- 2002 Tableless Web Design
- 2005 AJAX
- 2009 HTML 5

Centrándonos en los años más recientes vemos como HTML5 ha ido sustituyendo paulatinamente a AJAX. AJAX a su vez surgió tras una "guerra" entre los diferentes navegadores debida a la falta de estandarización. Microsoft durante este periodo creó la extensión que se conoció como XMLHttpRequest y tras una evolución se convirtió en la tecnología AJAX.

El problema de la no estandarización de los lenguajes es de capital importancia en esta industria y nos lleva a entender por qué la tecnología HTML5 se ha impuesto actualmente a XHTML.

El organismo internacional de estandarización de lenguajes web W3C (*World Wide Web Consortium*) se mostraba inicialmente más decantado hacia XHTML ya que esta tecnología apostaba por la estandarización de HTML4, pero, la lentitud en el proceso de estandarización, unido a desavenencias en lo referente al criterio del mismo, provocaron que en 2004 Opera, Firefox y Apple fundaran el Web Hypertext Application Technology Working Group (WhatWG [6]) al margen del W3C cuya finalidad era crear una nueva versión del estándar desde un punto de vista más práctico y no tan académico como pretendía el W3C [7].

Además, debido a la utilización que realiza del browser (concibiéndolo como una herramienta con funcionalidades y no sólo como un mero útil para mostrar la información), HTML5 permite realizar diseños web similares a las aplicaciones de escritorio, lo cual fue otra razón fundamental para su creación por parte de WhatWG. Por estos motivos el W3C se vio obligado a aceptar la coexistencia de ambos tipos de estándares y terminó definiendo a HTML5 como "un vocabulario y APIS asociadas para HTML y XHTML" y comenzó su estandarización cuya finalización se prevé para 2014 [8]. No obstante, uno de los riesgos que entraña que el proceso de estandarización sea lento, es que la tecnología ya se encuentra disponible para su uso y continuamente los diseñadores se encuentran creando nuevas funcionalidades, lo cual puede desencadenar una mala estandarización al final, como avisa Douglas Crockford, creador de Javascrpit, en el último párrafo de [9].

Como comentamos en el párrafo anterior, los diseñadores desean crear páginas web más similares a aplicaciones de escritorio, y esto se consigue gracias a las nuevas funcionalidades añadidas que presenta HTML5 como vamos a ver a continuación.

1.2.1 HTML5, funcionalidades añadidas.

Las diversas tecnologías que forman lo que conocemos como HTML5 nos permiten diseñar interfaces mucho más customizadas y agradables a la vista del usuario, además de aportar un mayor dinamismo a la página en cuestión.

Con HTML5 surgen nuevos *markups* con significado propio que nos permiten realizar una web con un significado más claro a primera vista, en lugar de tener que emplear el *tag* seguido de un identificador que le aporte un significado característico, tenemos nuevas etiquetas como <header> o <article> que por sí mismas

nos indican su significado y hasta su posición lógica en el DOM. En la figura 1 puede apreciarse con claridad lo que se ha comentado.

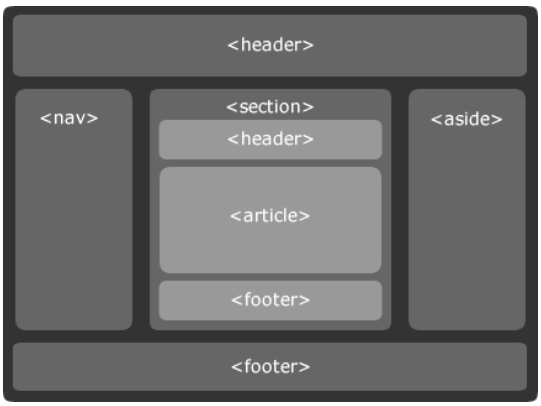

Figura 1. Nuevos Tag de HTML5. Fuente [10]

Este es un nuevo añadido sobre HTML4, pero no es aquel que le hace diferenciarse. Lo que ha conseguido hacer a HTML5 reinar frente a otros como XHTML han sido su funcionalidades nuevas, como puedan ser la comunicación bidireccional en tiempo real (aportando para ello tres tecnologías diferentes: *Web Workers*, *Web Sockets* y las notificaciones), la geolocalización o el nuevo tratado de imágenes y vídeo empleando Canvas o SVG que permiten realizar una web mucho más atractiva visualmente.

No obstante, vamos a centrarnos en la característica de HTML5 que ha sido indispensable para la realización de este proyecto, la persistencia.

Persistencia

Entendemos por persistencia el permitir que el contenido (en su totalidad o una parte) permanezca inalterado entre dos sesiones diferentes. Podemos definir la persistencia a nivel de memoria, a nivel de aplicación o a nivel de objeto [11], pero lo que nos interesa es mantener la sesión abierta entre dos accesos, sin necesidad de tener que reenviar la información cada vez que se acceda a la página.

Este problema se venía solventando mediante el uso de las cookies, solución no del todo satisfactoria principalmente por dos motivos: el almacenamiento mediante cookies tiene un tamaño máximo de 4KB (el cual en nuestro caso es a todas luces insuficiente como veremos posteriormente) y el usuario puede borrar en cualquier momento las cookies de su navegador borrando así toda la información que hayamos podido almacenar.

Para solventar el problema del espacio en HTML5 se ha conseguido implementar de varias maneras posibles una base de datos en el *front-end*, es decir, una base de datos que se encuentra almacenada en el dispositivo mediante el que se accede y de la cual se pueden extraer los contenidos directamente sin necesidad de disponer de conexión, ya que se encuentra en el propio dispositivo.

Existen varias tecnologías capaces de llevar a cabo este cometido, por ejemplo *WebStorage* y *Web SQL Database*. Nótese de nuevo la característica no estandarización que estará presente en todo el documento, que hace que aparezcan varias tecnologías diferentes para un mismo fin. A continuación presentamos una descripción de ambas siguiendo la comparativa realizada en [12].

WebStorage emplea una pareja de datos-clave para el almacenamiento de contenidos, así se logra extraer el contenido mediante la clave obtenida en el almacenamiento. Este método es bastante sencillo, pero tiene algunos inconvenientes como que hay que tener mucho cuidado al realizar las transacciones en tiempo real y además no permite realizar una búsqueda en base a un criterio genérico si no que hay que realizar la búsqueda iterando, con la carga computacional extra que esto supone.

Por otro lado tenemos la tecnología *Web SQL Database*, la cual es mucho más robusta al ser una base de datos basada en SQL, permitiendo realizar búsquedas genéricas sin tener que implementar una búsqueda iterativa. No obstante presenta el inconveniente de que dos grandes como Microsoft y Firefox no tienen pensado implementarlo.

Para nuestro diseño hemos optado por utilizar la tecnología *Web SQL Database* por una razón fundamental, la carga computacional. En aplicaciones móviles es de vital importancia la rapidez y la sencillez, es por ello que, el poder acceder a los datos filtrando por peticiones en vez de iterando, se nos presenta a priori más atractivo. Además, como sabemos, los navegadores Android y Safari están basados en *webkit* [13] y soportarán esta tecnología sin problemas.

Cabe reseñar llegados a este punto que, además del empleo de *Web SQL Database* para almacenar contenido específico de nuestra aplicación, HTML5 nos proporciona otro tipo de almacenamiento para almacenar datos genéricos y permitir una carga de la página más rápida. Este, el denominado almacenamiento mediante *application cache*, es un almacenamiento que se realiza en el servidor de la aplicación web y almacena como

hemos comentado datos característicos y genéricos habituales en la aplicación, procediendo a descargarlos de una forma mucho más rápida al usuario.

No obstante, sirvan estos párrafos a modo de introducción ya que en el capítulo 4 detallaremos con claridad como hemos llevado a cabo cada tipo de almacenamiento.

1.2.2 Dojo y Dojo Mobile.

Hemos comentado las bondades de HTML5, pero como hemos visto, no es sólo el lenguaje de *markup* lo que le permite a HTML5 destacar, son los lenguajes Javascript y CSS3 los que lo complementan de manera excelente.

Javascript [14] es un lenguaje que lleva existiendo varios años, pero no es hasta hace relativamente poco que su uso se ha incrementado exponencialmente, para encontrar la respuesta a esto debemos remontarnos a sus orígenes y, una vez más, al proceso de estandarización. Debido a que Javascript es un estándar y a la dificultad de estandarizar el mismo de modo que fuera compatible con todos los navegadores, se realizaron una serie de concesiones las cuales provocan que puedan comerse errores comúnmente de difícil detección. Pero, a su vez, el lograr un lenguaje que fuera compatible con todos los browsers, fue una de las claves del éxito de Javascript, como cuenta su creador Douglas Crockford en [15].

Para ayudarnos a lidiar con estos errores existen los *frameworks*. Los *frameworks* son librerías Javascript que nos proporcionan funciones ya implementadas para el tratado del DOM, de los eventos y otras características. Pero lo más importante para Javascript es que, implementando código Javascript mediante el empleo de frameworks nos abstraemos de cometer los errores típicos comentados en el párrafo anterior. Es gracias en gran medida a los *frameworks* que Javascript hoy en día es un lenguaje profundamente aceptado y utilizado.

Existen varios *frameworks* Javascript como por ejemplo jQuery, Mootools, prototype.js o Dojo. En la figura 2 podemos apreciar la búsqueda por palabras clave en Google de algunos de los *frameworks* comentados.

Figura 2. Evolución histórica de la búsqueda en Google de los diferentes Frameworks. Fuente [16]

Como puede apreciarse el *framework* más popular es jQuery, encontrándose Dojo incluso en tercer lugar. Esto nos plantea la pregunta de por qué escoger Dojo como framework para la implementación de nuestra web.

En nuestro caso hemos elegido Dojo, porque presenta una forma más robusta que jQuery, encontrándose formado por diferentes módulos: Dojo Base, Dojo Core, Dijit, DojoX y Dojo Util. Los cuales podremos cargar total o parcialmente en función de las características que necesitemos, lo que nos lleva a comparar con jQuery, que presenta un gran módulo inicial mediante el cual, en caso de necesitar otras funcionalidades, deberemos ir descargando diversos *plugins*.

Otras dos características que nos han hecho decantarnos por Dojo han sido que en su propia estructura se encuentra implementado el patrón de programación MVC [17] y que Dojo presenta una mayor velocidad de cálculo. Durante el Estudio Tecnológico anteriormente referenciado, hicimos una comparación entre la velocidad de los diferentes *frameworks* en distintas plataformas obteniendo como puede verse a continuación que tanto para Android, como para iOS, como para el navegador más extendido del mercado, Google Chrome el *framework* más rápido es Dojo.

Test		Ops/sec
Dojo	`var d1 = query(".content");`	124,648 ±2.90% fastest
jQuery	`var j1 = jQuery('.content');`	63,084 ±1.52% 49% slower
Mootols	`var m1 = $$('.content');`	64,372 ±0.44% 48% slower
jQuery - inside	`var j2 = jQuery('.content p');`	15,939 ±2.65% 87% slower
Mootools - inside	`var m2 = $$('.content p');`	16,021 ±2.74% 87% slower

Figura 3. Comparativa de la velocidad de los frameworks en navegador Google Chrome. Fuente [18]

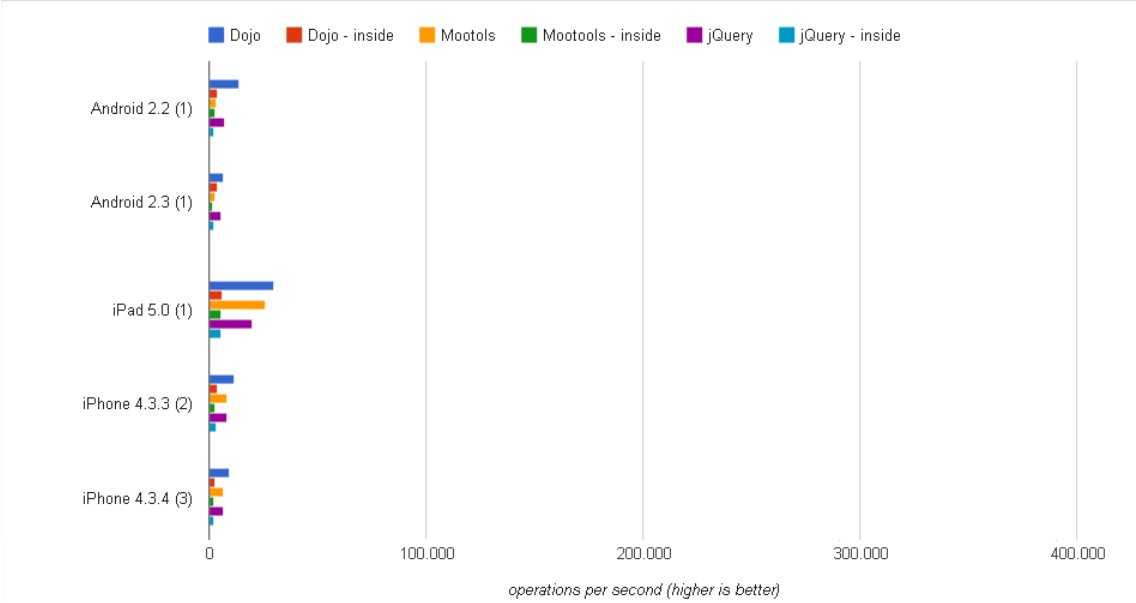

Figura 4. Comparativa de la velocidad de los frameworks en dispositivos Android e iOS. Fuente [18]

Una vez que nos hemos decantado por Dojo, vamos a examinar su extensión, Dojo Mobile (que se encuentra dentro del módulo DojoX) y las oportunidades que nos presenta.

Mediante Dojo Mobile podemos realizar una aplicación con dinamismo basada en el intercambio de vistas al igual que sucede con las diferentes aplicaciones móviles. Además, Dojo Mobile nos proporciona diferentes animaciones para las transiciones entre las distintas pantallas lo cual, a priori, nos proporciona una sensación de no

encontrarnos navegando por una página web si no de encontrarnos frente a una aplicación móvil, que es lo que se pretende.

Sirva este punto como introducción a Dojo Mobile, volveremos posteriormente en los capítulos 2 y 3 sobre los elementos que nos proporciona Dojo Mobile y por qué elegimos cada uno para entenderlo en más detalle.

1.2.3 Wordpress.

Como sabemos, de un tiempo a esta parte en el mundo web predomina la denominada Web 2.0, es decir aquella en la que el usuario puede participar activamente en la misma ya sea mediante comentarios o mediante algún otro tipo de actuación. Del mismo modo hay usuarios que no solo quieren dar su opinión sobre un tema, sus inquietudes van más allá y desean tener una página web donde puedan publicar sus propios contenidos y sean otros usuarios los que los comenten.

No obstante, en muchas ocasiones los usuarios no tienen conocimientos sobre diseño web y es ahí donde surge la fuerza de Wordpress [19].

Con Wordpress se nos permite la creación de un blog de forma tremendamente sencilla, pudiendo llegar a customizar el mismo hasta límites que nos hacen olvidar que estamos viendo un blog todo ello mediante la instalación de diferentes *plugins*. Pueden observarse algunos ejemplos de estas webs en [20].

Es esta característica la que nos hace ver por qué Wordpress tiene tanto éxito. Permite al usuario implementar y personalizar hasta el extremo su propio blog pudiendo así publicar sus contenidos y realizar una web atractiva visualmente sin tener que aprender prácticamente ningún tipo de conocimiento extra para tal fin.

Es por este motivo por el que se ha decidido en este proyecto interactuar con un servidor Wordpress. Veremos con posterioridad como mediante peticiones no muy complicadas podemos obtener información bien clasificada por categorías, por número de post, etc. Luego la extensa aplicación de Wordpress en la web, unido a la posibilidad de interactuar con su servidor nos ha llevado a plantearnos trabajar con Wordpress en este proyecto.

1.3 Objetivos del proyecto

El objetivo de este proyecto es explorar la posibilidad de realizar una página web con aspecto de aplicación móvil o de escritorio mediante el empleo de la tecnología HTML5, además se pretende interactuar con un servidor Wordpress y dotar de persistencia a la aplicación.

De estos objetivos generales se nos derivan los siguientes objetivos más específicos:

1- **Diseño de una página web con aspecto de aplicación móvil**

Mediante el empleo de la librería Dojo Mobile se pretende realizar una web basada en diferentes vistas, que, mediante sus transiciones y su apariencia, logren hacer creer al usuario que se encuentra frente a una aplicación.

2- **Interactuación con servidor Wordpress**

Investigar de qué manera es posible obtener contenidos de un servidor Wordpress, hacer peticiones basadas en contenidos específicos, procesar la información recibida y mostrarla de cara al usuario con un formato específico.

3- **Persistencia de los contenidos**

Conseguir visualizar los contenidos de la aplicación sin necesidad de conexión mediante el empleo de las diversas características que nos proporciona HTML5 para este fin. Así mismo, conseguir una mayor fluidez en la descarga de contenidos mediante el cacheo de la página.

4- **Comparativa con aplicaciones nativas**

Adquirir los conocimientos suficientes sobre la tecnología HTML5 para su empleo en aplicaciones móviles. Conocer sus fortalezas, debilidades y limitaciones para poder comparar su empleo frente al diseño de aplicaciones de forma nativa.

1.4 Estructura de la memoria

La estructura de esta memoria se encuentra dividida en diferentes capítulos con sus correspondientes contenidos así como diversos anexos que ayudan a complementar la información sobre el mismo.

1. *Introducción.* Resumen sobre las tecnologías empleadas en este proyecto y motivo por el cual se han escogido.

2. *Primera aplicación. Wordpress.* Explicación del primer ejercicio realizado en el desarrollo del proyecto. Puesta en marcha de un servidor basado en Wordpress y diseño de una aplicación sencilla capaz de interactuar con el mismo.

3. *Interacción con servidor Wordpress real. Movilforum.* Explicación de la aplicación completamente operativa de forma online, estructura y diseño de la página web y motivos por los cuales se han implementado de esa forma.

4. *Persistencia.* Capítulo encargado de explicar cómo dotamos a la aplicación de persistencia así como de contar las diferentes modificaciones que se han llevado a cabo para la versión final de la aplicación.

5. *Comparativa HTML5 vs Aplicaciones nativas.* Capítulo que versa sobre las cualidades que presenta HTML5 frente al desarrollo de forma nativa.

6. *Conclusiones.* Conclusiones finales del desarrollo llevado a cabo.

CAPITULO 2. Primeros pasos con Wordpress

2.1 Creación blog con Wordpress

En este segundo capítulo vamos a comentar el primer ejercicio del proyecto que nos sirvió de toma de contacto con un servidor Apache, así como para familiarizarnos con el entorno Wordpress y cómo gestiona éste su contenido. Para ello lo que hicimos fue crearnos nuestro propio blog empleando Wordpress.

El primer paso para poder trabajar con Wordpress de forma local es poseer nuestro propio servidor. Nos evitaremos de esta forma el tener que subir los ficheros a un servidor externo, nos sirve también para mantener nuestro contenido de forma privada, aunque no sea este el cometido final de un blog, por ser este primer ejercicio meramente de contacto no estamos interesados en hacer público ningún contenido. Además, para todo usuario interesado en el uso de Wordpress es aconsejable poseer un servidor propio para poder testear localmente los cambios antes de poner el contenido online.

2.1.1 Instalación Apache, php y Mysql

Para nuestro servidor hemos optado por el paquete WAMP. Como extraemos de [21], se conoce como WAMP al acrónimo que engloba la instalación de un servidor Apache junto con Mysql para la gestión de la base de datos y php, todo ello para Windows.

Es importante reseñar que la instalación del paquete Mysql es indispensable para el funcionamiento de Wordpress, ya que este requiere de una base de datos para almacenar su contenido.

Vamos a describir a continuación como llevamos a cabo la instalación y configuración del paquete WAMP siguiendo los pasos que se nos aconsejan en [21].

1- Instalando Apache

El primer paso es instalar el servidor, la versión elegida para nuestro proyecto es la 2.2.22. Para su instalación descargamos el fichero y lo ejecutamos. En la ruta de instalación le indicamos que queremos la ruta C:\appserv. Es importante conocer la ruta exacta puesto que luego deberemos

acceder a algunos ficheros de configuración. Una vez se ha finalizado el proceso de instalación del servidor si abrimos un navegador y en la barra de navegación ponemos la dirección localhost visualizaremos lo siguiente

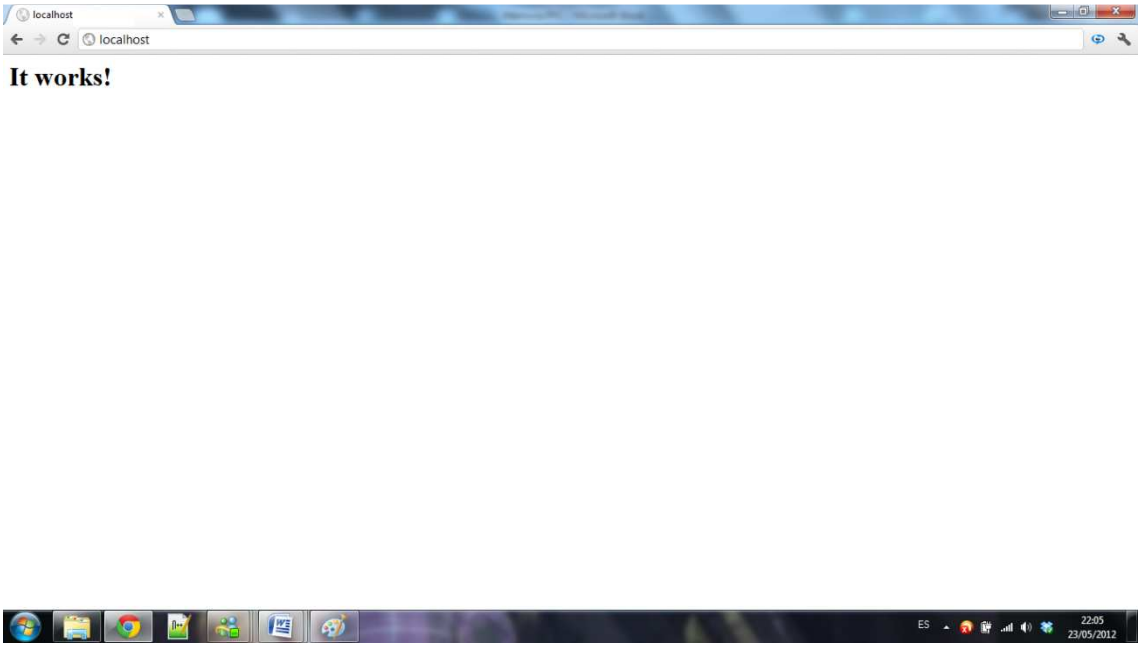

Figura 5. Página que muestra el servidor Apache una vez está operativo.

Que nos indica que la instalación del servidor apache se ha finalizado con éxito.

2- Instalando PHP

Una vez tenemos instalado y funcionando el servidor Apache vamos a proceder a la instalación de PHP y su integración con nuestro servidor. Para ello bajamos el archivo ejecutable para Windows 7 y lo ejecutamos, indicando como directorio para su instalación el directorio raíz C:\

Ya tenemos instalado PHP, vamos a proceder a integrarlo con nuestro servidor. Para ello acudimos a la carpeta de nuestro servidor, y en el directorio C:\appserv\conf\extra creamos un archivo denominado httpd-php.conf cuyo contenido debe ser igual al mostrado en la figura 6.

```
#load the php main library to avoid dll hell
Loadfile "C:\php\php5ts.dll"
#load the sapi so that apache can use php
LoadModule php5_module "C:\php\php5apache2_2.dll"
#set the php.ini location so that you don't have to waste time guessing where it is
PHPIniDir "C:\php"

#Hook the php file extensions
AddHandler application/x-httpd-php .php
AddHandler application/x-httpd-php-source .phps
```

Figura 6. Fichero de configuración httpd-php.conf.

Tras esto, debemos hacer algunas modificaciones en el archivo de configuración de nuestro servidor, esto es, en el archivo httpd.conf que se encuentra en C:\appserv\conf. Lo primero que debemos hacer es añadir al final de la sección “#Supplemental configuration” lo siguiente:

PHP settings
Include conf/extra/httpd-php.conf

Con esto le indicamos al servidor Apache que incluya en su configuración ese archivo. Guardamos las modificaciones realizadas y por último vamos a prepararnos para instalar nuestra base de datos, para ello vamos a habilitar el soporte para MySql.

Buscamos el archivo de configuración de PHP. Lo encontraremos en la ruta C:\php\php.ini. Abrimos el fichero y nos vamos a la sección “Paths and Directories” y modificamos la entrada extensión_dir para que quede como sigue:

extensión_dir = “C:\PHP\ext

Tras esto, habilitamos las extensiones de php, para ello nos vamos a la sección “Windows Extensions” y descomentamos las líneas:

extension = php_mysql.dll
extension = php_mysqli.dll

Guardamos el archivo y reiniciamos el servidor apache, para ello nos vamos a la barra de tareas, pinchamos sobre su icono y clicamos sobre restart. Tras esto, si abrimos el monitor de Apache podemos ver que ya se encuentra integrado con PHP.

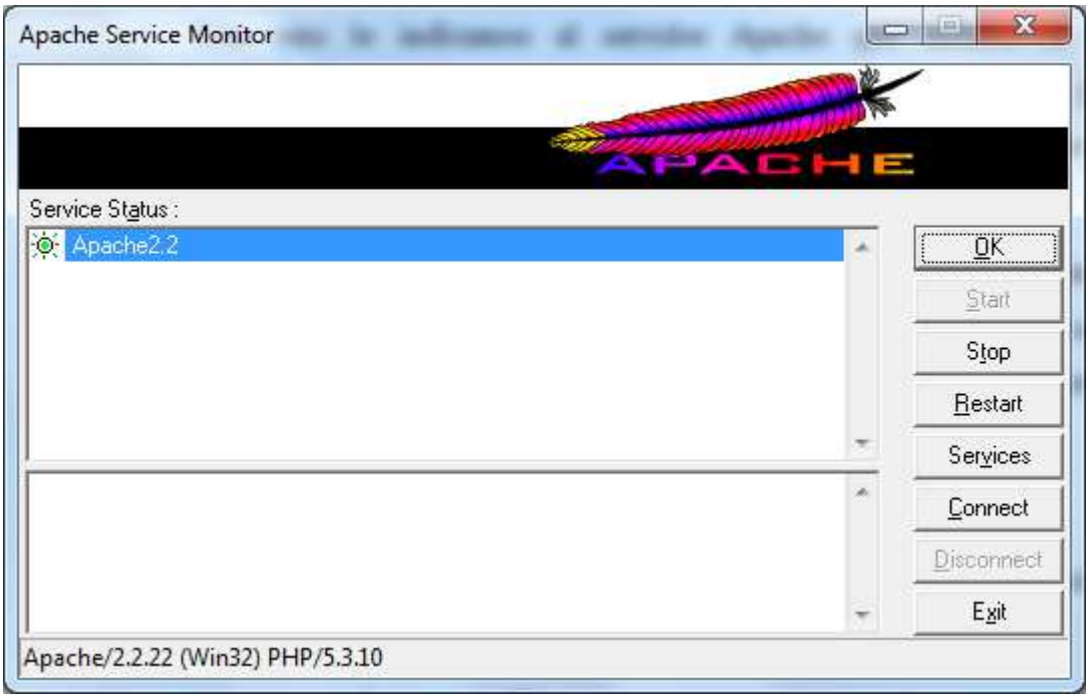

Figura 7. Monitor de Apache.

Vamos a instalar el último componente del paquete WAMP, el soporte para MySql.

3- Instalando MySql

Para la instalación de MySql nos bajamos el archivo binario y lo ejecutamos, elegimos una instalación personalizada y le indicamos el directorio donde debe instalarse, en nuestro caso C:\MySql. Tras esto iniciamos la instalación y una vez se ha finalizado, indicamos que queremos configurar el servidor MySql en ese momento.

Tras este paso se nos abre una nueva ventana en la que iremos indicando paso por paso lo siguiente:

Developer Machine → Multifuncional Database → Decision Support (DSS)/OLAP → Enable TCP/IP Networking (importante, abrimos el puerto que se nos indica) →Install as Windows Service e Include Bin Directory in Windows PATH.

Tras estas pantallas accedemos a la última de ellas en la cual indicaremos la contraseña que queremos para gestionar nuestra base de datos. Elegimos la contraseña deseada y procedemos a ejecutar la configuración.

Una vez hemos realizado estos pasos tenemos instalado nuestro servidor apache junto con php y MySQL. Podemos proceder a instalar Wordpress sobre nuestro servidor.

2.1.2 Instalación Wordpress y creación de varios post

El primer paso antes de instalar Wordpress es crear una base de datos con MySQL en nuestro servidor desde la cual pueda Wordpress gestionarse y almacenar nuevos contenidos. Para ello iniciamos el "MySQL 5.5 Command Line Client" de MySQL.

Lo primero que nos solicita es la contraseña de MySQL que le facilitamos en la instalación. La introducimos y ya podemos gestionar nuestras bases de datos. La creación de una base de datos es bastante simple, como podemos ver en [22] basta con ejecutar el siguiente comando:

CREATE DATABASE blog;

Nótese que blog es el nombre que hemos dado a la base de datos sobre la que operará Wordpress. Tras esto nos salimos del asistente ejecutando el comando:

\q

Bien, ya estamos listos para acometer la instalación de Wordpress. Nos descargamos el paquete de Wordpress y lo descomprimimos en una carpeta vacía dentro de la carpeta htdocs de nuestro servidor. En nuestro caso vamos a descomprimir el contenido directamente dentro de C:\appserv\htdocs, sin crear una subcarpeta, así cuando accedamos a la dirección localhost accederemos a nuestro blog sin necesidad de navegar entre subcarpetas.

Una vez tenemos descomprimido Wordpress, como nos indica [23] debemos acceder a su archivo de configuración "wp-config.php" y modificar las siguientes líneas para introducir la información relativa a nuestro servidor y a la base de datos que acabamos de crear:

```
/** El nombre de tu base de datos de WordPress */
define('DB_NAME', 'blog');

/** Tu nombre de usuario de MySQL */
define('DB_USER', 'root');

/** Tu contraseña de MySQL */
define('DB_PASSWORD', 'contrasena');

/** Host de MySQL (es muy probable que no necesites cambiarlo) */
define('DB_HOST', 'localhost');
```

Figura 8. Fichero configuración wp-config.php.

Tras guardar el fichero estamos listos para ejecutar el instalador de Wordpress, para ello accedemos desde nuestro navegador web a la siguiente dirección:

Localhost/wp-admin/install-php

En esa dirección se nos solicitará el nombre de acceso y la contraseña, los introducimos y se nos indicará un mensaje cuando la instalación haya finalizado con éxito. Para cerciorarnos de esto podemos acceder a la dirección localhost para visionar un contenido similar al mostrado en la figura 8.

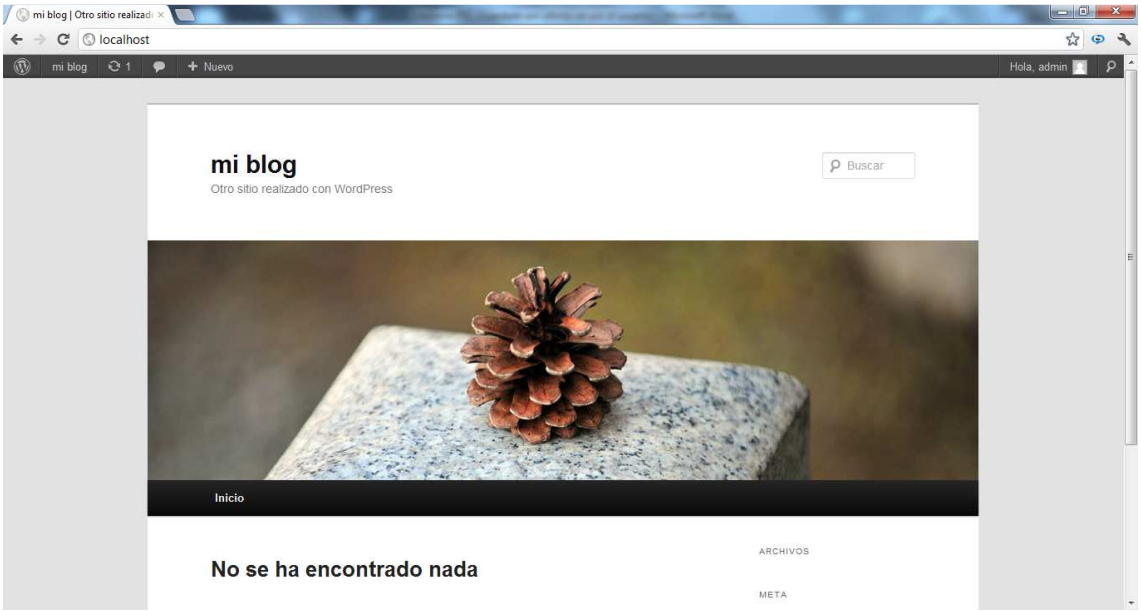

Figura 9. Aspecto inicial de Wordpress.

Ya tenemos disponible Wordpress en nuestro servidor, pero como se nos indica está vacío de contenidos. Vamos a proceder a llenar nuestro blog con post diferenciados por categorías para poder acceder posteriormente a ellos con nuestra aplicación.

Para la creación de contenidos en el blog accedemos como administrador y se nos mostrará la interfaz de gestión de nuestro blog con Wordpress. Tiene diversos menús

pero para nuestro cometido simplemente vamos a crear 6 post, dos de ellos pertenecientes a la categoría ocio, otros dos pertenecientes a la categoría académico, un quinto que pertenece a ambas categorías y un sexto al que no indicaremos ninguna categoría. Ya veremos posteriormente por qué dividimos los post de esta manera.

Lo primero que hacemos es crear las dos categorías. Para ello clicamos sobre la pestaña categorías y se nos muestran las categorías existentes (en este caso una única denominada "sin categoría"). Observamos un apartado donde indica "Añadir nueva categoría", añadimos el nombre "ocio", el slug "ocio" (el slug es importante porque será lo que nos permita identificar posteriormente esta categoría), una breve descripción de la categoría y pulsamos sobre el botón de añadir. Repetimos el proceso cambiando "ocio" por "académico" y añadimos la segunda categoría.

Una vez tenemos ambas procedemos a la creación de los post. Pulsamos sobre la pestaña situada a la izquierda añadir entrada y se nos abre el menú para crear la entrada. Rellenamos su título y el contenido que queramos (podemos insertar alguna imagen también), los etiquetamos en la categoría correspondiente y pulsamos sobre el botón publicar para hacerlos visibles.

En la figura 10 puede observarse como una vez realizado esto tenemos presente el contenido en el blog.

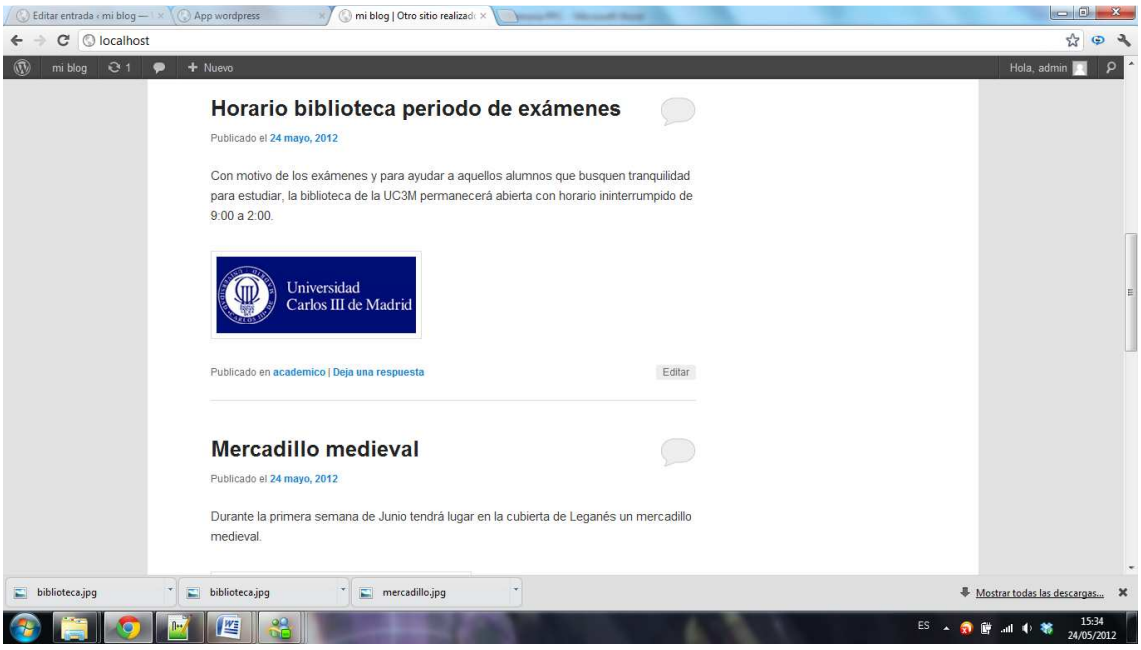

Figura 10. Aspecto de Wordpress relleno de contenidos.

Con todo esto tenemos listo nuestro servidor, perfectamente integrado con Wordpress y además con contenido publicado. Todo está listo para dar nuestros primeros pasos con Dojo Mobile y conseguir extraer los contenidos de nuestro servidor para nuestra aplicación.

2.2 Diseño página web que interactúe con Wordpress

En este apartado vamos a diseñar una página web con una interfaz sencilla, sin detenernos a elaborar en demasía su aspecto visual ya que lo que pretendemos es dar los primeros pasos con Dojo Mobile y conseguir recibir contenidos de nuestro servidor Wordpress.

En el primer punto vamos a considerar cómo diseñamos el aspecto de la página empleando Dojo Mobile, su comportamiento y qué acciones son susceptibles de producir errores. En el segundo punto investigaremos sobre cómo podemos extraer la información del servidor de Wordpress (el contenido del mismo) y en el tercer y último punto de este apartado veremos cómo, una vez hemos recibido la información del servidor la procesamos y extraemos su contenido para publicarlo al usuario.

2.2.1 Creación página web con Dojo

Si examinamos la documentación de la librería Dojox.Mobile disponible en [24] podemos encontrar el siguiente párrafo:

> The Dojo Mobile package provides a number of widgets that can be used to build web-based applications for mobile devices such as iPhone, Android, or BlackBerry. These widgets work best with webkit-based browsers, such as Safari or Chrome, since webkit-specific CSS3 features are extensively used. However, the widgets should work in a "graceful degradation" manner even on non-CSS3 browsers, such as IE or (older) Firefox. In that case, fancy effects, such as animation, gradient color, or round corner rectangle, may not work, but you can still operate your application. Furthermore, as a separate file, a compatibility module, dojox/mobile/compat, is available, which simulates some of CSS3 features used in this module. If you use the compatibility module, fancy visual effects work better even on non-CSS3 browsers.

En él se nos indica que Dojo nos proporciona una serie de Widgets que podemos emplear para construir aplicaciones basadas en la web y que serán perfectamente funcionales tanto en iPhone como en Android por ser navegadores basados en webkit. Además nos pone en conocimiento de que, a pesar de que puede perder alguna de sus características, también es compatible con navegadores no basados en webkit como

puedan ser Internet Explorer (cuyo motor es Trident [25]) o Firefox (basado en Gecko [26]).

De este primer párrafo extraemos que Dojo se encarga de resolvernos el problema de la no estandarización, nos ayuda con el *graceful degradation* y nos ahorrará tener que probar nuestro código Javascript en cada tipo de navegador para cerciorarnos de su funcionamiento. Como comentamos en el capítulo uno, este es uno de los motivos por los que Javascript se ha impuesto a otros lenguajes y a su vez por el que los *frameworks* se han extendido abundantemente, es una unión que beneficia a ambos.

Además de solventar los problemas de compatibilidad, Dojo Mobile nos presenta abundantes widgets con los que poder diseñar nuestra página. Mediante el empleo de widgets tipo View podemos conseguir que nuestra página web navegue entre distintas vistas cambiando entre una y otra con una animación tipo *slide* que Dojo se encarga de implementar. Este simple hecho ya nos permite diferenciarnos de una página web estática normal ya que la animación es muy similar a la que se produce en un dispositivo iOS o Android al navegar entre menús. En la figura 11 puede observarse el planteamiento de este widget.

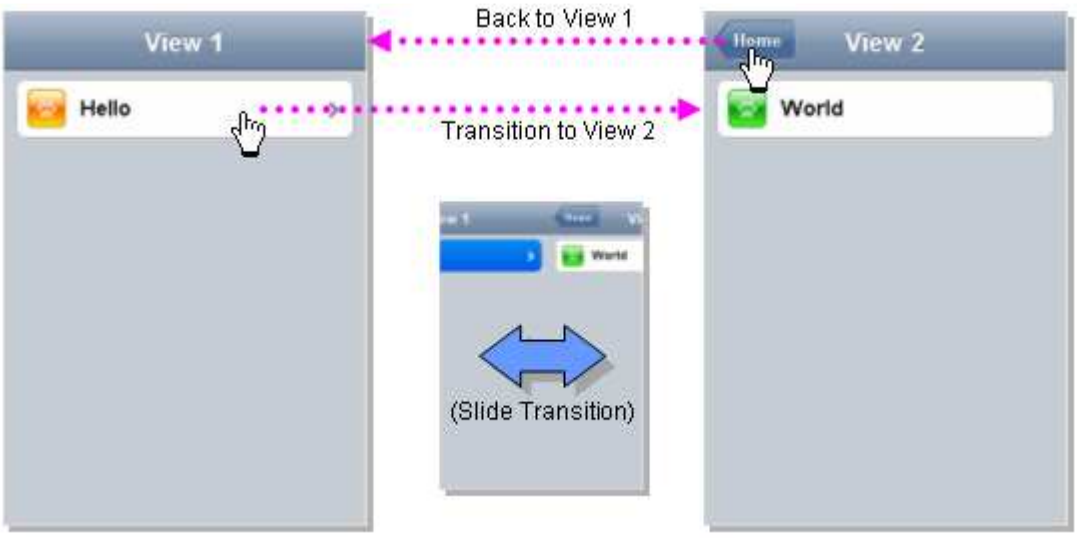

Figura 11. Esquema de funcionamiento de transición entre vistas Dojo Mobile. Fuente [24]

En la figura 11 se observan diversos objetos que son necesarios para permitir a Dojo ejecutar la transición. A continuación mostramos la porción de nuestro código que se encarga de la transición entre las vistas y comentaremos sobre ella cómo lo logramos.

```
<div id="principal" data-dojo-type="dojox.mobile.View" data-dojo-props="selected: true">
    <h1 data-dojo-type="dojox.mobile.Heading">Vista principal</h1>
    <ul data-dojo-type="dojox.mobile.RoundRectList">
        <li data-dojo-type="dojox.mobile.ListItem" data-dojo-props="icon:'../img/fiesta.jpg', moveTo: 'ocio'">
            Posts de la categoría ocio...

        </li>
        <li data-dojo-type="dojox.mobile.ListItem" data-dojo-props="icon:'../img/virrete.jpg', moveTo: 'academico'">
            Posts de la categoría académico...
        </li>

    <li data-dojo-type="dojox.mobile.ListItem" data-dojo-props="moveTo: 'instalacion1'">
      Instrucciones de instalación...
    </li>

    </ul>

</div>

<div id="ocio" data-dojo-type="dojox.mobile.View">
    <h1 data-dojo-type="dojox.mobile.Heading" data-dojo-props="back:'Principal', moveTo:'principal'">Ocio</h1>
```

Figura 12. Código que permite el intercambio entre vistas.

Como puede verse en la figura 12 tenemos *tags* de html5 normales, pero dentro de ellos les asignamos características intrínsecas a Dojo mediante la opción "<data-dojo-type>" o bien mediante "<data-dojo-props>". En la captura se observan dos diferentes tipos de View, el denominado principal y el denominado ocio. Mediante la propiedad de Dojo selected indicamos que la vista principal es la que se debe mostrar al usuario en primer lugar, luego será el controlador de Dojo el encargado de modificar esta propiedad haciendo visible la vista correspondiente de manera acorde al patrón MVC.

Mediante el tipo de Dojo Heading indicamos cual es el título de cada vista, en nuestro caso son Vista principal y Ocio. Dicho título se visualizará en la parte superior de la aplicación dependiendo de qué vista esté activa.

El último objeto necesario para poder realizar la transición entre dos vistas es un ListItem. A este objeto podemos otorgarle la propiedad moveTo mediante la cual, al indicarle el identificador de la vista a la que queremos movernos, al clicar sobre este objeto ListItem se iniciará la animación finalizando la misma en la vista indicada con moveTo.

Una vez se ha completado la transición, mediante la propiedad back podemos crear un botón en la cabecera de la nueva vista, que al clicarlo se desplace hasta la vista indicada en moveTo, como hemos realizado en el caso de la vista Ocio, mediante el cual le indicamos con back que vuelva a la vista principal.

Además cabe reseñar que también mediante las propiedades de Dojo podemos asignarle un icono a cada objeto, consiguiendo así realizar un aspecto muy parecido al que tiene por ejemplo el menú de ajustes de un dispositivo Android.

Por último hay que indicar que, a pesar de permitirnos realizar transiciones similares a los móviles, Dojox.Mobile tiene el inconveniente de que todo objeto del DOM debe tener asignado un tipo de Dojo Mobile, ya que si no se producirá un error y la página no funcionará correctamente.

La necesidad de que cada *tag* de HTML lleve asociado un tipo de objeto Dojo Mobile nos impedirá la creación de objetos en el DOM de forma dinámica empleando Javascript, hecho que nos ocasionará algún problema que comentaremos más adelante en los capítulos 3 y 4.

En este punto tenemos diseñado como va a ser el aspecto de nuestra página, pero para su correcto funcionamiento debemos cargar la librería de Dojo y en último lugar, debemos parsear la página empleando la librería dojox.mobile.parser. Para importar la librería utilizamos el API de Google y añadimos la siguiente línea en la cabecera:

```
<script src="http://ajax.googleapis.com/ajax/libs/dojo/1.7.1/dojo/dojo.js" data-dojo-config="async: true"></script>
```

Figura 13. Código para cargar librería Dojo.

Con esta línea cargamos el módulo principal de Dojo o lo que es lo mismo, la librería Dojo.Base, pero en nuestro caso necesitamos más funcionalidades a parte de las que nos proporciona esta librería (por ejemplo, Dojox.Mobile no se encuentra ahí). Para solucionar este problema Dojo nos proporciona el método require [27] que nos sirve para indicar a Dojo qué elementos adicionales necesitamos.

Nótese que este método es el más adecuado para aplicaciones móviles ya que nos da la posibilidad de descargar únicamente aquellos módulos que necesitemos para hacer funcionar nuestra aplicación, alcanzando así un compromiso ideal entre funcionamiento y datos descargados.

Así pues, para conseguir parsear la página y dotarla del aspecto que deseamos introducimos las siguientes líneas:

```
require(["dojox/mobile/parser", "dojox/mobile", "dojox/mobile/deviceTheme", "dojo/on", "dojo/domReady!", "dojo/io/script",
 "dojo/_base/lang", "dojox/mobile/compat", "dojo/domReady!"], function(parser,on) {
    dojo.requireIf(!dojo.isWebKit, "dojox.mobile.compat");
    parser.parse();
});
```

Figura 14. Uso de los métodos require y parse.

Como se observa de la figura 12 mediante el método require indicamos aquellos módulos que necesitamos y posteriormente mediante la línea parser.parse() formateamos nuestra página y le dotamos el aspecto que nos proporciona Dojo. Además mediante el módulo dojox.mobile.deviceTheme Dojo detecta automáticamente el User Agent desde el que se está accediendo y se encarga de cargar una hoja de estilo u otra diferente según el dispositivo. A continuación se adjuntan dos capturas de pantalla, mostrando la vista principal desde un móvil Android y desde el navegador Chrome.

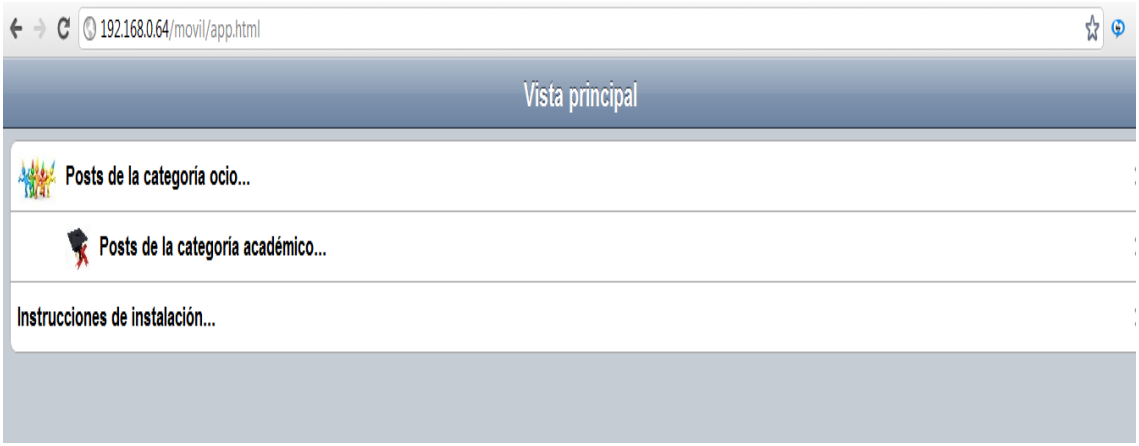

Figura 15. Aspecto de la página desde Chrome.

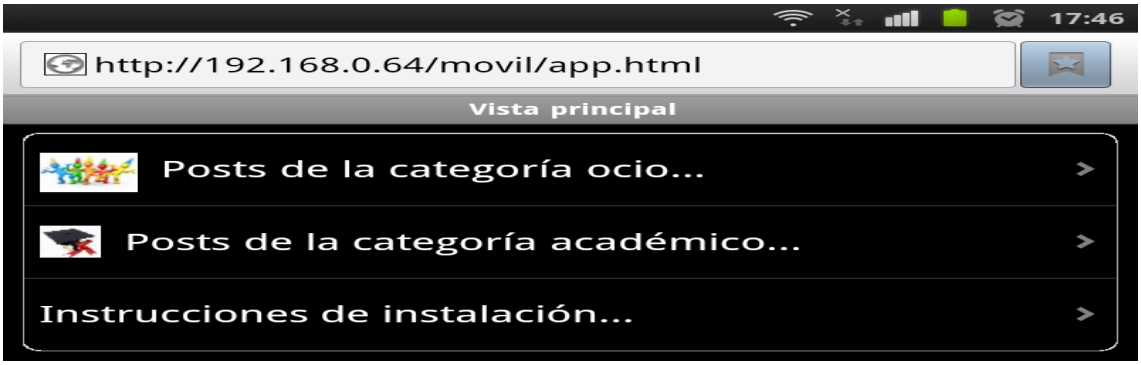

Figura 16. Aspecto de la página desde Android.

Como puede verse la hoja de estilo por defecto cargada es la de iPhone ya que a Chrome le asigna la hoja de estilos por defecto, sin embargo para Android le asigna una hoja de estilo personalizada para el dispositivo la cual dota a la aplicación de una apariencia muy similar al menú de ajustes de los dispositivos Android. No obstante,

como veremos posteriormente, podremos editar la hoja de estilo para elegir nosotros el aspecto que tendrá la página.

Vamos a echar un vistazo ahora a como queda nuestra vista de Ocio, en esta ocasión desde un móvil Android.

Figura 17. Aspecto de la vista de Ocio desde un móvil Android.

En esta captura podemos apreciar el botón que añadimos mediante la propiedad back, si pulsamos este botón volveremos a la vista Principal, ya que así se lo indicamos con la propiedad moveTo. Además vemos que tenemos creado el espacio para almacenar los post, pero se encuentra vacío, vamos ahora a explicar cómo realizamos las peticiones para rellenar este espacio con la información correcta.

2.2.2 Extracción información de Wordpress. API RESTful vs SOAP

Una vez tenemos diseñada nuestra primera página web con Dojo Mobile, vamos a ver cómo podemos extraer la información que hemos publicado anteriormente en el Wordpress para poder visionarla en nuestro dispositivo móvil sin tener que acceder a la página de nuestro blog directamente. Este es un paso importante porque si no conseguimos extraer la información desde un servidor externo hasta nuestra aplicación, ésta no podrá cumplir con su cometido final puesto que únicamente podríamos mostrarle al usuario los contenidos estáticos que nosotros deseemos y no los más recientes publicados en alguna página web externa.

Para acometer esta tarea existen diversos protocolos, siendo los dos con mayor presencia en la web SOAP y RESTful Web Services.

Tras la instalación de Wordpress, tenemos en nuestro directorio un archivo denominado xmlrpc.php que se encarga de gestionar las peticiones y respuestas empleando el protocolo xml-rpc que es el predecesor de SOAP [28].

Como podemos ver en la página web de xml-rpc [29], se nos dice lo siguiente:

(XML-RPC) It's a spec and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet.
It's remote procedure calling using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.

Este protocolo gestiona las peticiones codificándolas mediante XML y, a su vez, es independiente del protocolo de transporte de la información, pudiendo emplear HTTP, SMTP [30], TCP u otros. Además, los mensajes de SOAP presentan cabeceras adicionales que permiten realizar QoS [31] o establecer opciones de configuración.

No obstante, este protocolo no va a ser empleado en nuestro diseño puesto que, como indican en su propia página web los lenguajes soportados por este protocolo son:

for Perl, Python, Java, Frontier, C/C++, Lisp, PHP, Microsoft .NET, Rebol, Real Basic, Tcl, Delphi, WebObjects and Zope

Como podemos ver, Javascript no es uno de los lenguajes soportados, luego debemos buscar otras soluciones que nos permitan realizar peticiones y que sí se encuentren soportadas por Javascript.

Como comentamos anteriormente el otro protocolo con amplia presencia en la web es RESTful Web Services o REST [32].

REST basa sus peticiones en URIs, esto es importante puesto que provee un esquema de direccionamiento descentralizado acorde con la web. Cada objeto de petición se representa mediante una URI, al contrario que SOAP en el que cada objeto es independiente y puede tener sus propios métodos. El presentar un esquema de direccionamiento unificado permite a REST tener un menor consumo de recursos [33], lo cual es interesante para nuestro cometido.

REST presenta diferencias con SOAP en la concepción que tiene de HTTP, mientras que HTTP en SOAP es concebido únicamente como protocolo de transporte,

en REST HTTP es más que eso y se considera un protocolo de aplicación, empleando los métodos característicos GET, POST o PUT para realizar las peticiones [34]. No obstante esto tiene sus desventajas, si hay un fallo en un HTTP server, al emplear REST, la responsabilidad recae sobre el usuario.

Hay que indicar también que REST es un protocolo sin estado puesto que cada mensaje es autodescriptivo, llevando toda la información necesaria, pero se puede conseguir guardar el estado mediante diversas técnicas como el empleo de cookies u otras.

La última diferencia reseñable es que REST no presenta cabeceras adicionales como vimos que sí presentaba SOAP, esto le impide implementar QoS extremo a extremo, pero del mismo modo hace que presente un menor consumo de ancho de banda, lo cual nos hace decantarnos finalmente por emplear para nuestra aplicación un protocolo basado en RESTful Web Services.

Navegando entre los *plugins* de Wordpress encontramos el *plugin* para JSON, este *plugin* implementa peticiones de tipo REST y nos permite extraer contenido de Wordpress empleando, como hemos visto, peticiones de tipo HTTP. Vamos a ver ahora si se encuentra soportado por Javascript y más concretamente si nuestro *framework* Dojo nos proporciona el código para interactuar con él.

Encontramos la documentación de la librería dojo.io.script [35] en la cual se nos indica:

> Aside from being able to do cross-site data access, this implementation provides support for JSON which is a way callback names can be added to a script return and executed appropriately. Not all services that provide script-tag data formats support JSONP, but many do and it makes it very flexible.

Así pues, podemos concluir que hemos encontrado el protocolo que necesitamos, pero, al contrario que XMLRPC este protocolo no es soportado directamente por Wordpress, debemos proporcionarle soporte mediante un *plugin*. Para ello debemos descomprimir la API que nos descargamos de [36] en el siguiente directorio C:\appserv\htdocs\wp-content\plugins (directorio en el cual se deben instalar los *plugins* que añadamos a Wordpress).

Tras instalarlo en el directorio, no ha finalizado nuestra labor, debemos activar el *plugin* para que funcione correctamente. Accedemos a nuestro blog Wordpress como administradores y en la pestaña de *plugins* clicamos sobre JSON API y pulsamos activar. Ya tenemos operativo el *plugin*, es decir, ya podemos hacer peticiones desde nuestro código Javascript y obtener el contenido de los diferentes post de nuestro blog para mostrarlo en la aplicación. Vamos a explicar cómo realizamos esto a continuación.

2.2.3 JSON, formato de peticiones y decodificación de la información

Una vez hemos encontrado la herramienta adecuada para extraer información del servidor, vamos a dedicar un apartado a indagar sobre las diferentes formas de solicitar información que nos proporciona el *plugin* JSON [36].

Lo primero que observamos es que tenemos dos maneras de realizar peticiones, de manera implícita y de manera explícita. Si empleamos la manera implícita debemos realizar peticiones a una página concreta de Wordpress y la respuesta a la petición será devolvernos lo que se encuentre en esa página en formato JSON. Un ejemplo de este tipo de petición sería solicitar la siguiente dirección:

http://www.example.org/?p=47&json=1

Sin embargo JSON es más potente y nos proporciona diferentes métodos con distintos parámetros para filtrar nuestra petición por categorías, por fecha, por número de página…

Para poder emplear estos métodos debemos realizar peticiones de modo explícito indicándole el método que deseamos emplear y sus parámetros. Nótese que en este tipo de petición no estamos accediendo directamente al contenido de una página web, si no que le estamos indicando al servidor que nos filtre la información y nos devuelva únicamente lo que nosotros deseamos. Además para realizar las peticiones de modo explícito podemos emplear el parámetro JSON o podemos indicar directamente el método de la API. Veamos algunos ejemplos:

http://www.example.org/?json=get_recent_posts

http://www.example.org/api/get_recent_posts/

En ambas peticiones estamos empleando el modo explícito y estamos solicitando, mediante el método get_recent_posts que nos devuelva el servidor, en formato JSON el contenido de los últimos post publicados. En este caso no le estamos indicando el número de post que queremos, pero podríamos indicarle mediante un argumento más cuántos queremos que nos sean devueltos.

También podemos solicitar el contenido de un único post mediante el método get_post de la siguiente manera:

http://www.example.org/?json=get_post&post_id=47

http://www.example.org/api/get_post/?post_id=47

En estas peticiones, tenemos un ejemplo de cómo añadimos parámetros a la petición, en este caso, el método get_post necesita que le indiquemos el identificador del post que estamos solicitando.

Vamos a comprobar ahora el formato JSON que nos está devolviendo la petición, en la figura 16 puede apreciarse una respuesta a una petición:

```
{
  "status": "ok",
  "count": 1,
  "count_total": 1,
  "pages": 1,
  "posts": [
    {
      "id": 1,
      "type": "post",
      "slug": "hello-world",
      "url": "http:\/\/localhost\/wordpress\/?p=1",
      "title": "Hello world!",
      "title_plain": "Hello world!",
      "content": "<p>Welcome to WordPress. This is your first post. Edit or c
      "excerpt": "Welcome to WordPress. This is your first post. Edit or dele
      "date": "2009-11-11 12:50:19",
      "modified": "2009-11-11 12:50:19",
      "categories": [],
      "tags": [],
      "author": {
        "id": 1,
        "slug": "admin",
        "name": "admin",
        "first_name": "",
        "last_name": "",
        "nickname": "",
        "url": "",
        "description": ""
      },
      "comments": [
        {
          "id": 1,
          "name": "Mr WordPress",
          "url": "http:\/\/wordpress.org\/",
          "date": "2009-11-11 12:50:19",
          "content": "<p>Hi, this is a comment.<br \/>To delete a comment, ju
          "parent": 0
        }
      ],
      "comment_count": 1,
      "comment_status": "open"
    }
  ]
}
```


Figura 18. Formato de respuesta a una petición JSON. Fuente [36]

Como puede apreciarse, se nos proporciona en un array información relativa a la solicitud, en este caso al pedir un post tenemos información como la fecha en la que se publicó, el título del mismo, el autor, y como no, el contenido. Además también nos devuelve los comentarios que se han producido en el post.

Ya tenemos claro el protocolo a utilizar, vamos a ver cómo realizamos estas peticiones empleando Javascript, y más concretamente nuestro *framework* Dojo.

Examinando la librería dojo.io.script [35] vemos que nos proporciona un método dojo.io.script.get(jsonpArgs). Este método necesita como parámetro una variable en la

que debe ir contenida la información relativa a la petición que estamos realizando. A continuación en la figura 19 podemos ver la porción de nuestro código que se encarga de ejecutar el método get.

```
function peticion(tipo) {
    //vamos a crear el objeto de la peticion de la categoria pruebas
   var arg_tipo_pruebas = {
          url: "eo",
          callbackParamName: "callback",

          // con load cargamos los datos donde queremos
          load: function(datos){
          // en datos tenemos el objeto de la peticion, intentamos acceder a sus campos
           carga_datos(datos,tipo);
         },

        error: function(error){
            nodo.innerHTML = "An unexpected error occurred: " + error;
        }
  };

  if(tipo=="ocio"){
      arg_tipo_pruebas.url = "http://192.168.0.64/?json=get_category_posts&slug=ocio";
  }else{
     if(tipo=="academico"){
      arg_tipo_pruebas.url = "http://192.168.0.64/?json=get_category_posts&slug=academico";
      }
  }

    //una vez tenemos todos los parametros para la peticion la realizamos
    dojo.io.script.get(arg_tipo_pruebas);

}
```

Figura 19. Función petición encargada de establecer el formato para peticiones JSON.

Como podemos observar de la captura, para realizar la petición hemos creado una función denominada petición a la cual le pasamos como parámetro un String para distinguir entre si la petición es para los post de ocio o para los post de la categoría académico. En esta función lo primero que hacemos es crear la variable que pasamos como argumento al método get. La variable debe contener indispensablemente los atributos url, callbackParamName, load y error.

El parámetro url contiene la dirección de la petición JSON. En nuestro caso en este primer ejercicio solicitaremos los post en función de su categoría, para ello empleamos la función de la API JSON “get_category_post” y como puede apreciarse, en función del tipo de información que deseemos, modificamos el valor de la url a una dirección u otra, cambiando únicamente el *slug* de la petición.

Mediante el parámetro load indicamos qué acción queremos llevar a cabo una vez se nos ha devuelto la información desde el servidor, nótese que se realiza de forma síncrona, primero se recibe la información y una vez ha sido recibida se procede a

realizar la acción indicada en load. En nuestro caso llamamos a una función de nuestra creación denominada carga_datos a la cual le pasamos los datos que han sido recibidos como respuesta a la petición y en tipo le indicamos mediante un String si los datos pertenecen a ocio o a académico para mostrarlos una vez procesados en una vista o en otra.

Por último reseñar que mediante el parámetro error indicamos la acción a realizar si se produce algún fallo durante la petición o la recepción de datos.

Una vez hemos recibido la información del servidor debemos procesarla para distinguir entre sus contenidos y extraer aquellos que nos interesen. Esto es necesario ya que, como pudo apreciarse en la figura 18 en la que veíamos el resultado de la petición, ésta tiene numerosos campos. Vamos a ver a continuación el código de la función carga_datos y sobre dicho código explicamos cómo procesamos la información.

```
function carga_datos(datos,tipo){
            var nodo;
            if(tipo=="ocio"){
               nodo = dojo.byId("posta");
            }else{
               nodo = dojo.byId("postb");
            }
            //tenemos que transformar a formato json los datos que recibimos
            var json = dojo.toJson(datos,true);
           //ahora ya podemos crearnos un objeto empleando fromjson
            var obj = dojo.fromJson(json);
            //una vez tenemos el objeto podemos acceder a cualquiera de sus campos
            var contenido ="";

                for(i=0;i<=obj.posts.length;i++){

                     contenido = contenido + "<b> " + obj.posts[i].title  + " </b>" ;
                     contenido = contenido + obj.posts[i].content;
                     nodo.innerHTML = contenido;
                }

}
```

Figura 20. Función carga_datos encargada del procesado de la información.

Como vimos anteriormente, esta función recibe dos parámetros, los datos que se han obtenido de la petición al servidor y el tipo de petición que se realizó.

Para poder extraer el contenido de la información debemos realizar dos acciones, en primer lugar transformaremos la petición recibida a un String y en segundo lugar crearemos un objeto Javascript a partir de ese String. Este proceso es necesario ya que el resultado de la petición al servidor no es un objeto Javascript y por tanto no podemos

acceder directamente a sus contenidos. Mediante la función de Dojo toJson realizamos la serialización a formato JSON en forma de String de la petición y finalmente, mediante la función fromJson creamos el objeto Javascript a partir de un String con formato JSON.

Mediante el parámetro tipo sabemos a qué petición corresponden esos datos, para publicarlos en su lugar mediante la función dojo.byId encontramos el nodo del DOM en el que deben ir los contenidos. Si la petición es de tipo ocio debemos publicarlos en el recuadro con id “posta” y si pertenecen a académico los publicaremos en el espacio con identificador “postb”. Como a priori no sabemos cuántos post nos son devueltos, vamos almacenando en la variable contenido los diferentes post y los publicamos al final. Esto debemos hacerlo así puesto que, como se comentó con anterioridad, no podemos crear nodos del DOM de forma dinámica, puesto que todos los objetos deben pertenecer a un tipo de Dojo Mobile y no se puede indicar el tipo al crear un objeto dinámicamente. Volveremos sobre este problema en mayor detalle en el siguiente capítulo.

En las siguientes dos figuras podemos observar los resultados obtenidos de la aplicación:

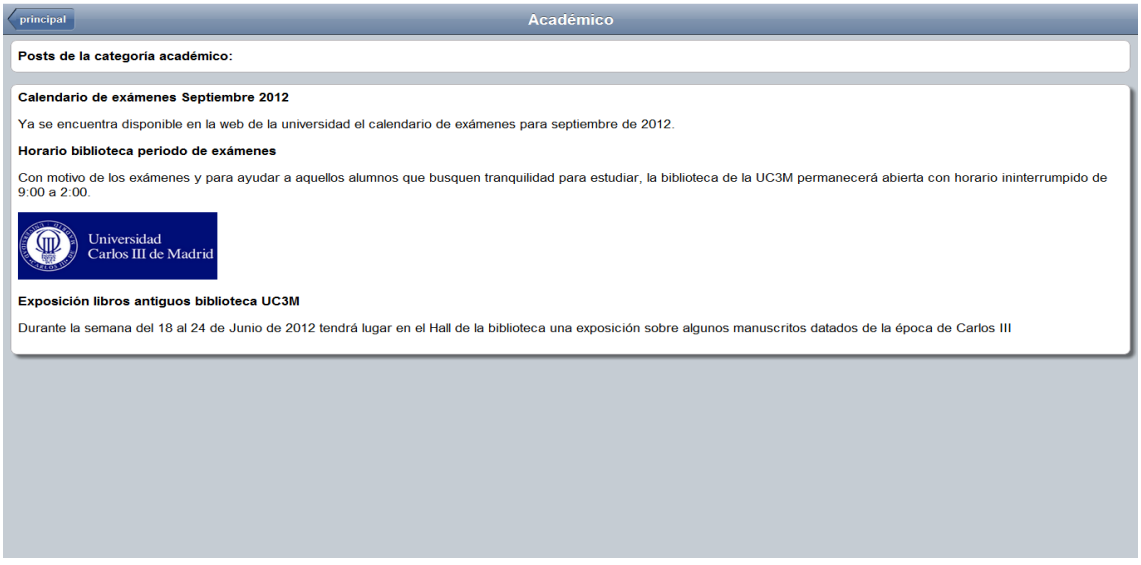

Figura 21. Vista académico con el contenido obtenido de la petición.

Figura 22. Vista ocio con el contenido obtenido de la petición.

Si observamos el contenido de las noticias podemos ver que, en ambas hay tres post y que en las dos aparece el mismo post, "Exposición libros antiguos biblioteca UC3M" esto es así ya que, cuando creamos los post en nuestro Wordpress indicamos que este post perteneciera a ambas categorías. Cabe destacar que, si recordamos, creamos un post que no pertenecía a ninguna de las dos categorías y como puede observarse el post no nos ha sido devuelto en ninguna de las dos peticiones por no pertenecer a ninguna de las dos categorías.

Tras este primer ejercicio hemos adquirido algunos conocimientos básicos sobre el funcionamiento de Wordpress, la forma de clasificar información que tiene y la forma en la que debemos solicitar información, así como su posterior procesado. Además hemos raspado la superficie de Dojo Mobile diseñando nuestra primera página y hemos comenzado a intuir de igual forma algunos posibles problemas que nos vamos a encontrar debido a su empleo. El siguiente paso a realizar es tratar con una página real basada en Wordpress, vamos a ello en el capítulo 3.

CAPITULO 3. Servidor Wordpress real, Movilforum

3.1 Aspecto inicial de la página

En este capítulo vamos a tratar con un servidor de Wordpress que se encuentra operativo de forma online para todos los usuarios. Vamos a trabajar con el servidor de la página www.espana.movilforum.com [37] para solicitarle los contenidos de las noticias, los eventos así como del blog (que a su vez se encuentra en una dirección diferente [38]). Además en nuestro diseño debemos tener en cuenta el aspecto de la página web para implementar en nuestra página un aspecto similar pero a su vez con características intrínsecas a aplicaciones móviles. La página web de Movilforum luce como puede apreciarse en la figura 23.

Figura 23. Aspecto general de la página www.espana.movilforum.com. Fuente [37]

Podemos apreciar a simple vista que tiene un diseño basado en tonos blancos y azules marino, además consta de una imagen que ocupa buena parte del contenido de la página y posee unos iconos característicos que podremos emplear en nuestro diseño.

3.1.1 Elección de elementos Dojo Mobile

Tenemos claro nuestro cometido y las herramientas que poseemos, vamos pues a analizar cómo podemos realizar la página web para que albergue nuestros contenidos y tenga funcionalidad parecida a una aplicación móvil. Cabe destacar que en este apartado trataremos la estructura de la página en general, y en el próximo apartado comentaremos como logramos mediante la hoja de estilo conseguir que tenga un aspecto similar a la página de Movilforum. Debemos recalcar en este punto un aspecto muy importante, nuestra página debe ser visible en todo tipo de dispositivos adaptándose a la resolución de pantalla y al tamaño de la misma siguiendo los principios de *responsive design*. Esto lo conseguimos añadiendo el meta viewport tal y como se nos indica en [2] y en [24].

Tal y como comentamos anteriormente, vamos a mostrar el contenido de las noticias, del blog y de los eventos. Para ello vamos a diseñar una vista principal en la que se muestren mediante un listado las tres opciones diferentes y pulsando en cada una de ellas nos desplazaremos a otras tres vistas denominadas Eventos, Noticias y Blog dentro de las cuales mostraremos el contenido referente a cada una de ellas. Esto lo conseguimos de igual manera que explicamos en el punto 2.2.1 luego no volveremos ahora sobre ello. Además, en la vista principal incluiremos dos imágenes que juntas conforman la imagen que copa la mayor parte de la página de Movilforum junto con el mismo icono que tienen en la página web a la izquierda de cada menú. Por último añadimos un recuadro que deberemos rellenar de espacios en blanco para el correcto funcionamiento en el navegador nativo de Samsung, no obstante volveremos sobre él en el punto 3.3.3 en el que nos referimos a los problemas encontrados. El código de la vista principal puede apreciarse en la figura 24.

```
<div id="principal" data-dojo-type="dojox.mobile.View" data-dojo-props="selected: true">
    <h1 data-dojo-type="dojox.mobile.Heading">Movil Forum   </h1>

    <img id="cabecera" class="imagenCabecera" data-dojo-type="dojox.mobile.ListItem" src="./pictures/cabecera.jpg" style="width:100%; height:auto"; />
    <img id="home" data-dojo-type="dojox.mobile.ListItem" src="./pictures/contenido_home.jpg"
     style="width:100%; height:auto"; />

    <ul data-dojo-type="dojox.mobile.RoundRectList" >

        <li data-dojo-type="dojox.mobile.ListItem" data-dojo-props="icon:'./pictures/eventos2.jpg', moveTo: 'eventos'">
            Eventos
        </li>
        <li data-dojo-type="dojox.mobile.ListItem" data-dojo-props="icon:'./pictures/noticias2.jpg', moveTo: 'noticias'">
            Noticias
        </li>

        <li data-dojo-type="dojox.mobile.ListItem" data-dojo-props="icon:'./pictures/blog2.jpg', moveTo: 'blog'">
            Blog
        </li>
    </ul>

    <div data-dojo-type="dojox.mobile.RoundRect" shadow="true">
       <div id="destacado" data-dojo-type="dojox.mobile.RoundRect" shadow="true"> </div>

    </div>

</div>
```

Figura 24. Código de la vista principal.

Respecto a las vistas de Eventos, Noticias y Blog han sido diseñadas las tres de igual manera. Teniendo en cuenta que nos encontramos frente a una aplicación móvil debemos lograr que sea lo más ligera posible y no muestre todos los contenidos de golpe para no abrumar al usuario. Es por este motivo que hemos optado por mostrar el inicio de los diferentes contenidos y añadimos un botón que al pulsarse mostrará el contenido entero. Para lograr esto hemos dividido las distintas vistas en cinco objetos dojox.mobile.RoundRect dentro de los cuales tenemos un espacio para mostrar el contenido de la noticia y justo después el botón para leer el contenido completo. Además, cuando el botón para leer la noticia completa haya sido pulsado lo ocultaremos. Por último, en la parte inferior de la vista contamos con dos botones, uno para pedir las noticias siguientes y otro para volver a mostrar las noticias anteriores, permaneciendo este último oculto en un principio y mostrándose si nos encontramos en la segunda página de noticias. También destacar que hemos añadido un objeto tipo <li> al principio a modo de título de la página al igual que podemos ver en Movilforum en tonos grisáceos. El aspecto de la vista de Noticias puede apreciarse en la figura 25, manteniendo una apariencia similar para el Blog y para los Eventos.

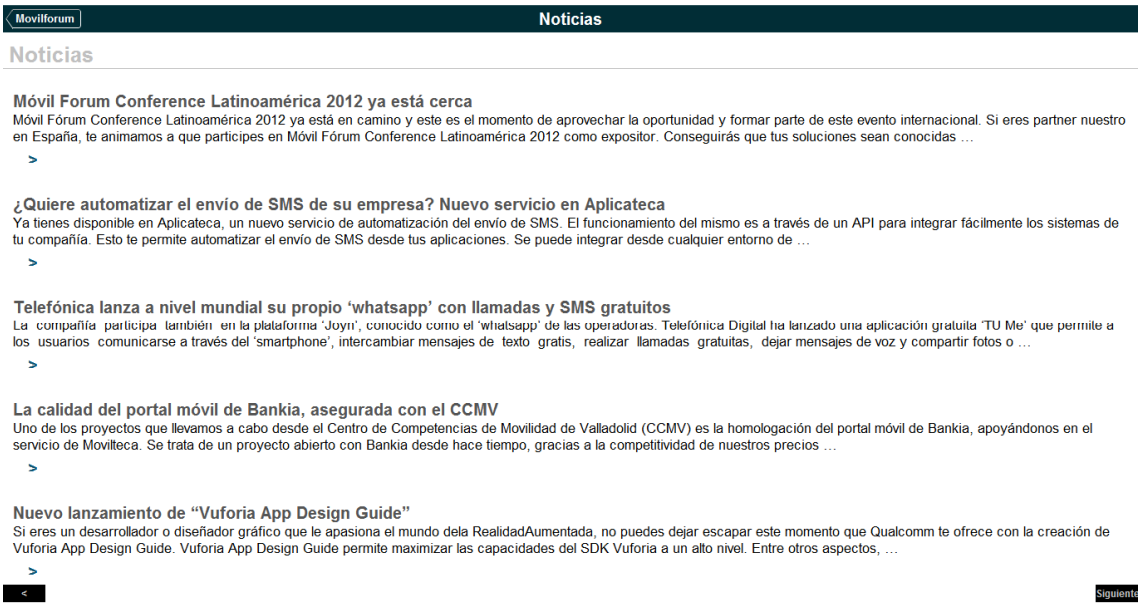

Figura 25. Aspecto de la vista Noticias.

En la figura 25 podemos observar varias cosas: las diferentes tonalidades que tienen los elementos según el tipo, el botón para volver a la vista principal, los dos botones al final de la página para solicitar más contenido y hasta el diferente tamaño y aspecto del título de la noticia con respecto al contenido. Como vemos todos estos elementos presentan un aspecto diferente, vamos a ver en el siguiente punto cómo conseguimos esto.

3.1.2 Creación de hoja de aspecto (CSS)

Si comparamos la figura anterior con las figuras 21 y 22 que nos mostraban el aspecto de nuestra primera aplicación en el capítulo 2, vemos que cambia su aspecto a pesar de la estructura ser similar. Esto es logrado mediante las diferentes hojas de estilo.

En este punto entra en consideración la tercera tecnología de la nos habíamos hecho eco en el capítulo uno de esta memoria, el CSS [39]. Es gracias a este lenguaje que podemos personalizar nuestra página hasta conseguir que sea justo de la forma en que nosotros deseamos. Mediante la hoja de estilos definimos el aspecto que queremos que tengan los diferentes objetos del DOM que hemos diseñado con *tags* HTML. Podemos crear clases y subclases para conseguir dotar a varios objetos de características similares, y a su vez hacer que uno de esos objetos en concreto se comporte de otra forma. Gracias al CSS podemos, además, identificar un único objeto del DOM mediante su id y asignarle el aspecto que nosotros queramos sin alterar lo más mínimo a otros que

comparten su tipo… En definitiva, es una herramienta realmente potente que con un buen uso nos permite justo lo que venimos buscando, dotar a nuestra página con un aspecto similar a una aplicación móvil.

Para entender mejor el funcionamiento de la hoja de estilos vamos a mostrar a continuación dos capturas. En la primera de ellas se puede apreciar nuestra página con la hoja de estilos que nos proporciona Dojo por defecto para los móviles Android y en la segunda captura observamos cómo podemos conseguir modificar el aspecto hasta conseguir que adquiera la apariencia que nosotros queremos sin más que modificar la hoja de estilo.

Figura 26. Aspecto de la vista principal con la hoja de estilos por defecto para Android.

Figura 27. Aspecto de la vista principal con hoja de estilos personalizada.

Si comparamos la figura 27 que está diseñada con nuestra propia hoja de estilos con la página web de Movilforum de la que pudimos observar su aspecto en la figura 23, vemos que hemos implementado el mismo aspecto que se ve en la página para los diferentes menús, poniendo la fuente de las letras en el mismo tono que en la página web, además tenemos una pequeña línea gris que separa cada menú al igual que en la web y, como comentamos anteriormente, mediante la conjunción de dos imágenes distintas logramos conformar una sola en la que se ve la imagen principal de la web así como el título de Movilforum.

Para lograr este aspecto en la vista principal hemos seguido algunos consejos de [40] hemos modificado en la hoja de estilos el campo body poniéndole como background el color blanco o #FFFFFF. Para que el título de la vista aparezca en blanco sobre fondo azul marino hemos cambiado en el elemento .mblHeading tanto el fondo como el color y tamaño de la fuente de las letras. Respecto a las imágenes para conseguir que se adapten perfectamente al tamaño de la pantalla en cada ocasión lo hemos realizado mediante la propiedad de dojo style indicándole width 100% y height auto. Por último reseñar que para los menús hemos tenido que modificar tres elementos, el primero es el que le dota de aspecto general y es el .mblRoundRectList, le hemos

indicado un margen de 15px a mano izquierda y cero en el resto, un padding de cero así como un fondo blanco y el color de las letras a #0d4f5b (un tono azul oscuro como puede apreciarse en la figura 27), los otros dos elementos que hemos modificado son el primer hijo y el último de .mblRoundRectList dejando su contenido vacío consiguiendo así que el primer elemento no tenga una línea por encima. Por último para tener la línea de color gris tal y como sucede en la página de Movilforum modificamos el elemento .mblListItem, en concreto su campo border-bottom y lo colocamos a silver.

Vamos a proceder a analizar ahora la vista de Noticias ya que las de Eventos y Blog son prácticamente idénticas en su apariencia. Como pudimos apreciar en la figura 25 en la vista noticias tenemos bastantes tipos diferentes de elementos. La cabecera ya tenemos diseñado su aspecto porque modificamos previamente el elemento .mblHeading para la vista principal. Dentro de la cabecera nos encontramos con la flecha para volver a la vista anterior que integramos mediante las propiedades de Dojo, para dotarle de ese aspecto característico modificamos el elemento .mblArrowButtonBody indicándole el tamaño del borde de 2px y de color blanco para poder distinguirlo entre el azul verdoso del fondo e indicándole a su vez que las letras deben ser de color blanco sobre fondo del mismo color que el elemento .mblHeading.

Respecto al título a modo de entradilla en gris que nos indica Noticias lo conseguimos añadiendo un elemento de lista ya que se le asocia directamente la clase .mblListItem que ya tenemos diseñada de la vista principal. Pero hay que notar aquí que, mientras que en la vista principal el color del texto era azul marino ahora el color de las letras es gris. Esto es posible gracias a una característica fundamental de css, el orden jerárquico. Cuando metemos el elemento <li> en el DOM le indicamos que además de ser de tipo dojox.mobile.listitem pertenece a la clase “cabecera” y en la hoja de estilos de css nos creamos esta clase cabecera y le indicamos que los elementos del tipo .mblListItemTextBox que se encuentren dentro de uno con la clase cabecera tengan de color para las letras el gris. Esto lo conseguimos definiendo la siguiente línea en la hoja de estilos:

```
.cabecera .mblListItemTextBox {
        color:silver;
        font-size: 26px
;}
```

Ya tenemos definida la cabecera y el título, vamos ahora a dar formato a los elementos que contendrán las noticias. Hemos dividido la vista en un <section> el cual en su interior alberga cinco <article> para el contenido de la noticia y un botón por cada una de ellas. También hay que destacar que, como sabemos, todo objeto que creemos en el DOM debe tener en sus propiedades el tipo de elemento Dojo Mobile que es, así pues hemos indicado que los elementos <article> pertenecen a dojox.mobile.RoundRect y el botón es un dojox.mobile.button.

A la hora de dotarles de estilo, el espacio para las noticias queremos que sea en blanco para poder visionar el contenido con claridad, además es así como se muestran las noticias en la página de Movilforum. Para ello hemos cambiado en la hoja de estilo el elemento .mblRoundRect y le hemos indicado que queremos el fondo blanco y que no deseamos que el borde sea visible, además es necesario quitarle el sombreado que trae por defecto, para ello modificamos .mblRoundRect .mblShadow y vaciamos su contenido. Por último para el botón hemos definido su propia clase denominada .leerMas, hemos tenido que definir una clase ya que tenemos diferentes tipos de botones que queremos dotar de un estilo u otro según su función. El botón de leerMas lo hemos diseñado idéntico a la página web de Movilforum, para ello le hemos indicado el fondo blanco y el color del texto (que será “>” simplemente) de color #105775.

El estilo del contenido lo creamos dinámicamente cuando añadimos las noticias, para ello mediante el atributo style [41] de HTML le indicamos para el título el color #555555 (gris oscuro), tipo de letra arial y tamaño 20px, para el contenido de la noticia dejamos el formato por defecto.

Para completar el aspecto de la vista únicamente nos quedan los dos botones al inferior de la pantalla que sirven para pedir los siguientes contenidos. Ambos botones son del tipo dojox.mobile.button pero como comentamos anteriormente para dotarles de un estilo propio debemos crearnos una clase nueva, esta clase se ha llamado .masnoticias y le hemos puesto un fondo negro y el color del texto tipo silver, así conseguimos que luzca como los botones habilitados para la misma función que hay en la página de Movilforum [37].

Con todas estas modificaciones conseguimos tener nuestra aplicación con el aspecto que deseamos, pero, únicamente se mostrará este aspecto si se consigue cargar la librería de Dojo, luego los navegadores que no soporten o no tengan habilitado

Javascript se encontrarán con una página con contenido HTML estático y sin formato. Luego no estamos cumpliendo aquí con el *graceful degradation*. En el capítulo cuatro veremos cómo solventamos este problema y cómo conseguimos dotar del mismo estilo a la página web tenga o no tenga habilitado Javascript.

3.2 Funcionalidad de la página

Una vez tenemos completado el aspecto de nuestra página vamos a acometer el código Javascript necesario para dotarle de funcionalidad, es decir, realizar las peticiones al servidor, procesar la información, publicarla en el lugar necesario, etc.

Nótese que en este punto, un usuario accediendo a la web desde un navegador o dispositivo sin Javascript no podrá ni visionar correctamente la página web (ya que la hoja de estilos es cargada dinámicamente al parsear la página mediante el método parse de Dojo), ni tampoco tendrá ninguna funcionalidad ya que para su funcionamiento la página necesita el empleo de Javascript.

Dicho esto, hay que destacar que un usuario cuyo navegador no soporte Javascript en ningún momento podrá disfrutar de la página completamente operativa, pero en el capítulo 4, una vez tengamos nuestra aplicación totalmente operativa, introduciremos algunas modificaciones en la hoja de estilo para dotar a la página de un aspecto visualmente atractivo a pesar de no contar con Javascript, y del mismo modo, avisaremos al usuario mediante un mensaje que la página requiere del uso de Javascript para funcionar.

3.2.1 Excerpt. Importancia y utilización

Como debemos tener en mente durante toda la memoria, nos encontramos diseñando una página web que debe poder visionarse en todo tipo de dispositivos, cumpliendo así con el *responsive design*, pero fundamentalmente nuestro cometido es orientar este diseño hacia dispositivos móviles tales como smartphones o tablets. Es por este motivo que adquiere importancia el excerpt [42].

Tal y como sabemos, en un dispositivo móvil el espacio se encuentra mucho más limitado que en un ordenador de mesa o incluso que en un portátil por pequeño que sea éste. Es debido a esto que, si cargamos directamente todo el contenido de las noticias (o

del blog o de los eventos), la pantalla del usuario se llenará de información obligándole al mismo a hacer *scroll* para llegar al contenido que le interesa. Por esta razón hemos optado en nuestro diseño por mostrar en primer lugar al usuario el título y un pequeño extracto del contenido (lo que se denomina excerpt) y le hemos añadido un botón cuya funcionalidad es permitir al usuario leer el contenido completo.

Hay que darnos cuenta no obstante, que no todas las páginas realizadas con Wordpress tienen habilitado o relleno el campo excerpt. En nuestro caso comentaremos posteriormente como al realizar la petición desde un navegador de ordenador tipo Chrome o Firefox el campo excerpt se nos devuelve vacío en el caso de la petición del blog, mientras que para noticias o eventos este campo sí se encuentra relleno.

3.2.2 Peticiones JSON y procesado

En esta segunda aplicación, a diferencia de la primera, explicada en el capítulo dos, en la que únicamente solicitábamos el contenido de dos categorías diferentes, debemos distinguir entre numerosos tipos de peticiones en función de lo que deseemos. Además, tenemos variables globales para almacenar el contenido, en la variable obj guardamos las noticias, en la variable evento los eventos del año y en la variable obj_blog almacenamos la información de los post. También, debido a los problemas con el excerpt del blog hemos tenido que crear una variable denominada post que almacena la información sobre un post en concreto.

El funcionamiento general es el siguiente, cuando se carga la página pedimos la información referente a las noticias, a los eventos y al blog. Una vez recibimos esa información, decodificamos su contenido y lo almacenamos en las variables mencionadas anteriormente. Cabe destacar que, como respuesta a la petición recibimos toda la información, en ella ya se encuentra el contenido completo de las noticias y de los eventos (no así del blog), simplemente nosotros únicamente mostramos inicialmente el excerpt por los motivos que se comentaron en el punto anterior, pero toda la información se encuentra almacenada en las variables globales.

Con esto, cuando el usuario pulse sobre el botón de leer más sobre una noticia o un evento, lo que tenemos que hacer es cargar el contenido completo que se encuentra almacenado en la variable global correspondiente (los post del blog funcionan de manera diferente y lo comentaremos en el punto 3.3.1).

También debemos tener claro en cada momento qué página de noticias o del blog quiere leer el usuario, es por eso que contamos con dos variables globales denominadas pag_noticias y pag_blog que nos indican la página que el usuario desea cargar en cada momento, estas variables globales se controlarán desde el manejador de los botones de siguientes noticias que se encuentran al final tanto de la vista Noticias como de la vista Blog.

Una vez tenemos claro el funcionamiento general de nuestro código, vamos a entrar en más detalle a explicar las funciones que lo llevan a cabo, relatando cómo pedimos el contenido de las noticias y de los eventos. El tratado del blog lo comentaremos posteriormente en otro apartado por ser ligeramente diferente.

Para realizar las peticiones al servidor contamos con una función denominada petición que recibe como parámetros el tipo de petición que realizamos (si es noticias, eventos, blog en general o un único post del mismo), otro parámetro llamado num que nos ayudará a tratar las peticiones de un único post y por último un tercer parámetro, id_espacio, que utilizaremos para identificar posteriormente el objeto del DOM en el que debemos publicar el contenido en cada caso.

Al igual que comentamos en el punto 2.2.3 para que funcione la petición debemos crear un objeto que contenga los parámetros url, callbackParamName, load y error. Es mediante la url que distinguimos el contenido que queremos pedir.

Cuando queremos pedir por las noticias lo realizamos como puede verse en la figura 28.

```
if(tipo=="noticias"){
    if(pag_noticias==1){
      arg_tipo_pruebas.url =  "http://espana.movilforum.com/category/noticias/?json=get_recent_posts&count=5";
    }else{
      //si no es la primera peticion tenemos que acceder a diferentes url donde pag_noticias es la pagina que queremos cargar
      arg_tipo_pruebas.url =  "http://espana.movilforum.com/category/noticias/page/"+pag_noticias+"/?json=get_recent_posts&count=5";
    }
}
```

Figura 28. Formato de la url para pedir noticias.

Como puede apreciarse en la figura 28 lo primero que hacemos para modificar la url es identificar el tipo de petición que queremos hacer, una vez vemos que queremos las noticias identificamos mediante la variable global pag_noticias, cual es la página por

la que estamos preguntando ya que, si solicitamos una página distinta de la primera, la url cambia y debemos emplear esta variable global para pedir la página que deseamos.

Respecto a los eventos la url que debemos solicitar es un poco diferente a la petición de noticias, ya que, para poder gestionar los eventos desde nuestra aplicación, se creó un objeto específico en la página web de movilforum denominado ai1ec_event. La url en concreto por la que debemos pedir es la siguiente:

http://espana.movilforum.com/api/get_recent_posts/?post_type=ai1ec_event&orderby=ai1ec_event_date&order=desc

Lo primero que notamos es que esta es una petición tipo explícita que usa directamente la API de JSON [36], debemos emplear este tipo de petición para poder indicarle que queremos que nos devuelva todos los post del tipo ai1ec_event (ya que así se identifican los eventos) y además le indicamos como parámetros que nos los ordene por fecha y en orden descendente.

Una vez hemos recibido la respuesta a la petición mediante el parámetro load, indicamos a nuestro código como proceder, en nuestro caso llamamos a una función de nuestra creación denominada carga_datos y le pasamos como parámetros los datos que han sido recibidos como respuesta a la petición, el tipo de la petición y el identificador del espacio donde debe publicarse el contenido.

En la función carga_datos lo primero que hacemos es, mediante las funciones dojo.toJson y dojo.fromJson procesamos la información obtenida y almacenamos en las variables globales correspondientes la información decodificada. Una vez tenemos la información traducida publicamos su contenido en el espacio correspondiente que identificamos mediante el parámetro id_espacio. Puede apreciarse el código en el que tratamos las noticias en la figura 29.

```
if(tipo=="noticias"){
    for(i=0;i<=obj.posts.length;i++){
        contenido = contenido + "<b style='font-family:arial;color:#555555;font-size:20px;line-height:1.3em'> " + obj.posts[i].title  + " </b>" + "<br>";
        cadena = obj.posts[i].excerpt;
        //tenemos que quitar los últimos cincuenta caracteres para que desaparezcnm el Leer mas que viene puesto y el dibujo
        cadena =  cadena.substring(0,cadena.length-50);
        contenido = contenido + cadena ;
        dojo.byId("noticia"+i).innerHTML = contenido;
        contenido = "";
        cadena = "";
    }
}
```

Figura 29. Código para publicar las noticias en el DOM.

En primer lugar identificamos el tipo de petición, una vez sabemos que es de noticias iteramos mediante un for por todo su array. En cada iteración publicamos el contenido dotándole de su propio estilo para el título como puede apreciarse en la primera línea del for. Posteriormente debemos eliminar unos cuantos caracteres del excerpt puesto que trae una imagen y su propio link leer más el cual, si lo pulsamos, nos redirigiría a la página web de origen y no es lo que pretendemos con nuestra aplicación. Por último reseñar que identificamos el elemento del DOM sobre el que vamos a publicar el contenido mediante el método dojo.byId en el cual le pasamos como parámetro “noticia” seguido de la i que valdrá en cada iteración del for de 0 a 4, así pues el identificador completo será en cada caso noticia0, noticia1 … hasta noticia4. Evidentemente mediante el código HTML tenemos creados estos elementos con ese mismo nombre como identificador.

Hemos explicado como realizamos las peticiones al servidor y cómo procesamos su información mediante la función carga_datos, pero esta función también es llamada cuando queremos leer el contenido completo de una noticia o de un evento. Cuando pulsamos el botón de leer más invocamos a esta función pasándole como parámetros en tipo “post_noticias” o “post_evento” y en identificador de espacio el correspondiente entre 0 y 4 (ya que cargamos 5 por defecto siempre). Cuando esto sucede, en carga_datos identificamos el tipo de petición y publicamos el contenido almacenado en la variable global correspondiente en el nodo adecuado. Nótese que tanto para las noticias como para los eventos cuando el usuario desea leer el contenido entero no tenemos que realizar otra petición al servidor porque cuando pedimos las noticias ya nos llega la información entera. No obstante no sucede del mismo modo con el blog así

pues, vamos a comentar en el siguiente punto como tratamos las peticiones del tipo blog.

3.3 Dificultades y limitaciones

A lo largo de la realización de este proyecto nos hemos encontrado, como no podía ser de otra forma, con algunas dificultades que hemos tenido que trabajar de manera especial. Estas han tenido diferentes orígenes, desde temas relacionados con las peticiones al servidor como particularidades de Dojo Mobile y de la necesidad de mantener el sincronismo de la página.

En este punto vamos a tratar las incidencias relacionadas con el funcionamiento online. Veremos en el capítulo cuarto como dotamos de persistencia a la aplicación y como trabajamos con la base de datos lo cual nos ocasionará también algún problema, pero lo trataremos posteriormente. Englobaremos en este punto aspectos de diferente índole tales como el funcionamiento no del todo correcto del plugin JSON que instalamos previamente en el Wordpress, las limitaciones de Dojo Mobile o el comportamiento peculiar de la página en algunos dispositivos distintos.

3.3.1 Dificultades en el comportamiento de blog.movilforum.com

Como comentamos anteriormente, el blog de movilforum [38], que es accesible desde la url www.blog.mobilforum.com, no presentaba relleno el campo excerpt en los post, luego nos veríamos obligados a mostrar todo el contenido al usuario, rompiendo así una de las características clave de nuestro proyecto, el no abrumar al usuario con contenidos que no desea ver.

Para solventar esto, instalamos un plugin en el Wordpress del blog de movilforum que identifica el *user agent* mediante el cual se está accediendo a la página web y si detecta que es un dispositivo móvil, formatea la página y le muestra una versión reducida al usuario. En esta versión reducida se enseña al usuario el excerpt y se le facilita la posibilidad de leer el contenido completo.

Así pues, ya tenemos el excerpt en nuestros post si accedemos desde un dispositivo móvil, no obstante si estamos accediendo desde un ordenador o un ipad el

campo excerpt sigue vacío, para estos casos optamos por mostrarle al usuario únicamente el título del post en cuestión.

El segundo problema que nos plantea el blog viene a causa de cómo gestiona las peticiones el *plugin* de JSON. Este *plugin* no gestiona la información, si no que nos devuelve el contenido tal cual se muestra si se accede a la página web en cuestión. Es decir, si se accede desde un dispositivo móvil a la dirección www.blog.movilforum.com [38] se nos muestran los diferentes post con su excerpt, pero si queremos leer el contenido de un post concreto, al pinchar sobre el botón, nos redirige a otra dirección distinta. Esto, unido al funcionamiento de JSON provoca que, cuando solicitamos los post del blog, únicamente nos devuelva el excerpt del mismo y la url dónde se encuentra almacenado el contenido completo, pero, a diferencia de las noticias y los eventos, la petición inicial no nos devuelve el contenido entero de los post. Luego, cuando un usuario quiera leer el contenido completo de un post debemos realizar una solicitud a parte para dicho post al servidor.

Además debemos reseñar que tuvimos que realizar algunas modificaciones en el plugin por dos motivos, en el excerpt únicamente se mostraban 40 caracteres lo cual es realmente escaso, y además el plugin realizaba paginación y como JSON nos devuelve lo que se muestre en la página, si solicitamos el contenido completo de un post nos devolvía únicamente el link a la primera página del post y si queríamos leer más debíamos acceder de nuevo a un link externo para poder continuar leyendo. Tras las modificaciones, ahora el excerpt devuelve un número aceptable de caracteres y ya no realiza paginación luego ya podemos trabajar con él.

Una vez tenemos claro que debemos solicitar los post uno a uno si se quiere leer el contenido completo, vamos a explicar como lo hemos llevado a cabo en nuestro código.

Cuando pedimos los post para mostrar su excerpt en nuestra función petición comprobamos el tipo de petición y una vez sabemos que solicitamos el blog debemos comprobar, al igual que hacíamos con las noticias, si estamos pidiendo por la página inicial o por alguna posterior. Esto lo controlamos mediante la variable global pag_blog que será modificada desde el manejador del botón que se encarga de pedir los siguientes post. En la figura 30 puede apreciarse lo que hemos comentado.

```
if(tipo=="blog"){

  if(pag_blog==1){
    arg_tipo_pruebas.url = "http://blog.movilforum.com/?json=get_recent_posts";
  }else{
    arg_tipo_pruebas.url = "http://blog.movilforum.com/page/"+pag_blog+"/?json=get_recent_posts";
  }
}
```

Figura 30. Formato de la url para petición post del blog.

Si por el contrario queremos leer el contenido completo de un post, debemos pedir por la url del post en concreto. Esta url la conseguimos una vez hemos solicitado los post del blog, así pues se encuentra almacenada en la variable global obj_blog. Esta variable contiene un array con la información de todos los posts que nos han sido devueltos tras la primera petición, luego debemos indicarle mediante el parámetro num que mencionamos cuando explicamos la función petición, cual es el post concreto que estamos solicitando. Dicho parámetro valdrá entre 0 y 4 y se lo indicaremos desde el manejador del botón leer más (trataremos el comportamiento de estos manejadores en el siguiente punto). Además, debemos añadirle a la url nuestra petición JSON para identificar el tipo de petición que estamos realizando. Puede apreciarse como realizamos esto en la figura 31.

```
if(tipo=="post_blog"){
  arg_tipo_pruebas.url =obj_blog.posts[num].url+"?json=get_recent_posts";
}
```

Figura 31. Formato de url para la petición de un único post.

Una vez hemos recibido la información procedente del servidor la procesamos mediante la función carga_datos. Si la petición es del tipo blog almacenamos la información en la variable global obj_blog y si es de un post concreto la almacenamos en la variable global post. Una vez tenemos decodificada la información, rellenamos el DOM del mismo modo que hacíamos con las noticias y los eventos. Recorremos mediante un for el array con todos los post, identificamos el nodo del DOM en el que debemos publicar la información mediante la función dojo.byId, damos formato al título, eliminamos el leer más que trae la petición por defecto y rellenamos el DOM con el contenido. Puede verse el código en la figura 32.

```
if(tipo=="blog"){
    //Rellenamos el espacio del blog con los post
    for(i=0;i<=obj_blog.posts.length;i++){
        contenido = contenido + "<b style='font-family:arial;color:#555555;font-size:20px;line-height:1.3em'> " + obj_blog.posts[i].title  + " </b>" + "<br>";
        cadena = obj_blog.posts[i].excerpt;
        cadena = cadena.substring(0,cadena.length-9);
        //tenemos que quitar los últimos nueve caracteres para que desaparezca el Leer mas que viene puesto
        contenido = contenido + cadena;
        nodo = dojo.byId("post"+i);
        nodo.innerHTML = contenido;
        contenido = "";
        cadena = "";
    }
}
```

Figura 32. Procesado de la información de los post del blog.

Si la petición era de un post en concreto, además del contenido completo, mostramos el autor y la fecha de publicación del mismo con un formato en tonos grisáceos similar al que puede apreciarse en la página del blog de movilforum. Puede observarse el formato tanto del título como del autor y la fecha en la figura 33.

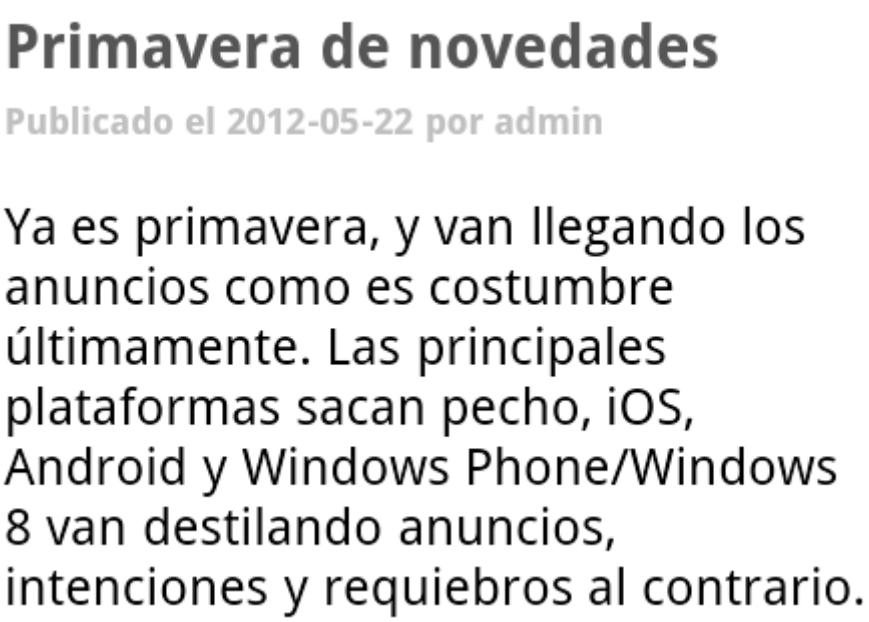

Figura 33. Ejemplo del formato de un post del blog.

Llegados a este punto tenemos claro como realizamos las peticiones al servidor, cómo nos devuelve la información y cómo procesamos esta información y la publicamos en el DOM. Nos falta por explicar cómo gestionamos cuando un usuario solicita leer el contenido completo de una noticia, evento o post o cuando quiere que se le muestren las siguientes noticias o post. Esto es gestionado mediante los botones que añadimos en la página.

3.3.2 Utilización únicamente de objetos Dojo Mobile

En este apartado vamos a explicar las limitaciones que nos produce tener que diseñar nuestra página únicamente con objetos de tipo Dojo Mobile. Como vimos en el punto 2.2.1 todo elemento de nuestra página web debe tener asignado un tipo de objeto de Dojo Mobile indispensablemente. Esto es debido a que si no se realiza así, al intentar parsear la página mediante el método parse los nodos HTML que no se encuentren asignados a un tipo de Dojo Mobile quedarán sin asignar produciendo un error que impedirá todo funcionamiento de la página, desde la navegación entre las diferentes vistas, hasta pedir la información al servidor, etc.

Esto nos fuerza a crear con anterioridad a recibir las noticias el espacio para almacenar las mismas. Como se puede apreciar de la página web de movilforum, tanto las noticias como los eventos como los post del blog se muestran de cinco en cinco, luego debemos crear cinco objetos de tipo dojox.mobile.RoundRect para almacenar el contenido de cada vista.

Además, esta característica de Dojo Mobile, nos impide crear nodos de HTML de forma dinámica mediante Javascript, ya que, aunque podemos crear los *tags* de HTML de igual manera que en una página no basada en Dojo Mobile, no tenemos forma de indicarle mediante Javascript al nodo qué tipo de elemento de Dojo Mobile es, con lo cual al parsear no funcionaría la página porque desconocería ese nodo.

Otro aspecto que tenemos que comentar, es que los botones de Dojo Mobile no admiten el manejador de eventos que nos proporciona Dojo por defecto, luego debemos controlar si se han pulsado o no y la acción a realizar mediante la propiedad onclick que nos proporciona Dojo Mobile a tal efecto [43]. Esto no ocasiona mayor problema ya que su comportamiento es el adecuado, pero sería deseable y más intuitivo para el observador ajeno a la aplicación mantener el principio de separación entre el aspecto de la página (los *tags* de HTML) y el funcionamiento de la misma, acorde a lo que dicta el patrón MVC [17], pero de este modo nos vemos forzados a tener que romper este principio y a añadir el código para controlar las acciones sobre el botón en la parte de código de los *tags* HTML.

En la captura de la figura 34 podemos observar en mayor detalle lo comentado en el párrafo anterior. Podemos apreciar como estamos creando y reservando el espacio para la primera noticia mediante el RoundRect y como colocamos el botón debajo de

este espacio. Lo deseable sería realizar con HTML únicamente esta estructura y posteriormente añadir mediante Javascript el código encargado de manejar las pulsaciones sobre el botón, pero como hemos comentado, si se intenta realizar así Dojo Mobile rompe su funcionalidad. Así pues, debemos, mediante la propiedad onclick indicarle qué queremos hacer si se pulsa el botón. En la figura 34 podemos apreciar el caso del botón de leer más de una noticia, en ese caso, llamamos a la función carga_datos, pasándole como parámetros la posición que ocupa esa noticia en el array que contiene la variable global obj (que recordemos contiene la información devuelta de la petición al servidor), como tipo de petición le indicamos post_noticias, que identifica una sola noticia, y como tercer parámetro indicamos un número que nos sirve para identificar el nodo del DOM en el que debemos publicar el contenido. Además, una vez se ha pulsado el botón de leer más lo ocultamos hasta que el usuario solicita otra página distinta de noticias.

```
<!-- Ventana de noticias -->
<div id="noticias" data-dojo-type="dojox.mobile.View">
        <h1 data-dojo-type="dojox.mobile.Heading" data-dojo-props="back:'Movilforum', moveTo:'principal'">Noticias</h1>
    <ul data-dojo-type="dojox.mobile.RoundRectList">
            <li class="cabecera" data-dojo-type="dojox.mobile.ListItem" >
                Noticias
            </li>
        </ul>


  <div data-dojo-type="dojox.mobile.RoundRect" shadow="true">
  <div id="noticia0" data-dojo-type="dojox.mobile.RoundRect" shadow="true"> </div>
  <button id="bn0" class="leerMas" data-dojo-type="dojox.mobile.Button" data-dojo-props='label:">",
    onClick:function(e){
    dojo.byId("bn0").style.visibility = "hidden";
    carga_datos(0,"post_noticias",0);
    return true;
   }'></button>

  </div>
```

Figura 34. Código para la creación de los elementos y manejador del botón leer más.

Debemos reseñar que el botón leer más para los eventos se comporta exactamente igual que el de noticias, simplemente a la función carga_datos le indicaremos como tipo de petición post_evento. Pero en el caso que solicitemos leer un post del blog entero, como comentamos anteriormente, debemos pedir al servidor su contenido. Para ello en vez de llamar a la función carga_datos llamamos a la función petición pasándole como parámetros el número del post que estamos pidiendo, la petición de tipo post_blog y el identificador para el nodo.

Por último vamos a comentar como tratamos las peticiones de siguiente página de noticias o del blog, como se comentó con anterioridad, contamos con dos botones destinados a tal fin, tal y como puede apreciarse en la figura 23 nos encontramos con

dos botones en el fondo de la página, el de la izquierda vale para pedir los post más recientes, luego sólo lo haremos visible si nos encontramos en de la segunda página para arriba, y el botón de la derecha que vale para solicitar los post anteriores, luego siempre permanecerá visible. Cuando pulsamos cada uno de estos botones, hacemos visibles todos los botones de leer más que se encuentran en la misma vista, para poder solicitar mediante los mismos leer de nuevo el contenido completo de las siguientes noticias que se vayan a cargar. Además hacemos *scroll* al principio de la página para ahorrarle al usuario el trabajo de realizar el *scroll* manualmente hasta arriba para leer las noticias siguientes. Puede observarse el código de este tipo de botones en la figura 35.

```
<button class="masnoticias" data-dojo-type="dojox.mobile.Button" style="float:right" data-dojo-props='label:"Siguiente >", onClick:function(e){
 pag_noticias=pag_noticias+1;
 peticion("noticias");
 window.scrollTo(110,0);

 dojo.byId("not_anteriores").style.visibility = "visible";
 for(i=0;i<5;i++){
    dojo.byId("bn"+i).style.visibility = "visible";
 }
 return true;
 }'></button>

 <button id="not_anteriores" class="masnoticias" data-dojo-type="dojox.mobile.Button" style="visibility:hidden" data-dojo-props='label:"< Anterior",
    onClick:function(e){
    dojo.byId("not_anteriores").style.visibility = "visible";
    pag_noticias=pag_noticias-1;
    peticion("noticias");
    window.scrollTo(110,0);
    for(i=0;i<5;i++){
      dojo.byId("bn"+i).style.visibility = "visible";
    }

    if(pag_noticias==1){
      dojo.byId("not_anteriores").style.visibility = "hidden"

    }

 return true;
 }'></button>
```

Figura 35. Código de los manejadores de los botones siguientes noticias y noticias anteriores.

3.3.3 Problemas visualización navegador nativo Samsung

El último punto de incidencias que debemos reseñar es un problema de visualización que hemos detectado en algunos dispositivos de la marca Samsung. Hay que notar que éste problema sólo se ha producido en algunos modelos, no todos, de la marca Samsung y que sólo ocurre con el navegador nativo del dispositivo, si se instalan otros navegadores de los disponibles en el market de Android basados en webkit como Dolphin Browser, la página se visiona sin problemas. También ha sido testeada en numerosas marcas y versiones de Android tales como HTC, SonyEricsson o Motorola de diferentes compañías telefónicas y ha funcionado sin ningún tipo de incidencia. Por último indicar que ha sido testeada en dispositivos con el sistema operativo iOS tales como iphones o ipads y se ha visionado perfectamente.

El problema es el siguiente, como hemos comentado en el capítulo 2, la aplicación se basa en diferentes vistas con transiciones animadas entre cada una de ellas. En el navegador nativo de los dispositivos Samsung, esta animación se produce sin problemas, pero, una vez finalizada ésta, al intentar el usuario hacer *scroll* sobre la página se produce un solapamiento entre la vista anterior y la actual resultando un efecto óptico algo molesto. Cabe reseñar que el efecto se produce mientras el usuario realiza el primer *scroll*, una vez que su dedo deja de hacer contacto con la pantalla, el navegador parece notificar que se ha producido la transición entre las vistas y el contenido se visiona perfectamente.

Es por eso que para paliar este problema hemos tenido que descargar la clase View.js de Dojo Mobile [44] para realizar algunas modificaciones sobre su código. Dicha clase nos proporciona una función denominada onAfterTransitionIn (moveTo, dir, transition, context, method) mediante la cual podemos identificar por el parámetro moveTo qué transición se estaba llevando a cabo ya que dicho parámetro lleva el nombre de la vista a la que nos estamos moviendo. Para paliar el problema lo que hacemos es hacer un pequeño *scroll* automáticamente para que el navegador de Samsung se dé cuenta de que se ha producido una transición y deje visible la vista deseada y no superponga la anterior.

Esta es la idea principal, pero para llevarla a cabo tenemos que tener en cuenta un par de consideraciones. En nuestra vista principal, únicamente tenemos la foto y una lista que no nos ocupa el largo entero de la página, luego cuando intentamos hacer *scroll*

no nos deja puesto que no hay contenido suficiente para realizar el *scroll*. Es por eso que nos hemos visto obligados a añadir un nuevo nodo en la vista principal del DOM y rellenarlo de espacios en blanco para poder hacer el *scroll*. La segunda consideración que tenemos que tener en cuenta es que si nos encontramos en el borde de la página, hacemos una transición hacia otra vista y mediante nuestro código le decimos que vuelva a arriba de la página, es como si no hiciéramos *scroll* y el problema persiste, es por ello que detectamos si estamos haciendo la transición hacia la vista principal o hacia otra distinta y según el caso hacemos *scroll* hacia puntos diferentes de la página. Puede apreciarse con más claridad esto en el código que se ve en la figura 36.

```
onAfterTransitionIn: function(moveTo, dir, transition, context, method){
    // summary:
    //      Stub function to connect to from your application.
    // description:
    //      Called after the arriving transition occurs.
      if(moveTo=="principal"){
         window.scrollTo(3,10);
      }else{
        window.scrollTo(3,3);
      }

},
```

Figura 36. Modificaciones realizadas sobre el código de la clase View.js.

Tras todo lo visto en el capítulo tres nos encontramos con nuestra aplicación perfectamente operativa siempre y cuando tengamos conexión. Si nos encontramos sin conexión y tratamos de acceder a la página web, se nos mostrará un mensaje indicándonos un error. Como sabemos esta página web está destinada a hacer frente a las aplicaciones móviles, y dichas aplicaciones deben funcionar tanto con conexión como sin ella. Vamos a proceder en el capítulo 4 a indicar como dotamos a nuestra página de funcionalidad offline.

CAPITULO 4. Persistencia

4.1 Importancia de la persistencia

En este capítulo vamos a tratar un aspecto fundamental del proyecto, sin el cual nuestra página no podría compararse a una aplicación móvil. Este aspecto es la persistencia, definida en [45] como “la característica o estado del proceso que sobrevive al proceso que la creo”. Sin esta capacidad, el estado sólo existiría en la RAM y se perdería cuando la RAM perdiera la alimentación, como por ejemplo, cuando se apagara el ordenador. En nuestro caso queremos que los datos de nuestra aplicación sobrevivan al proceso del navegador y que se encuentren disponibles la próxima vez que se abra nuestra página.

Como comentamos en el punto 1.2.1, la persistencia puede definirse a nivel de memoria, a nivel de aplicación o a nivel de objeto. En nuestro diseño nos interesa la persistencia a nivel de aplicación que es la que nos permite almacenar el contenido y que se encuentre disponible entre dos sesiones diferentes. Como nos indica [11] “Este tipo de persistencia requiere que los datos sean almacenados en un medio secundario, no volátil, para su posterior reconstrucción y utilización, por lo que su tiempo de vida es independiente del proceso que los creó”.

Cabe destacar que la persistencia es de vital importancia en nuestra página ya que, al estar orientada al mundo móvil, la conexión no siempre estará disponible para el usuario, y pueden incluso producirse intervalos en los que el usuario tenga conexión y pierda la cobertura. Nuestra página debe ser capaz de detectar todo eso y funcionar correctamente en todos los casos posibles. Además, si pretendemos comparar nuestra página web con las aplicaciones nativas debemos, al menos, tratar de equiparar su funcionalidad. Así pues el funcionamiento offline se nos presenta indispensable.

Hemos visto por qué la persistencia es importante en nuestra aplicación, pero, ¿cómo podemos lograrla con HTML5? Observando el punto 5.1 de [46] que nos habla sobre las nuevas APIs que se han introducido en HTML5 respecto a sus anteriores versiones, vemos que HTML5 nos proporciona dos nuevas APIs para cumplir con nuestro cometido, estas son la de application cache y la de localstorage que nos permite

almacenar información en el cliente. Veremos con más detenimiento estas dos en los siguientes apartados.

4.2 Acceso offline a la página

Vamos a comentar en este punto como logramos que el usuario pueda acceder a la página si se encuentra sin conexión. Esto lo conseguimos mediante el cacheo de nuestra página web por su navegador. Evidentemente, para poder cachear los contenidos, el usuario debe haber accedido al menos una vez teniendo conexión a nuestra página para poder descargarla a su dispositivo y cachear las librerías y el código necesario.

El permitir el cacheo de la página por parte del usuario es posible mediante una de las dos librerías que comentamos en el punto anterior, en concreto la librería de application cache.

4.2.1 Application cache

Mediante application cache vamos a conseguir dos objetivos, el primero y fundamental es que el navegador sea capaz de mostrar nuestra página web aún cuando él mismo se encuentra sin conexión, pero además, mediante application cache permitimos una carga más rápida de nuestro contenido ya que el navegador compara un fichero que tiene almacenado con el que hay online y si encuentra algún cambio descarga de nuevo los contenidos, pero si no lo hay (como ocurrirá la mayoría de las veces) muestra el código de la página que tiene almacenado.

Es muy importante distinguir que, mediante application cache lo que conseguimos es almacenar en el dispositivo del usuario el código de nuestra página y podemos cachear los elementos que el usuario ha visionado, tales como imágenes etc, **pero no tenemos almacenado el contenido para procesarlo**. Cuando indicamos que no se producirán cambios, nos referimos a que no cambiaremos el código de nuestra página web a no ser que se detecten fallos o se produzcan nuevas versiones del mismo y es éste código el que se almacena en el dispositivo. Así, logramos con application cache que el usuario no tenga que descargarse más contenido que el de las noticias, etc que él quiera leer, ahorrándole así tiempo y tarifa de datos.

No obstante, en el punto siguiente comentaremos más aspectos sobre la application cache y sobre cuánto tiempo permanece almacenado el contenido en memoria.

Una vez tenemos clara la importancia y la fortaleza de application cache, vamos a ver cómo podemos implementarla. Tal y como vemos en [47], para poder usar application cache debemos crearnos un fichero en el que le indicamos el contenido que queremos que cachee y además, debemos introducirle a nuestro fichero html en la cabecera que mire ese fichero, esto lo hacemos añadiéndole al *tag* html lo siguiente:

<html manifest = “cache.manifest”>

Podemos observar un ejemplo de fichero cache manifest en la figura 37.

```
CACHE MANIFEST
/clock.css
/clock.js
/clock-face.jpg
```

Figura 37. Ejemplo fichero cache manifest. Fuente [30].

Mediante ese fichero indicamos al navegador que, después de descargarlos, cachee los elementos clock.css (hoja de estilo), clock.js (código javascript) y clock-face.jpg (imagen). Una vez el navegador haya cacheado esos elementos, si recargamos la página, el browser comparará de nuevo con ese fichero y al observar que ya tiene cacheados esos ficheros no los volverá a descargar, ahorrando así al usuario tiempo de espera y datos descargados.

Además, al fichero de caché se le pueden añadir diferentes secciones. En la sección network le indicamos todos aquellos archivos que no deben ser cacheados y, si queremos mostrar algún otro archivo en lugar de los de la sección network, lo indicaremos mediante una sección fallback. Por último, destacar que si a la sección network le indicamos el carácter especial *, nuestra página se encargará de cachear todos aquellos contenidos que el usuario visione durante su periodo online, tales como fotos, artículos, etc, incluso cuando pertenezcan a un dominio diferente al de nuestra aplicación.

Con esto tenemos claro que nuestro fichero cache.manifest debería tener en el apartado general las imágenes que queramos guardar, así como nuestro propio código y

muy importante, **las librerías de Dojo necesarias para dotar de funcionalidad a nuestra página.**

Es por este motivo que nos hemos encontrado con problemas en los dispositivos de iOS, como se puede observar en [48], los dispositivos que emplean Safari (y quizá otro tipo de navegadores) presentan problemas de sincronización con la forma que tiene Dojo de descargar los datos. Esto es debido a que Dojo no es totalmente asíncrono, empleando XMLHttpRequest (XHR [49]) para descargar los datos y en el navegador Safari, el proceso de cacheado y el de descarga de contenidos es simultáneo, lo cual se traduce en errores.

Para solventar este problema, hemos optado por descargar a nuestro servidor las librerías de Dojo que empleamos en nuestro código [44], y le indicamos a Dojo que emplee el modo asíncrono para descargarlas desde nuestro servidor. De este modo evitamos realizar peticiones XHR que son las que nos inducen a errores.

Como necesitamos bastantes clases Dojo para hacer funcionar nuestra aplicación, si tuviéramos que indicar una por una, el cacheo en nuestro archivo cache.manifest alcanzaría unas dimensiones muy grandes, por lo que hemos optado por optimizar el código indicando mediante php que cachee todos los archivos disponibles en el directorio. Así, introducimos únicamente en el directorio los archivos que necesiten ser cacheados. Además hemos empleado la propiedad del * del apartado network como comentamos anteriormente para indicar a nuestra web que cachee todas las imágenes que visione el usuario, así se encontrarán disponibles cuando acceda sin conexión.

Como hemos utilizado código php para recorrer todo nuestro directorio, nuestro archivo de caché pasará de tener la extensión .manifest a la extensión .php, debiendo indicárselo a nuestra página con el cambio del *tag* html como vimos anteriormente, por:

<html manifest = “cache.php”>

También, debemos indicar en nuestro código que busque las librerías en nuestro servidor, así pues, ya no usamos como en los capítulos 2 y 3 la API de Google para descargar la librería, si no que indicamos que busque los archivos en la ruta local como puede verse en la figura 38.

```
<link id="hoja" rel="stylesheet" href="movilforum.css">

  <script src="./dojo/dojo.js" data-dojo-config="async: true"></script>
  <script src="./View.js"></script>
  <script src="./persistence.js"></script>
  <script src="./persistence.store.sql.js"></script>
  <script src="./persistence.store.websql.js"></script>
```

Figura 38. Archivos cargados localmente.

Como podemos ver, estamos cargando la hoja de estilo denominada movilforum.css, la librería de Dojo, la clase View a la cual le hemos hecho algunas modificaciones, y los archivos denominados persistence corresponden a un framework que nos ayudará a gestionar la base de datos como veremos posteriormente en el punto 4.3. Hay que observar también que, como comentamos anteriormente, debemos indicar a Dojo que emplee el modo asíncrono para evitar los problemas de Safari, así al emplear el método de Dojo require descargará los archivos correctamente.

Para dar por finalizado este punto, mostramos en la figura 39 el contenido del fichero cache.php que será el encargado de indicar que ficheros debe cachear nuestro navegador.

```
<?php
  header('Content-Type: text/cache-manifest');
  echo "CACHE MANIFEST\n\n";
  echo "CACHE:\n";
  $hashes = "";
  $dir = new RecursiveDirectoryIterator(".");
    //si queremos que la aplicacion cachee de nuevo aumentamos el numero de version
    // (si ve que ha cambiado algo en el fichero cachea, si cache.php se mantiene constante no cachea de nuevo)
    // version 1
  foreach(new RecursiveIteratorIterator($dir) as $file) {
    if ($file->IsFile() &&
        $file != "cache.php" &&
        substr($file->getFilename(), 0, 1) != ".")
    {
      echo $file . "\n";
      $hashes .= md5_file($file);
    }
  }
  echo "\nFALLBACK:\n";

  echo "NETWORK:\n";
  echo "*\n\n";
  echo "# Hash: " . md5($hashes) . "\n";

?>
```

Figura 39. Contenido del fichero cache.php.

4.2.2 Volatilidad de la caché

Una vez tenemos almacenado el contenido mediante el fichero de caché, debemos tener en cuenta dos aspectos relativos a la volatilidad de la misma.

El primero de ellos es que el usuario puede borrar en cualquier momento la caché de su navegador y nuestra página dejará inevitablemente de funcionar offline. Esta es una característica intrínseca de HTML5 frente a la que no podemos hacer nada y será comentada de nuevo en el capítulo 5 cuando hablemos de las debilidades de HTML5.

El segundo aspecto es, una vez que el usuario ha cacheado el contenido de la página, cómo le indicamos si necesita volver a descargarse algún contenido porque ha sido actualizado, etc. Como comentamos en el punto anterior, una vez que el navegador ha cacheado la información necesaria, comparará su fichero de caché con el disponible en el servidor, si no se han producido cambios en este fichero, el navegador no descargará de nuevo los ficheros y librerías y trabajará con los que tiene cacheados. Así, la solución más sencilla para controlar cuando queremos que los usuarios vuelvan a descargar los contenidos consiste en modificar el archivo de caché.

Como puede apreciarse en la figura 39, mediante un comentario le indicamos al fichero cache.php una versión, de este modo, sin más que modificar esta línea de comentario, el navegador verá que el fichero ha cambiado su contenido y procederá a volver a bajar los contenidos y a cachearlos de nuevo.

A lo largo de este apartado 4.2 hemos indagado sobre cómo podemos conseguir que nuestro código no sea descargado cada vez mediante la application cache, así pues, nuestra aplicación ya se encontraría accesible offline, pero los contenidos no se pueden guardar mediante este tipo de almacenamiento, es por eso que debemos recurrir a la segunda API que vimos en el punto 4.1 que nos proporciona HTML5 para almacenar contenidos, el local storage.

4.3 Almacenamiento de contenido en el navegador

Como discutimos en el punto 1.2.1, HTML5 nos proporciona diversos métodos de almacenar nuestro contenido en el navegador del dispositivo tales como *Web Storage* y *Web SQL Database*. En nuestro caso elegimos *Web SQL Database* por encontrarse basado en SQL y ser, por tanto, más potente que el almacenamiento tipo *Web Storage* que se basaba en un par de parejas clave-contenido.

En este punto vamos a ver como implementamos este tipo de almacenamiento mediante código Javascript. Tal y como comentamos anteriormente, los *frameworks* han dotado a Javascript de una enorme utilidad y nos solventan numerosos errores de difícil detección, es por esto que buscamos un *framework* que nos proporcione librerías para el almacenamiento de datos mediante *Web SQL Database*.

4.3.1 Base de datos. Persistence.js

El *framework* que hemos encontrado se denomina Persistence.js y, como podemos ver en [50], nos permite implementar cuatro tipos diferentes de almacenamiento: Google Gears, MySQL, *localstorage* y el que nos interesa HTML5 *Web Storage Database.*

Además, se nos indica que el *framework* persistence.js da soporte en navegadores basados en *WebKit* como Chrome o Safari, en Firefox (a través de Google Gears), en Opera y que han sido testeados con resultados satisfactorios en Android desde su versión 1.6 y en iOS a partir de la versión 3.

Una vez elegido el *framework* que nos va a ayudar con nuestra labor de almacenar el contenido, vamos a ver cómo ponerlo en funcionamiento.

El primer paso, como vimos en la figura 38, es indicar a nuestro navegador dónde puede encontrar las librerías necesarias para hacer funcionar el almacenamiento, en nuestro caso las descargamos localmente y se denominan persistence.js, persistence.store.sql.js y persistence.store.web.sql.js.

Tras descargar las librerías y una vez han sido reconocidas por la página, vamos a ver cómo creamos la base de datos y cómo almacenamos contenidos en ella.

4.3.1.1 Almacenamiento en la base

El primer paso es crear el tipo de variable que vamos a almacenar en nuestra base de datos, como nos indica [51] tenemos diferentes tipos de datos que podemos almacenar en la base, desde String hasta int, pasando incluso por objetos JSON. En nuestro caso hemos creado un tipo de variable para las noticias, otro para el blog, otro para los eventos, otro para los casos de éxito (que hemos añadido en la aplicación final a diferencia del capítulo 3) y un último tipo de datos encargado de almacenar la fecha de la última actualización para mostrarla cuando estemos sin conexión. Puede apreciarse en la figura 40 cómo definimos estos tipos de datos para la base.

```
var Noticias = persistence.define('Noticias', {
                name: "TEXT",
                contenido: "TEXT",
                excerpt: "TEXT",
                titulo: "TEXT"
});
var Blog = persistence.define('Blog', {
                name: "TEXT",
                contenido: "TEXT",
                excerpt: "TEXT",
                titulo: "TEXT",
                //los post ademas tienen fecha y autor
                fecha: "TEXT",
                autor: "TEXT"
});
var Eventos = persistence.define('Eventos', {
                name: "TEXT",
                contenido: "TEXT",
                excerpt: "TEXT",
                titulo: "TEXT"
});
var Soluciones = persistence.define('Soluciones', {
                name: "TEXT", //definira el tipo de solucion
                contenido: "TEXT",
                titulo: "TEXT"
});
var Fecha = persistence.define('Fecha',{
                dia: "INT",
                mes: "INT",
                ano: "INT"
});
```

Figura 40. Definición tipos datos.

Como se ve de la figura, cada tipo de datos contiene un campo name que será su identificador, el campo contenido será donde almacenemos el contenido completo de la noticia, el campo título que es para el título de la misma. Además el tipo blog contiene

espacio para el autor y la fecha de publicación del mismo. El tipo de datos Fecha es el que empleamos para saber la fecha de la última actualización del contenido, para indicárselo al usuario en el caso de que éste se encuentre offline.

Si observamos detenidamente, los tipos de datos Noticias, Blog y Eventos podrían haber sido un mismo tipo, pero los hemos dividido en tres para evitar tener que realizar una búsqueda iterativa y poder realizar la búsqueda mediante el tipo de dato. Ya que ésta es una de las ventajas de *Web SQL Database* y no íbamos a desaprovecharla.

Una vez hemos definido todos los tipos de datos que se almacenarán en la base de datos, vamos a explicar cómo conseguimos guardarlos en nuestra base. Para el almacenamiento hemos creado una función denominada basedatos(). En esta función lo primero que hacemos es inicializar la base de datos como se nos indica en [33], para ello añadimos la siguiente parte de código:

```
if(base_inicializada==0){
    //inicializamos base de datos
    persistence.store.websql.config(persistence, 'datos', 'Base datos de la aplicacion', 5 * 1024 * 1024);
}
if(conexion==1){
    if(base_inicializada==0){
        //borramos lo que hubiera en la base
        persistence.reset();
        base_inicializada=1;
    }
```

Figura 41. Inicialización base datos.

Mediante la línea config declaramos la base de datos, el primer argumento debe ser persistence, el segundo argumento es el nombre de la base de datos, el tercero es una descripción de la base, y el cuarto argumento, muy importante, es el que indica el tamaño que se debe almacenar para la base de datos. El tamaño es de 5 * 1024 * 1024, equivalente a 5Mb, que es el máximo almacenamiento que permiten los dispositivos iOS. Una vez declarada la base, la reseteamos para borrar todo su contenido, es por eso que sólo realizamos esta acción si nos encontramos con conexión, ya que si lo hacemos sin conexión borraríamos todo el contenido almacenado y no podríamos mostrarle nada al usuario.

Una vez creada la base de datos, procedemos a almacenar el contenido, para eso empleamos las líneas de código mostradas en la figura 42.

```
persistence.schemaSync(function(){
    dojo.byId("cargando").innerHTML = "Almacenando en base de datos";
    for(i=0;i<obj.posts.length;i++){
        //Creamos las noticias, los eventos y los post que queremos meter
        var noticia = new Noticias();
        noticia.name = "Noticia " + i;
        noticia.contenido = obj.posts[i].content;
        noticia.excerpt = obj.posts[i].excerpt;
        noticia.titulo = obj.posts[i].title;
        //con add los vamos metiendo, pero hasta que no hacemos flush no pasan a la base de datos
        persistence.add(noticia);
    }
    persistence.flush(function(){
    //mediante flush los volcamos a la base
    });

});
```

Figura 42. Almacenamiento en base de datos.

En la figura 42 mostramos únicamente el almacenamiento de las noticias, el resto de los contenidos se almacenan de igual forma. En primer lugar mediante la función persistence.schemaSync indicamos a la base que vamos a almacenar datos. En la captura puede apreciarse además cómo mostramos un mensaje indicando que se está almacenando el contenido en la base de datos. Tras esto, procedemos a iterar sobre la variable global en la que tenemos almacenadas las noticias y para cada una, nos creamos una variable, indicándole el tipo de dato Noticias y rellenamos sus campos a partir del array de la variable global. Una vez rellenados todos los campos, mediante el método persistence.add añadimos las diferentes noticias a la base de datos, y tras rellenar todas ellas, llamamos al método persistence.flush que procede a volcar los datos sobre la base. **Es muy importante notar como el llamamiento a las funciones se produce de forma anidada para mantener el sincronismo**.

Como hemos visto, hemos sido capaces de crear una base de datos, de especificarle el tamaño, de indicar los diferentes tipos de datos que van a componer la misma y posteriormente de almacenar el contenido que queremos mediante funciones sencillas que no ha sido necesario implementar, lo cual es un ejemplo muy bueno de la fuerza que une a Javascript y a los *frameworks* y cómo ambos salen beneficiados de esta unión como comentamos con anterioridad en el punto 1.2.2 en el que empezábamos a introducir Dojo.

Una vez tenemos almacenado el contenido que queremos, vamos a ver cómo podemos leer de la base de datos el mismo cuando nos encontremos offline.

4.3.1.2 Lectura de la base

Una vez tenemos creada la base de datos y llena de contenidos, podemos utilizarla para mostrarle al usuario los contenidos cuando se encuentra en modo offline.

El primer paso es detectar mediante Javascript si estamos trabajando con conexión o sin ella. Para ello, intentamos cargar desde Google una imagen. Después, mediante el método onerror identificamos si hubo algún error, lo cual indicará que no hay conexión o por el contrario, mediante el método onload, sabremos que la imagen se cargó con éxito y por tanto, sabremos que la conexión se encuentra activa. Para saber en todo momento si tenemos conexión o no, cambiamos el valor de una variable global que creamos a tal efecto.

Una vez hemos discernido si nos encontramos con conexión o sin ella, iniciamos nuestro programa mediante una llamada a la función programa que es la encargada de emplear el método dojo.require, de realizar las peticiones, etc. En la figura 43 mostramos el código mediante el cual detectamos la conexión.

```
//comprobamos si hay o no conexion y ponemos la variable en concordancia
var imgsrc = 'http://www.google.es/intl/en_com/images/logo_plain.png';
    var img = new Image();

    img.onerror = function () {
        conexion = 0;
        //una vez sabemos si hay conexion iniciamos el programa
        programa();
    }
    img.onload = function () {
        conexion = 1;
        //una vez sabemos si hay conexion iniciamos el programa
        programa();
    }

img.src = imgsrc;
```

Figura 43. Detección de conexión.

Tras detectar si el dispositivo tiene conexión, en el caso de que no la tenga, llamamos a la función rellenaDOM que es la encargada de leer de la base de datos y mostrar el contenido de las diferentes noticias, post del blog, etc. En el punto 4.5.1, en el que comentaremos los fallos que se han producido y las soluciones tomadas, veremos que estamos obligados a llamar a rellenaDOM tanto si hay conexión como si no, pero ya especificaremos más adelante los pormenores de esta función.

En la función rellenaDOM extraemos el contenido de las noticias, de los post del blog, de los eventos, y de las soluciones o casos de éxito que tengamos almacenados y mediante la función dojo.byId identificamos el elemento del DOM y publicamos el contenido adecuado. En la figura 44 mostramos el código de esta función cuando lee las noticias. Dicho código es similar al empleado para los post del blog y los eventos.

```
function rellenaDOM(){
    //con esta funcion cargamos el contenido desde la base de datos
       var contenido="";
       var cadena="";
       var contador =0; //es para identificar el elemento del DOM donde meter el contenido
       persistence.store.websql.config(persistence, 'datos', 'Base de datos de la aplicacion', 5 * 1024 * 1024);
       Noticias.all().list(function(noticias){
           //IMPORTANTE CONTAR CUANTOS NOS DEVUELVEN PARA ITERAR
           var contadorNoticias = noticias.length;
           //Una vez hemos extraido los datos iteramos sobre ellos
           noticias.forEach(function(noticia){
               contenido = contenido + "<b style='font-family:arial;color:#555555;font-size:20px;line-height:1.3em'> " + noticia.titulo  + " </b>" + "<br>"
               cadena = noticia.excerpt;
               cadena =  cadena.substring(0,cadena.length-50);
               contenido = contenido + cadena ;
               dojo.byId("noticia"+contador).innerHTML = contenido;
               contador = contador + 1;
               contenido = "";
               cadena = "";
               //Comprobamos si ya hemos iterado sobre todos
               if(--contadorNoticias == 0){
                   //Una vez hemos modificado todos los volvemos a subir a la base de datos
                   contador = 0;
                   persistence.flush(function(){
                       //volcamos datos
                   });
               }

           });
       });
}
```

Figura 44. Código función rellenaDOM.

Lo primero que hacemos en esta función es definir la base de datos que vamos a emplear mediante la función config, le indicamos la misma base que tenemos creada mediante la función basedatos. Tras esto, sacamos todas las noticias mediante la función Noticias.all.list (vemos aquí lo comentado anteriormente de que con Web SQL Database podemos filtrar las búsquedas por contenido), nos creamos una variable para saber cuántas tenemos almacenadas y posteriormente iteramos sobre cada una de ellas publicando su contenido en el nodo del DOM correspondiente, en el momento en que hemos extraído información de todas las noticias, mediante el método flush las volcamos de nuevo a la base de datos.

Mediante los puntos 4.3.1.1 y 4.3.1.2 hemos mostrado los métodos que nos proporciona persistence.js para trabajar con la base de datos. En el siguiente punto vamos a comentar cómo empleamos nosotros estas funciones, de que nuevos añadidos hemos dotado a nuestra página para la versión final respecto a las anteriores y las

modificaciones que hemos tenido que realizar para la correcta sincronización de toda la web.

4.4 Versión final de la aplicación. Modificaciones y añadidos

En este punto vamos a comentar, por un lado, los nuevos menús que hemos añadido para completar la página, obteniendo su versión definitiva, ya que hasta ahora los capítulos dos y tres nos han servido para ir acercándonos a la versión final de nuestra aplicación, sin presentar todo su contenido. Por otro lado, indicaremos todos aquellos cambios que hemos tenido que realizar para sincronizar la base de datos con las peticiones y para el correcto funcionamiento de la página de manera online como de forma offline.

4.4.1 Elementos añadidos al DOM

En el capítulo 3 vimos que nuestra página web tenía el contenido de las noticias, de los post del blog y de los eventos. Pero, si comparamos con la figura 23 en la que mostrábamos la página web de movilforum [37] vemos que ésta tenía más menús, en esta última versión de la aplicación hemos añadido también los menús de soluciones y casos de éxito. También hemos modificado la vista principal, sustituyendo la imagen de las personas sentadas en una mesa por algo más minimalista, más acorde con una aplicación móvil.

Por último, señalar que hemos añadido un elemento para indicar al usuario el proceso de actualización que está siguiendo la página. El contenido de este elemento será modificado dinámicamente en función de lo que se encuentre haciendo la aplicación en este momento.

Los diferentes mensajes que mostrará este elemento son:

- Cargando. Cuando se accede por primera vez a la página y descargamos el contenido.
- Almacenando en base de datos. Este mensaje lo mostraremos una vez la información ha sido descargada y estemos almacenando en la base de datos.

- Actualizado + fecha. Cuando se ha descargado el contenido y se ha almacenado en la base, indicamos mediante este mensaje que ya se ha actualizado.
- Última actualización + fecha. Cuando se accede a la aplicación y ya se tiene información guardada, se indica mediante este mensaje cual fue la fecha de la última actualización.
- Error en la base de datos. Este mensaje se mostrará únicamente si se ha producido algún error al intentar guardar los datos en la base de datos.

En la figura 45, mostrada a continuación, puede verse el aspecto de la página principal de nuestra aplicación tras estas modificaciones.

Figura 45. Vista principal. Captura desde móvil Android.

Como puede apreciarse en la figura anterior, esta es la vista principal desde un móvil Android. Vemos la imagen que hemos utilizado como cabecera, justo debajo el mensaje que nos indica que se han actualizado correctamente los contenidos y, tras este, la lista de los diferentes contenidos, en la que, hemos añadido dos nuevos elementos a la lista, las soluciones y los casos de éxito. En las figuras 46 y 47 mostramos la vista de las soluciones y de los casos de éxito.

Figura 46. Vista Soluciones. Captura desde móvil Android.

Figura 47. Vista reducida casos de éxito. Captura desde móvil Android.

Como puede observarse en las figuras 46 y 47, hemos empleado los mismos iconos que emplea www.espana.movilforum.com [37]. Además hemos optado por modificar el icono de back para volver a la vista anterior, ahora tiene un aspecto más móvil y algo más intuitivo al indicar home. Hay que indicar también, que la figura 47 muestra una versión reducida de la vista casos de éxito puesto que tiene un menú más extenso, pero sirva esta captura para hacernos una idea de su aspecto.

Una vez hemos indicado los nuevos añadidos a la aplicación, vamos a ver las modificaciones que hemos introducido en nuestra web a nivel funcional.

4.4.2 Necesidad de descargar todo el contenido antes de almacenar.

En este apartado vamos a ver cómo sincronizamos las peticiones con la base de datos. Cabe destacar que Javascript no espera por la ejecución de una función antes de comenzar con la ejecución de la siguiente. Es por este motivo que debemos ser nosotros los encargados de gestionar que se haya recibido respuesta a todas las peticiones antes de intentar grabar el contenido en la base de datos ya que, si no hemos recibido respuesta a la petición realizada, al intentar guardar en la base de datos, intentaremos acceder al array que nos devuelve la petición y nos producirá un error.

En nuestro diseño hemos optado por realizar las peticiones de las noticias, del blog y de los eventos nada más producirse el acceso del usuario a la página web. Sin embargo, para el caso de los menús soluciones y casos de éxito, únicamente solicitaremos su contenido si el usuario accede a ese menú. La decisión de hacerlo de esta forma se debe a una razón de optimización ya que, generalmente, las noticias, el blog y los eventos son los elementos más visitados, frente a los casos de éxito y las soluciones, en los que cada usuario sólo estará interesado en alguno de ellos en particular. Si cargáramos todos de inicio, este hecho se traduciría en una espera mayor para el usuario y en una descarga de datos innecesaria ya que en la mayoría de las ocasiones el usuario no estará interesado en ese contenido. Como puede verse, una vez más, seguimos el principio de minimizar las esperas para el usuario y la descarga de contenido que es vital en el mundo móvil. Además, almacenaremos en la base de datos para mostrarle al usuario de forma offline únicamente las cinco primeras noticias, los cinco primeros post y los eventos. Esto lo hacemos así de igual manera para evitar almacenar en el dispositivo del usuario demasiado contenido.

Vamos a proceder a explicar, en primer lugar, cómo realizamos las peticiones de las noticias, los eventos y los post del blog y cómo los sincronizamos con la base ya que es algo que realizamos siempre que tengamos conexión, y posteriormente indicaremos con detalle cómo gestionamos las soluciones y los casos de éxito.

En la primera función, denominada programa, llamamos a la función petición indicándole como tipo noticias y posteriormente eventos. Una vez que hemos recibido las peticiones de las noticias y de los eventos, realizaremos desde la función carga_datos la petición para los post del blog. Debemos controlar mediante una variable global si ya tenemos las noticias y los eventos antes de realizar la petición de los post del blog. Esto lo tenemos que realizar así debido a que no sabemos si el servidor nos proporcionará respuesta antes para los eventos o para las noticias. La variable global utilizada es la llamada peticiones_resueltas. Esta variable se incrementará en uno cuando recibimos respuesta para las noticias y para los eventos, así pues en el caso de que valga 2 significará que ya hemos recibido respuesta a las noticias y a los eventos y podemos proceder a pedir los post del blog.

Tras haber recibido respuesta para los post del blog, ya tenemos el contenido completo de las noticias, el contenido final de los eventos y el excerpt de todos los post del blog, tras lo que llamamos a la función basedatos que, como comentamos en el punto 4.3.1.1 es la encargada de almacenar el contenido en la base de datos.

Pero además de mostrar el excerpt de los post al usuario cuando acceda sin conexión, queremos poder mostrarle el contenido completo, lo que nos obliga a realizar, además, peticiones directamente a la url del post en concreto para descargar el contenido entero. Como sabemos, cada página en la versión móvil del blog contiene 5 post, así pues, deberemos realizar esta petición cinco veces. Realizamos la primera petición desde este punto y las siguientes las realizaremos desde el caso post_blog de la función carga_datos.

En la figura 48 mostramos el código de la función carga_datos para el caso de los post del blog que hemos explicado en el párrafo anterior.

```
case  "blog":
    //Rellenamos el espacio del blog con los post
    for(i=0;i<obj_blog.posts.length;i++){
        if((obj_blog.posts[i].title!=null)&&(obj_blog.posts[i].excerpt!=null)){
            contenido = contenido + "<b style='font-family:arial;color:#555555;font-size:20px;line-height:1.3em'>"+obj_blog.posts[i].title +"</b>"+"<br>";
            if(obj_blog.posts[0].excerpt!= null ){
                cadena = obj_blog.posts[i].excerpt;
                cadena = cadena.substring(0,cadena.length-9);
            }
            contenido = contenido + cadena;
            nodo = dojo.byId("post"+i);
            nodo.innerHTML = contenido;
            contenido = "";
            cadena = "";
        }
        if(i==4&&peticion_inicial==1){
            //cuando tenemos todas las url de los post, pedimos su contenido para almacenarnos en la base de datos
            //solo si es la primera vez, si no, no guardamos los post en la base. Empezamos pidiendo el primero y una vez procesado este pediremos
            //los siguientes
            basedatos();
            peticion("post_blog", 0,0);

        }
    }
break;
```

Figura 48. Código función carga_datos para blog.

Asimismo, en el código puede apreciarse que comparamos con la variable global peticion_inicial, esto es así porque, como comentamos anteriormente, únicamente almacenaremos los post correspondientes a la primera página del blog, luego si nos encontramos pidiendo la página dos o posteriores del mismo, la variable peticion_inicial valdrá 0 y ya no llamaremos a la base de datos para almacenar la información.

Ahora vamos a pasar a explicar cómo realizamos las peticiones de los post para almacenar su contenido completo. Hemos visto que, tras recibir las noticias, los eventos y los post, su contenido era almacenado en nuestra base mediante la función basedatos, pero para los post, nos falta conocer cómo almacenar el contenido completo ya que la petición inicial no nos lo devuelve como ya sabemos.

Una vez hemos recibido respuesta a la petición a la url de un post concreto, extraemos los post del blog de la base de datos, comparamos los títulos de los almacenados con el que nos ha devuelto el servidor y, una vez encontrado, rellenamos su contenido completo y volcamos todos de nuevo a la base de datos. Una vez hemos terminado de trabajar con la base de datos, solicitamos el siguiente post al servidor y repetimos el proceso.

Tras haber rellenado el contenido de los 5 post completos en la base de datos, modificamos la variable peticion_inicial, comentada anteriormente, y ponemos su valor a cero para que las siguientes veces que realicemos peticiones de cualquier tipo no tratemos de almacenar más contenido en la base de datos. Además mostramos los botones de leer más, de pedir siguientes noticias, etc para permitirle al usuario

interactuar con el contenido de la página. Esto lo realizamos en este punto para evitar que el usuario interactúe antes de tener el contenido correctamente almacenado en la base porque si no podría producir errores debido al sincronismo con la base de datos.

A continuación mostramos, en la figura 49, la parte del código que se encarga de realizar lo comentado en el párrafo anterior.

```
if(peticion_inicial==1){
//si es la primera vez que se carga la pagina almacenamos el contenido del post del blog en la base de datos
//extraemos los post de la base, modificamos el que corresponda y los volvemos a introducir.
//esto es necesario puesto que cuando se piden todos los post, solamente nos es devuelto el excerpt, pero no todo su contenido
Blog.all().list(function(blogdatos){
    var contadorBlog = blogdatos.length;
    blogdatos.forEach(function(unpost){
        //comprobamos el titulo del post de la base con el que nos ha llegado de la peticion cuando lo encontremos rellenamos los campos
        if(unpost.titulo==post.posts[0].title){
            contadorBlog = 1;
            unpost.contenido = post.posts[0].content;
            unpost.fecha = post.posts[0].date;
            unpost.autor = post.posts[0].author.name;
        }
        if(--contadorBlog == 0){
            //Una vez hemos modificado todos los volvemos a subir a la base de datos
            persistence.flush(function(){
                //volcamos los datos
            });
            //pedimos el siguiente post, debemos hacerlo desde aqui para cerciorarnos que se piden los post de uno en uno una vez almacenado
            //el anterior en la base, si no, se pierde el sincronismo con la base de datos
            id_espacio=id_espacio+1;
            if(id_espacio<=4){
                peticion("post_blog",id_espacio,id_espacio);
            }else{
                peticion_inicial=0;
                dojo.byId("cargando").innerHTML = "Actualizado "+date.getDate()+"/"+(date.getMonth()+1)+"/"+date.getFullYear();
                for(i=0;i<5;i++){
                    dojo.byId("bn"+i).style.visibility = "visible";
                    dojo.byId("bl"+i).style.visibility = "visible";
                    dojo.byId("be"+i).style.visibility = "visible";
                }
            }
        }
    });
});
```

Figura 49. Código función carga_datos para post_blog.

Como puede apreciarse en el código, cuando tenemos todo el contenido disponible, indicamos en el nodo del DOM habilitado para tal efecto, la fecha de la última actualización. Para ello utilizamos la variable global llamada date y los métodos de Javascript getDate() que nos indica el día del mes, getMonth() que nos indica el mes de 0 a 11 (con lo cual debemos sumarle uno) y por último la función getFullYear() que nos devuelve con cuatro dígitos el año.

Llegados a este punto tenemos almacenados en la base de datos el contenido completo de todos los elementos. Vamos a explicar ahora como hacemos para mostrar al usuario el contenido total de una noticia cuando pulsa el botón leer más.

Con el manejador del botón leer más, controlaremos si tenemos conexión y si estamos mirando la primera página de contenidos (con lo cual el contenido se encontrará almacenado en la base de datos porque es lo que pedimos inicialmente) o bien estamos solicitando el contenido completo de una noticia de las páginas posteriores, en cuyo caso el contenido se encontrará almacenado en la variable global al igual que en el ejercicio del capítulo 3.

De esta forma, si comprobamos que el contenido solicitado corresponde a la primera página llamamos a la función carganoticia pasándole como parámetro un id para el número de noticia y para identificar el nodo del DOM y en esta función nos encargamos de leer de la base de datos y publicar el contenido correspondiente. Si por el contrario el contenido solicitado pertenece a una página distinta a la primera el contenido se encontrará almacenado en la variable global, por ello llamamos a la función carga_datos pasándole como parámetro el tipo post_noticias o post_evento y en dicha función nos encargamos de publicar el contenido en el DOM. En la figura 50 vemos el código del manejador de un botón leer más de noticias, para los eventos es similar.

```
<article id="noticia0" data-dojo-type="dojox.mobile.RoundRect" shadow="true"> </article>
<button id="bn0" class="leerMas" data-dojo-type="dojox.mobile.Button" style="visibility:hidden" data-dojo-props='label:">"
    onClick:function(e){
    dojo.byId("bn0").style.visibility = "hidden";
        if(conexion==1&&pag_noticias!=1){
            carga_datos(0,"post_noticias",0);
        }else{
            carganoticia(0);
        }
    return true;
}'></button>
```

Figura 50. Código del manejador del botón leer más de una noticia.

Para el caso de los post del blog el manejador es prácticamente idéntico, pero en el caso de pedir un post de una página distinta a la primera llamaremos a la función petición, ya que habrá que pedir a su url el contenido completo.

Como comentamos anteriormente, los casos de éxito y las soluciones son tratados de manera un poco diferente. Vamos a explicar cómo realizamos las peticiones para este caso.

Al inicio de este punto 4.4.2 explicamos que habíamos optado en nuestro diseño por solicitar únicamente el contenido de un caso de éxito o de una solución si el usuario accedía directamente a ese menú. Para controlar esto, volvemos a modificar el código de la clase View.js de Dojo con la cual ya nos familiarizamos en el punto 3.3.3 cuando comentamos los problemas del navegador nativo de Samsung. En concreto,

modificamos el método onAfterTransitionIn que es llamado cuando se accede a una vista nueva y mediante el parámetro moveTo identificamos a qué vista se encuentra accediendo. Así, si estamos accediendo a una vista correspondiente a un caso de éxito o a una solución realizaremos la petición correspondiente.

Respecto a las peticiones, los diferentes casos de éxito y soluciones tienen una categoría propia dentro de la web de movilforum [37] que es identificada por un id numérico concreto, luego debemos realizar una petición por categorías y mediante su id identificamos la categoría que queremos. La petición en concreto tiene el siguiente formato:

http://espana.movilforum.com/api/get_category_posts/?id=313

Además, tenemos una variable global denominada pedido que es un array mediante el cual identificamos si hemos solicitado ya ese caso de éxito o no para evitar pedirlo cada vez que el usuario acceda a ese menú. En la figura 51 puede verse un extracto del código encargado de hacer esto.

```
switch(moveTo){
    case "ventas":
        if(pedido[0]==0){
            //si no hemos pedido pedimos
            peticion("ventas",329,0);
            pedido[0]=1;
        }
        //329 es el id de fuerza de ventas
         break;
```

Figura 51. Pedido fuerza de ventas.

En la función petición implementamos el formato de url comentado anteriormente y mediante el id que le pasamos como parámetro formalizamos la petición. Luego, en la función carga_datos, identificamos el elemento del DOM mediante el tipo y publicamos el contenido donde corresponde. Además, una vez que el usuario haya solicitado una solución o un caso de éxito concreto, almacenamos su contenido en la base de datos para mostrarlo también cuando el usuario acceda de forma offline. Esto lo hacemos mediante la función metesolucion a la que le pasamos dos parámetros, el tipo y un segundo parámetro que indica si debemos meter uno o dos post, en función de lo que nos haya sido devuelto. El código de esta función es el de la figura 52.

```
function metesolucion(tipo,numero){
    //esta funcion es la encargada de meter en la base de datos las soluciones y los casos de exito,
    //con numero controlamos si debemos almacenar una o dos
    persistence.store.websql.config(persistence, 'ptest', 'A database description', 5 * 1024 * 1024);
    persistence.schemaSync(function(){
        for(i=0;i<=numero;i++){
            var solucion = new Soluciones();
                //importante el name es el identificador del id del DOM para meter el contenido
                //sera ventas0 ventas1 ocio0 ocio1
                solucion.name = tipo+i;
                solucion.contenido = sol.posts[i].content;
                solucion.titulo = sol.posts[i].title;
                //con add los vamos metiendo, pero hasta que no hacemos flush no pasan a la base de datos
                persistence.add(solucion);
        }
        persistence.flush(function(){
            //mediante flush los volcamos a la base
        });
    });

}
```

Figura 52. Código función metesolucion.

Cabe reseñar que nos vemos lastrados por no poder crear objetos del DOM de forma dinámica tal y como comentamos en el punto 3.3.2 en el que tratamos las limitaciones de tener que emplear sólo objetos de tipo Dojo Mobile.

Decimos que estamos lastrados debido a que, dependiendo de la categoría para soluciones o casos de éxito, tendremos entre 1 y 4 posts y lo ideal sería poder crear los nodos del DOM en los que publicar el contenido de forma dinámica ya que no sabemos a priori cuantas soluciones o cuantos casos nos serán devueltos. Para solventar esta limitación hemos optado por reservar espacio para dos elementos en todas las vistas y mostrar únicamente las dos soluciones o los dos casos de éxito más recientes. Asimismo, por mostrar únicamente dos post y no cinco como en las noticias o el blog, mostramos directamente todo el contenido y no sólo el excerpt.

4.4.3 Doble funcionalidad de la página (online – offline)

A lo largo de éste capítulo 4 hemos explicado cómo llegamos a almacenar los contenidos para que se encuentren disponibles cuando el usuario acceda a la página sin tener conexión disponible. Vamos a analizar en este punto como implementamos la alternancia entre el funcionamiento offline y online sin que se resienta la funcionalidad de nuestra aplicación.

El primer punto que debemos comentar aquí es como logramos el *graceful degradation* en nuestra página. Deberemos ser capaces de ir detectando las características que presenta el dispositivo de un usuario en concreto y partir de menos

funcionalidad a mayor funcionalidad. Para ello lo primero que detectamos el si el usuario tiene activado Javascript, esto lo sabemos mediante la librería de Dojo al emplear el método parse, si no se consigue parsear el código no tendrá funcionalidad ninguna así pues sabemos que el usuario no soporta Javascript y se lo indicamos mediante un mensaje.

No obstante, en la hoja de estilo que creamos en el capítulo 3 y que comentamos en el punto 3.1.2, todo diseño estaba basado en elementos de tipo Dojo Mobile, así pues, cuando el método parse asignaba a cada *tag* de HTML el objeto Dojo Mobile correspondiente la página adquiría el aspecto deseado, pero un usuario que no tuviera activado Javascript vería un diseño HTML plano.

Respondiendo a la necesidad de *graceful degradation* debemos mostrarle al usuario el mismo diseño de nuestra página aunque no tenga activado Javascript, es por ello que hemos realizado en este capítulo modificaciones sobre el css para dotar del mismo aspecto a los elementos HTML que sus homólogos del capítulo anterior.

Hemos añadido un id a cada elemento de la lista de la vista principal y hemos creado una regla específica para cada uno para dotarle de su icono sin tener que utilizar las propiedades de Dojo. En la figura 53 mostramos el código de la vista principal. Hay que destacar que si el usuario no tiene habilitado Javascript, esta vista es la única que verá, ocultamos el resto de vistas y las mostraremos posteriormente mediante Javascript si el usuario lo tiene habilitado.

Además hemos añadido el pie de página en el que facilitamos la posibilidad al usuario de acceder vía web a la sección de contacto para ponerse en contacto con el administrador de la web de movilforum, así como información relativa al copyright de la página de movilforum [37].

```
<!-- vista principal -->
    <section id="principal" data-dojo-type="dojox.mobile.View" data-dojo-props="selected: true">
        <img id="home" class="imagenCabecera" data-dojo-type="dojox.mobile.ListItem" src="./pictures/logo_movilforum.png" style="width:100%; height:auto"; />

        <ul class="lista_princ" data-dojo-type="dojox.mobile.RoundRectList" >
            <li id="cargando"  class="lista" data-dojo-type="dojox.mobile.ListItem" style="visibility:hidden" data-dojo-props="icon:'./pictures/cargando.gif'">
                Actualizando contenido...
            </li>
            <li id="princ_soluciones" class="lista" data-dojo-type="dojox.mobile.ListItem" data-dojo-props="moveTo: 'soluciones'">
                Soluciones
            </li>
            <li id="princ_casos" class="lista" data-dojo-type="dojox.mobile.ListItem" data-dojo-props="moveTo: 'casos'">
                Casos de éxito
            </li>
            <li id="princ_eventos" class="lista" data-dojo-type="dojox.mobile.ListItem" data-dojo-props="moveTo: 'eventos'">
                Eventos
            </li>
            <li id="princ_noticias" class="lista" data-dojo-type="dojox.mobile.ListItem" data-dojo-props="moveTo: 'noticias'">
                Noticias
            </li>
            <li id="princ_blog" class="lista" data-dojo-type="dojox.mobile.ListItem" data-dojo-props="moveTo: 'blog'">
                Blog
            </li>

        </ul>
        <div id="destacado" data-dojo-type="dojox.mobile.RoundRect" shadow="true"> Esta página requiere de Javascript para funcionar, por favor habilitelo </div>
        <footer  class="abajo" data-dojo-type="dojox.mobile.RoundRect">
            <p class="contacto" style="margin-top:5px;"  >
                <img src="./pictures/icon_contact.png" style="padding:0px 5px 0 0;"> </img>
                <a  target="_blank" style="color:#64C3D6;text-decoration:none; margin-top:-10px; font-size: 10px; line-height:10px;"
                href="http://espana.movilforum.com/about-us/contacto/">admin@movilforum.com  </a>
            </p>
            <p  style="margin-top:-10px; color:#c0c0c0; font-size: 10px; line-height:10px;">
            <a  target="_blank" href="http://espana.movilforum.com/texto-legal/" style="color:#fff; text-decoration:none; font-size:10px; float:left;margin-right:5px;">Texto Legal</a>
            | © 2011-2012 Telefónica S.A. Todos los derechos reservados </p>
        </footer>
    </section>
```

Figura 53. Código HTML de la vista principal.

En la figura 54 puede verse el código css que le dota a la página del mismo estilo que si se cargara Dojo.

```
body {
  background-color: #FFFFFF;
    width: 100%;
  margin: 0px;
  padding: 0px;
}
ul{
    margin: 10px;
    padding: 0px;
    list-style-type: disc;
}
.lista {
  position: relative;
  margin: 0px;
  padding: 0px;
  padding-left: 35px;
  padding-top: 5px;
  background-color: #FFFFFF;
  font-family: Helvetica;
  color: #0d4f5b;
  font-size: 16px;
  font-weight: bold;
  position: relative;
  list-style-type: none;
  vertical-align: bottom;
  height: 43px;
  border-bottom: 1px solid silver;
  font-weight: bold;
  color: #004250;
  line-height: 43px;
}
#princ_eventos{
    background: url(./pictures/eventos.png) center left no-repeat;

}
```

Figura 54. Código CSS.

Como vemos en la figura 54, hemos dotado de estilo propio al elemento body, haciéndolo de color blanco, luego mediante el elemento ul le hemos dotado de un aspecto general a la lista y la clase más importante es la clase lista en la que le

indicamos el color de las líneas, la fuente a utilizar, el color de la misma, el tamaño, etc, todo ello para que tenga el mismo aspecto que si se encuentra activado Javascript. Por último también vemos en la captura la regla para el identificador princ_eventos para cargar el icono sin necesidad de utilizar la propiedad de Dojo, esta regla es similar para el resto de elementos de la lista.

Mostramos a continuación en la figura 55 el aspecto que tendría la página para un usuario que se encuentra accediendo a la misma con Javascript desactivado.

Figura 55. Aspecto de la página sin Javascript. Captura desde móvil Android.

El siguiente punto a tener en cuenta en *graceful degradation* es la funcionalidad de la que vamos a dotar a nuestra página si el usuario se encuentra sin conexión. Tal y como comentamos anteriormente, por razones de diseño y de almacenamiento hemos optado por almacenar en la base de datos la primera página de las noticias y los post, los eventos y únicamente, aquellas soluciones o casos de éxito que el usuario solicitara cuando accedió de forma online a la aplicación, así pues, cuando se acceda sin conexión a la página, estos contenidos y no otros serán los que podamos mostrar.

El primer paso que realizamos es detectar si tenemos o no tenemos conexión, tal y como vimos en el código mostrado en la figura 43, una vez que sabemos que nos encontramos sin conexión ocultamos los botones para pedir las siguientes noticias y los siguientes post del blog, ya que al encontrarse el usuario sin conexión si intenta pedir las siguientes noticias no le serán devueltas y la presencia de ese botón puede llevarle a confusiones. Una vez ocultados ambos botones llamamos a la función rellenaDOM que, como pudo verse en la figura 44, extrae los contenidos almacenados en la base de datos y los muestra en el DOM, tras publicar todos los contenidos, también mediante la propia función rellenaDOM hacemos visibles los botones de leerMas para las noticias, los post del blog y los eventos.

Lo último que nos queda por explicar del funcionamiento offline es como hacemos para leer el contenido completo de una noticia, de un post, o de un evento. Como vimos en el manejador del botón de leer más de una noticia mostrado en la figura 50, si nos encontramos sin conexión llamamos a la función carganoticia, pasándole como parámetro un id (tenemos métodos análogos para los post del blog y para los eventos), en esta función carganoticia extraemos las noticias de la base de datos, mediante el identificador pasado como parámetro identificamos con el parámetro name la noticia que estamos solicitando y mostramos su contenido completo en el nodo correspondiente.

Respecto al funcionamiento online, como explicamos en el punto 4.4.2 primero debemos descargar el contenido y posteriormente proceder a almacenarlo en la base de datos. Una vez hemos completado ese proceso el usuario puede interactuar sin problemas con la aplicación. En primer lugar, mediante Javascript mostramos todos los elementos que habíamos ocultado inicialmente para que no fueran mostrados en el caso de que el navegador no tuviera habilitado Javascript. Tras esto, procedemos a identificar si tenemos o no tenemos conexión y una vez detectamos que sí tenemos, solicitamos el contenido al servidor.

Una vez que hemos almacenado el contenido completo del último post en la base de datos ya estamos preparados para que el usuario interactúe, para ello le indicamos mediante un mensaje que el contenido ha sido actualizado y la fecha de la actualización y mostramos los botones de leerMas que permanecían ocultos hasta ahora.

En este punto hemos explicado la alternancia entre funcionamiento offline y funcionamiento online, pero como vamos a comentar a continuación, la página ha presentado un problema concreto a la hora de detectar la conexión, lo cual nos ha obligado a introducir algunas ligeras modificaciones que comentamos en el punto siguiente.

4.5 Dificultades y soluciones adoptadas.

En el punto 4.4 hemos explicado cómo se comporta la página cuando se detecta conexión y cómo lo hace cuando detectamos que no hay conexión, también hemos explicado cómo logramos ese comportamiento. En este punto 4.5 vamos a explicar un problema que hemos observado que sucede en todos los dispositivos respecto a la detección de conexión y problemas más específicos que nos hemos encontrado al trabajar con los dispositivos iPhone 4S que nos han forzado a reducir la funcionalidad para trabajar sólo de forma online para dicho dispositivo.

4.5.1 Fallos al detectar conexión.

El problema con la detección de conexión es el siguiente. Una vez se ha almacenado el contenido en la memoria del dispositivo, si cerramos el navegador y matamos el proceso, cuando nos desconectemos y volvamos a abrir el navegador para introducir la dirección de la página, el navegador en lugar de detectar que nos encontramos sin conexión, detecta que sí tiene conexión y tratará de ejecutar los contenidos de forma online como así lo dicta el código. Sin embargo, si una vez nos encontramos en la página web pulsamos el botón de actualizar la misma estando sin conexión, el navegador sí detecta que no tenemos conexión y ejecuta el código correspondiente al comportamiento offline.

Este error en la detección de la conexión ha sido imposible de subsanar, parece ser intrínseco al funcionamiento de los navegadores. Pero como sabemos, nuestra aplicación debe funcionar continuamente de manera tanto offline como de manera online, es por ello que nos hemos visto obligados a realizar modificaciones en nuestro código para tratar con este problema.

El principal cambio que hemos hecho es llamar a la función rellenaDOM tanto si detectamos que hay conexión como si no, a pesar de que la idea inicial era hacerlo

únicamente cuando no se tuviera conexión. Así, si tenemos contenidos almacenados en la base de datos los mostraremos en el DOM. Dicho contenido podrá ser sobrescrito si la petición devuelve contenidos más actualizados, pero si no, permanecerán inalterados. Así pues, el llamar a la función rellenaDOM también nos proporciona una mayor rapidez a la hora de acceder a los contenidos sin tener que esperar a que sean descargados de nuevo. **No obstante, para cerciorarnos de que está todo el contenido disponible deberemos esperar a que la aplicación nos indique el mensaje "Actualizado" más la fecha.**

Otro aspecto intrínseco del funcionamiento sin conexión es un mensaje que nos mostrará nuestro navegador cuando tratemos de acceder a la página web encontrándonos sin conexión. Ya sea accediendo desde un Android o desde un dispositivo iOS, el navegador nos mostrará un mensaje similar al que puede observarse en la captura de la figura 55. Es un mensaje que no podemos evitar que nos lance el navegador. No obstante, debemos darle al botón de aceptar y podremos trabajar con la aplicación sin problemas.

Figura 56. Mensaje de alerta sin conexión. Captura desde móvil Android.

Tras estas modificaciones conseguimos que nuestra aplicación se encuentre perfectamente operativa en dispositivos Android e iOS así como en navegadores de sobremesa de forma online como de forma offline. Por último para finalizar con este capítulo 4 que trataba sobre la persistencia vamos a explicar en el último punto del capítulo los problemas que nos hemos encontrado con los dispositivos iPhone 4S.

4.5.2 Base datos iPhone 4S.

El problema que hemos encontrado para los dispositivos de este tipo es que no permiten acceder a la memoria del teléfono mediante *Web SQL Database* tal y como hemos implementado en nuestro diseño. Tras contactar con los responsables del *framework* persistence.js nos indicaron que debíamos cerciorarnos de no rebasar el límite de la memoria del iPhone que son 5Mb, pero como vimos en el punto 4.3.1.1, en la figura 41, cuando definimos la base de datos le indicamos un tamaño de 5Mb, con lo cual no debería ocasionar conflicto, a pesar de ello, se redujo el espacio reservado a 3Mb pero persistieron los problemas con el iPhone 4S. Además la aplicación ha sido testeada en otros dispositivos iOS tales como iPad (1 y 2) e iPhone (3 y 4) y la base de datos ha funcionado correctamente sin ningún tipo de error.

Así pues, una vez más, dispuestos a seguir los principios de *graceful degradation*, nos vemos obligados a modificar el código de la aplicación para lograr dotar de completa funcionalidad online a nuestra aplicación sin necesidad de emplear la base de datos para nada, pero al mismo tiempo empleándola en el caso de que sí sea posible. Vamos a ver cómo logramos esto.

El elemento clave que nos ayuda en este caso son los *timeout*. Javascript nos proporciona una pareja de funciones denominadas setTimeout y clearTimeout que nos permitirán ejecutar una función o cancelar la ejecución de la misma en un tiempo determinado. Mediante la función setTimeout podemos pasarle dos parámetros, el primero una llamada a una función y el segundo el tiempo en minisegundos tras el cual, una vez transcurrida esa cifra llamará a la función que le hemos pasado como parámetro. A su vez, mediante la función clearTimeout podemos cancelar ese tiempo y así evitar que se llame a la función. Para trabajar con estas funciones nos creamos la variable global t, que será la encargada de controlar el tiempo.

Para el correcto funcionamiento de nuestra aplicación en el iPhone 4S, o en otros dispositivos incapaces de trabajar con la base de datos, nos hemos visto obligados a realizar modificaciones en la función carga_datos, así como en las funciones que son llamadas por los manejadores de los botones leerMas. En primer lugar, en la función carga_datos, una vez que hemos recibido la respuesta del servidor por todas las peticiones de las noticias, los post del blog y los eventos, cuando nos encontramos solicitando el contenido completo para el primer post, lanzamos un *timeout* que llamará transcurridos 15 segundos a la función muestraBotones. Esta función simplemente se encarga de hacer visibles los botones de leerMas y de indicar mediante un mensaje que se produjo un error en la base de datos ya que únicamente será llamada en el caso de que este *timeout* venza.

Estos quince segundos son el tiempo que damos para que nuestra aplicación solicite el contenido completo de los 5 primeros post y almacene su contenido en la base de datos. Si por algún motivo la aplicación es incapaz de acceder a la base de datos, una vez transcurridos esos 15 segundos se indica un mensaje de error al intentar acceder a la base de datos, pero se le deja actuar al usuario con la aplicación mostrándole los botones correspondientes, además si no podemos almacenar el contenido en la base de datos, únicamente pediremos el primer post y no solicitaremos los siguientes. Sin embargo, si todo transcurre según lo previsto y la aplicación puede actuar con la base de datos, una vez que hemos almacenado el contenido completo en la base, mediante la función clearTimeout cancelamos el *timeout* y mostramos los botones y un mensaje indicando la fecha de la última actualización, momento a partir del cual, como comentamos anteriormente, el usuario puede interactuar plenamente con la aplicación.

Nos queda explicar, como conseguimos que los manejadores de los botones de leerMas consigan mostrar al usuario el contenido completo de una noticia sin utilizar la base de datos si, como vimos en el punto 4.4.2, el manejador llamaba a una función que se encargaba de extraer el contenido directamente de la base de datos.

Para lograr esto hemos tenido que modificar las funciones carganoticia, cargaevento y además crearnos una auxiliar para cargar los post. Vamos a explicar primero la modificación llevada a cabo sobre carganoticia, que es similar a la de cargaevento.

En carganoticia lo que hacíamos era extraer el contenido de la base de datos y mostrarlo, ahora seguimos haciendo esto pero, para el caso de que se produzca algún error en la base de datos, mostramos el contenido también que se encuentra almacenado en la variable global. Cabe destacar que, en el caso en que accedamos a la web sin conexión, esta variable global se encontrará vacía y lo que se mostrará será el contenido proveniente de la base de datos, luego esta modificación no nos introduce ningún fallo en el comportamiento offline. En la figura 56 mostramos el código de la función carganoticia.

```
function carganoticia(id){
    //llamaremos a esta funcion para leer el contenido entero de la noticia, ya que accede directamente a la base de datos
    //mediante id identificamos que noticia desea leer el usuario
    var contenido="";
    var cadena="";
    Noticias.all().list(function(noticias){
        var contadorNoticias = noticias.length;
        noticias.forEach(function(noticia){
            if(noticia.name=="Noticia "+id){
                nodo = dojo.byId("noticia"+id);
                contenido = contenido +  "<b style='font-family:arial;color:#555555;font-size:20px;line-height:1.3em'> " + noticia.titulo  + " </b>" + "<br>";
                contenido = contenido + noticia.contenido ;
                nodo.innerHTML = contenido;
                contenido="";
            }
            if(--contadorNoticias == 0){
                //Una vez hemos modificado todos los volvemos a subir a la base de datos
                persistence.flush(function(){
                });
            }

        });

    });
    //si no funciona la base de datos cargamos desde la variable global
    nodo = dojo.byId("noticia"+id);
    contenido = contenido +  "<b style='font-family:arial;color:#555555;font-size:20px;line-height:1.3em'> " + obj.posts[id].title  + " </b>" + "<br>";
    contenido = contenido + obj.posts[id].content ;
    nodo.innerHTML = contenido;
    contenido="";

}
```

Figura 57. Código de la función carganoticia.

Vamos a explicar ahora cómo realizamos esto mismo para los post del blog, tenemos que realizar alguna modificación más porque, como indicamos anteriormente, si se produce algún error con la base de datos, únicamente pediremos el primer post y no tendremos en la variable global el contenido de ningún otro así pues deberemos solicitarlo a la url del post en cuestión.

Lo que hacemos en la función cargapost es lanzar un *timeout* mediante el cual, si en 500ms no se ha podido acceder a la base de datos, llamamos a una función de nuestra creación denominada auxicargapost que se encarga de lanzar la petición para el contenido de un post concreto. Si por el contrario, todo va según lo previsto y podemos acceder a la base de datos, el contenido del post ya se encontrará almacenado, luego lo

mostramos en el nodo del DOM correspondiente y paramos el *timeout* sobre la ejecución de la función auxicargapost para evitar pedirlo el contenido al servidor de nuevo. En la figura 57 mostramos el código de la función cargapost.

```
function cargapost(id){
    //llamaremos a esta funcion cuando queramos leer el contenido entero de un post, cargaremos su contenido desde la base de datos
    //mediante el id identificamos que post quiere leer el usuario
    var contenido="";
    var cadena="";
    var fecha="";
    t = 0;
    id_global = id;
    //ponemos un timeout, si no se accede correctamente a la base, pedimos el post online
    //solo se producira este caso en el supuesto de algun error en la conexion con la base de datos
    persistence.store.websql.config(persistence, 'ptest', 'A database description', 5 * 1024 * 1024);
    t = setTimeout('auxicargapost()',500);
    Blog.all().list(function(bls){
        var contador = bls.length;

        bls.forEach(function(bl){
            if(bl.name=="Blog "+id){
                nodo = dojo.byId("post"+id);
                contenido = contenido +  "<b style='font-family:arial;color:#555555;font-size:20px;line-height:1.3em'> " + bl.titulo  + " </b>" + "<br>";
                fecha = bl.fecha;
                fecha = fecha.substring(0,fecha.length-3);
                contenido = contenido + "<b style='font-family:arial;color:silver;font-size:12px;line-height:1.3em'>" + "Publicado el " +  fecha + " por " + bl.autor + "</b>" +"<br>";
                contenido = contenido + bl.contenido ;
                nodo.innerHTML = contenido;
                contenido="";
            }
            if(--contador == 0){
            //Una vez hemos modificado todos los volvemos a subir a la base de datos
                persistence.flush(function(){
                });
            }

        });
        clearTimeout(t);
    });
}
```

Figura 58. Código de la función cargapost.

Como puede apreciarse del código, tenemos una variable de carácter global denominada id_global, esta variable es la que usamos en la función auxicargapost para saber cuál es el post que estamos solicitando ya que, debido a una limitación de la función setTimeout, la función a la que llamemos no puede tener parámetros y no podemos indicarle de ninguna manera a auxicargapost que post queremos pedir.

Al finalizar este capítulo 4 ya tenemos nuestra aplicación completamente operativa y con comportamiento offline. Hemos adquirido conocimientos suficientes y nos hemos enfrentado a problemas intrínsecos de trabajar con el compendio de tecnologías que conforman HTML5. Podemos considerar pues, que tenemos unas bases para poder analizar las fortalezas y debilidades que presenta el trabajar con HTML5 frente a hacerlo de forma nativa para las diferentes plataformas. Es esto lo que vamos a comentar en el capítulo 5.

CAPITULO 5. Comparativa HTML5 vs Aplicación Nativa

Tras la finalización de nuestra aplicación y una vez se encuentra totalmente operativa, podemos afirmar que hemos adquirido conocimientos de las bases sobre las que se establece la tecnología HTML5. En este capítulo vamos a proceder a comentar la situación actual de HTML5 para el diseño de aplicaciones de escritorio y de móvil comentando las debilidades y las fortalezas intrínsecas, tanto de la propia tecnología, como de su soporte y su rendimiento en los diferentes dispositivos. Una vez conozcamos sus puntos débiles y fuertes, comentaremos si la situación actual es prometedora o por el contrario no lo es y las líneas futuras de trabajo que debe seguir en nuestra opinión esta tecnología.

5.1 Debilidades de HTML5.

Vamos a empezar reseñando los puntos débiles que presenta HTML5 frente al desarrollo de aplicaciones de forma nativa. Explicaremos primero las deficiencias técnicas de este tipo de diseño en comparación a realizar las aplicaciones de forma nativa y continuaremos con sus debilidades de carácter comercial y de cara al usuario.

Como vimos en el capítulo 4, la persistencia es muy importante en las aplicaciones móviles, así pues también lo es en nuestro diseño. Debido a esto, HTML5 presenta una desventaja frente a las aplicaciones nativas y es que, al estar basada la persistencia en los diferentes navegadores, el usuario en cualquier momento puede borrar el contenido que hemos almacenado en el navegador mediante el borrado de la caché, mientras que en las aplicaciones nativas el usuario no tiene acceso a priori a cómo tratan las aplicaciones los datos y, por lo tanto, no podrá borrarlos de forma tan sencilla. Además de esto, en un diseño de forma nativa la memoria de la que dispone una aplicación para almacenar su contenido es mucho mayor de la que disponemos mediante el almacenamiento directamente en el navegador, que en los dispositivos iOS es de 5Mb y en los dispositivos Android es similar.

Uno de los mayores puntos débiles de HTML5 frente a las aplicaciones nativas es la diferencia de rendimiento que presenta en dispositivos de sobremesa en comparación

con el que ofrece en los dispositivos portátiles. Tal y como se nos comenta en [52] el rendimiento en los móviles respecto a los ordenadores se reduce en una cantidad que oscila entre las seis y las mil veces, sobre todo para el tratamiento de uno de los puntos fuertes con los que cuenta HTML5, el tratamiento de imágenes mediante canvas. Para entender mejor este problema vamos a analizar que SO de los dispositivos portátiles nos proporciona un mayor soporte para HTML5 y tras esto, lo compararemos con un dispositivo de sobremesa que use el mismo SO.

En una comparativa extraída de [53] podemos ver la puntuación obtenida por los navegadores de cada sistema operativo (Android 2.3, Android 4.0 e iOS 5.1) en función de las funcionalidades de HTML5 que soporta cada uno. Como puede verse en la figura 58 el que obtiene mayor puntuación es iOS 5.1.

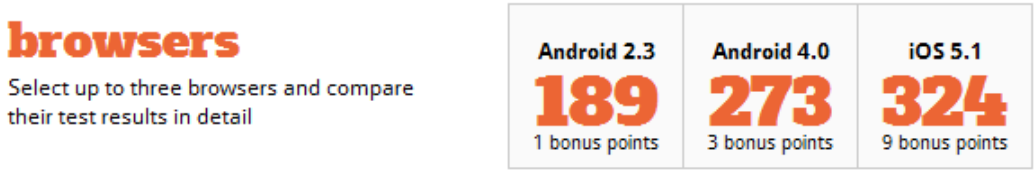

Figura 59. Comparativa por funcionalidades de diferentes navegadores. Fuente [36]

Esta comparativa es importante ya que una característica de HTML5 que funcione en un dispositivo puede no hacerlo en otro y en esta comparativa se otorga una puntuación acorde al número de características que soporta cada SO.

Existen diversos *frameworks* destinados a lidiar con este problema. Uno de ellos es Modernizr [54], este *framework* nos permite comprobar previamente si una funcionalidad de HTML5 es soportada por el navegador que está visionando la página o por el contrario no lo está, permitiendo así ejecutar un código u otro, siendo así una herramienta muy útil para cumplir con el *graceful degradation*.

No obstante, en el desarrollo de este proyecto no hemos utilizado esta librería puesto que todas las funcionalidades implementadas en el mismo han sido previamente testeadas y se ha comprobado que los navegadores dan soporte a las mismas, incluso la persistencia empleando *Web SQL Database.* Luego, el empleo de la librería Modernizr, no nos solucionaría el problema de la base de datos en iPhone 4S ya que la misma detectaría que el dispositivo sí proporciona soporte para dicha funcionalidad.

Mediante la comparativa de la figura 58 hemos visto que iOS 5.1 es el SO que proporciona más soporte para HTML5, para hacernos una idea de la diferencia de rendimiento de un dispositivo que emplee este SO con un *laptop* recurrimos a la comparativa que se ha llevado a cabo en [52]. En esta comparativa se especifica la diferencia de rendimiento que presentan los diferentes dispositivos al ejecutar alguna de las funcionalidades de HTML5 para el tratado de imágenes.

Como puede verse en la figura 59, los resultados obtenidos en esta comparativa dejan claro que la diferencia de rendimiento en cuanto al tema gráfico se refiere es abismal.

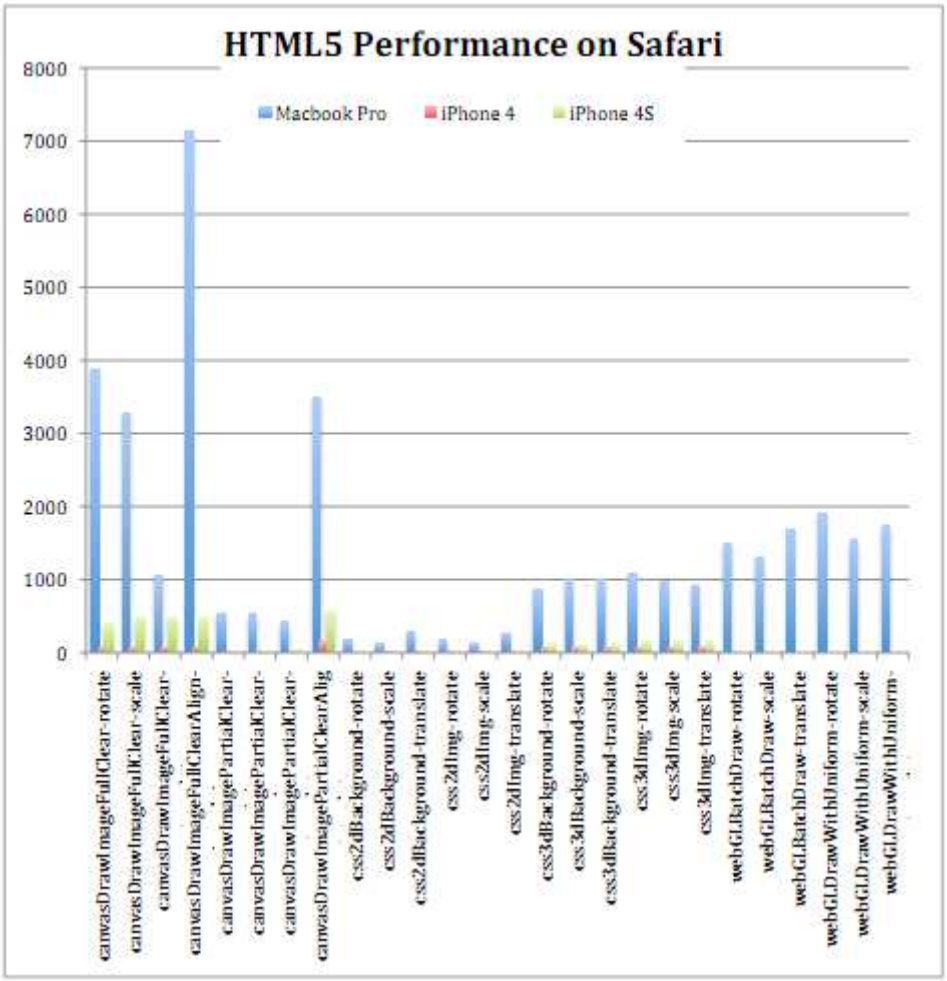

Figura 60. Comparativa rendimiento dispositivos iOS. Fuente [35]

De esta comparativa nos queda clara la idea de que aquellas aplicaciones que requieran un tratamiento exhaustivo de imágenes tales como juegos se encuentran en clara desventaja frente a las aplicaciones nativas que no presentan este tipo de retrasos. Podemos ver por nosotros mismos la diferencia de rendimiento que presentan este tipo de aplicaciones si accedemos a la dirección de [55]. En ella podemos encontrar el juego

Super Mario Bros implementado mediante HTML5, si accedemos desde un navegador Chrome en nuestro *laptop* el rendimiento se encuentra optimizado y es plenamente satisfactorio, mientras que si accedemos desde un dispositivo móvil el juego se visiona ralentizado y no del todo correctamente, haciendo que sea impracticable su uso.

Otra gran debilidad de HTML5 es, como se nos indica en [56], que las herramientas que nos proporcionan los navegadores para ayudarnos a *debuggear* nuestro código se traducen también en una fuga de seguridad, proporcionando a algún *hacker* malintencionado la posibilidad de modificar el valor de alguna de nuestras variables. Este es un punto crítico que se debe solucionar, aunque la criticidad varía de unas aplicaciones a otras, el mero hecho de que exista una falla tan sensible de seguridad es un punto a tener muy en cuenta.

Además, debido al empleo de HTML5 la experiencia de usuario es, en ocasiones y dependiendo del dispositivo, bastante diferente de una aplicación diseñada de forma nativa. En este punto hay que realizar distinciones entre Android e iOS por estar cada usuario acostumbrado a una experiencia de usuario diferente.

En primer lugar, una aplicación diseñada nativamente en ambas plataformas (Android e iOS) es accesible mediante un icono que se encontrará disponible en el menú de aplicaciones de los mismos. En nuestro caso nos encontramos diseñando una página web, luego necesitamos crear un icono de acceso directo a nuestra página. Ambas plataformas nos ofrecen esta posibilidad, pero es bastante más sencilla y habitual esta práctica en dispositivos iOS que en aquellos cuyo sistema operativo es Android.

En dispositivos iOS la creación del icono es prácticamente inmediata, sin más que pulsar sobre el icono situado en la barra de navegación del navegador Safari e indicarle crear acceso directo. Tras esto ya tenemos disponible el icono para nuestra página web entre los iconos de las aplicaciones y sin más que pulsar sobre él se nos abrirá nuestra página.

Por el contrario, en los dispositivos Android el proceso es más complejo y requiere de más pasos. En primer lugar hay que agregar nuestra página a favoritos del navegador, tras lo que debemos situarnos en el escritorio de nuestro dispositivo y seleccionar crear acceso directo. Entre los accesos directos buscamos el icono de favoritos y tras esto, seleccionamos nuestra página. Una vez completado este proceso, tendremos disponible

el icono para acceder a nuestra página web de forma directa en el escritorio de nuestro dispositivo. No obstante, como hemos indicando anteriormente, esta posibilidad de crear un acceso directo a los favoritos de su navegador es a menudo desconocida por los usuarios de Android, luego nos encontramos aquí con un problema de costumbres establecidas que representa un hándicap para este tipo de diseño.

Otro punto importante a reseñar en torno a la experiencia de usuario es que los dispositivos Android a menudo poseen un botón (ya sea físico o táctil) de retroceso para navegar entre los diferentes menús. Como pudimos ver en la figura 11, nuestra aplicación basa la navegación entre las diferentes vistas mediante un icono situado sobre la parte superior de la vista en cuestión y es mediante la pulsación de este botón que se accede a la vista anterior ya que no nos encontramos navegando entre distintos links como pueda suceder con una típica página web, si no que nos encontramos todo el tiempo en una misma página y navegamos entre sus contenidos mediante Javascript. Es por este motivo que los usuarios que usen Android tiendan, hasta que se acostumbren a la aplicación, a volver al menú anterior pulsando la tecla de retroceso, lo cual les hará salirse de nuestra página web y volver a la que estuvieran visionando en su navegador anteriormente.

Nuevamente, al igual que el punto anterior, este es un problema de la costumbre establecida entre los usuarios de Android, para los usuarios de iOS sin embargo este problema no sucederá puesto que la forma de navegar entre los menús de nuestra aplicación es muy similar a la que presentan los dispositivos iOS. Sin más que acceder al menú de Ajustes de un iPhone, presentado en la figura 60, podemos ver que la navegación entre los menús es igual que la de nuestra página web con un botón en la parte superior para volver al menú anterior.

Figura 61. Menú ajustes dispositivo iOS.

El último punto débil que presenta HTML5 relacionado con la experiencia de usuario es la imposibilidad de realizar notificaciones en la barra de tareas de su dispositivo. Así pues mediante HTML5 será imposible informar al usuario de que hay nuevo contenido disponible o cualquier otro tipo de notificación hasta que él no decida acceder a nuestra aplicación de forma voluntaria.

Otro aspecto interesante de HTML5 y una de sus mayores virtudes como comentaremos en el punto siguiente es la independencia que nos otorga diseñar nuestra página en HTML5 con respecto a los *markets* o *stores* de los dispositivos. Pero, esta misma ausencia de las aplicaciones basadas en HTML5 en los diversos *markets*, como queda reflejado en [57] se traduce en una mayor invisibilidad para el usuario, lo cual es un problema fundamental a la hora de conseguir que nuestra aplicación llegue al mayor público posible.

El último escoyo que vamos a comentar en este apartado es de otra índole, el de carácter comercial. A las grandes compañías que rigen el mercado de los *smartphones* no les interesa perder el control de las aplicaciones que son empleadas por sus dispositivos y es por eso que plantean mecanismos de control como el *market*. Como vimos en la figura 58 los dispositivos basados en Android e iOS no presentan todas las funcionalidades que nos proporciona HTML5, y está en la mano de estas dos compañías dotar de más soporte a sus dispositivos, luego debemos esperar a ver cómo evolucionan

estos intereses comerciales. No obstante, como veremos posteriormente, las perspectivas son halagüeñas.

5.2 Fortalezas de HTML5.

Una vez comentadas las debilidades que tiene HTML5 frente a las aplicaciones nativas vamos a ver cuáles son aquellas otras fortalezas que le dotan de un atractivo para los diseñadores.

Uno de los principales atractivos de HTML5 es la posibilidad que nos otorga de diseñar una única aplicación compatible con todo tipo de dispositivos, ya posean el SO Android o iOS, ya sean móviles, *laptops* o *tablets*. Esta es la característica de HTML5 más importante puesto que puede traducirse en un ahorro de costes muy significativo. Como hemos podido observar a lo largo del documento, siguiendo los principios de *responsive design* y *graceful degradation*, hemos conseguido que nuestra aplicación sea utilizable por varios tipos de dispositivos empleando un único código para el diseño de la misma. Ha sido necesario realizar algunas pequeñas modificaciones para ir añadiendo las funcionalidades paulatinamente, pero en cualquier caso, resulta más interesante para el diseñador tener que controlar estos aspectos que diseñar una aplicación completamente desde cero para cada tipo de plataforma.

No obstante, el empleo de HTML5 también nos permite optar por diseñar una aplicación específica en función de cada dispositivo. Para ello, habría que identificar el *user agent* del dispositivo que se encuentra accediendo en nuestro servidor y mostrarle un código u otro. En nuestro caso consideramos más potente, reducido en contenidos para el servidor y didáctico, el diseño de una única aplicación siguiendo los principios básicos en desarrollo software en lugar de realizar una página distinta para cada dispositivo.

Además, gracias a CSS3 podemos conseguir dotar de un aspecto u otro a la página como se vio en el capítulo 2. Así, podemos conseguir simplemente creando una hoja de aspecto diferente que nuestra aplicación luzca de forma totalmente distinta en un dispositivo Android que en un iOS para asemejarnos en mayor medida al aspecto de las aplicaciones nativas según el tipo de dispositivo.

Volviendo al aspecto económico, vamos a tratar de entender por qué afirmamos que HTML5 se traduce en un ahorro de costes. Fundamentalmente los costes de diseño en HTML5 frente a los costes de diseño de forma nativa son menores por dos motivos, el número de diseñadores web frente al de diseñadores nativos y el menor tiempo de desarrollo.

En el mercado laboral se encuentran un gran número de desarrolladores con conocimientos de HTML y Javascript frente al escaso número de desarrolladores de aplicaciones nativas que sepan a su vez diseñar para todo tipo de dispositivos. Esto se traduce en una mayor oferta y por consiguiente en una reducción de costes. Además, diseñar una única aplicación que pueda ser utilizada en todo tipo de dispositivos se traduce en una **reducción notable del tiempo de desarrollo** frente a tener que desarrollar códigos distintos en función del dispositivo y a su vez **permite llegar a un mercado mayor reduciendo esfuerzos.**

Otro de los puntos fuertes de HTML5 es, como ya se citó en el apartado de debilidades, la no necesidad de tener que pasar por un *market* para llegar al usuario. Para entender por qué es este aspecto muy atractivo para los diseñadores vamos a explicar cómo funciona el *market* y la *appstore*.

En el *market*, debido al problema de fragmentación, característico de Android, que puede apreciarse en la figura 61, es el propio diseñador el encargado de testear su aplicación en los dispositivos que él crea conveniente ya que testear la aplicación en todos los dispositivos es, prácticamente, imposible. Es por ello que la aplicación no tardará tanto tiempo en estar disponible en el *market* como en la *appstore*.

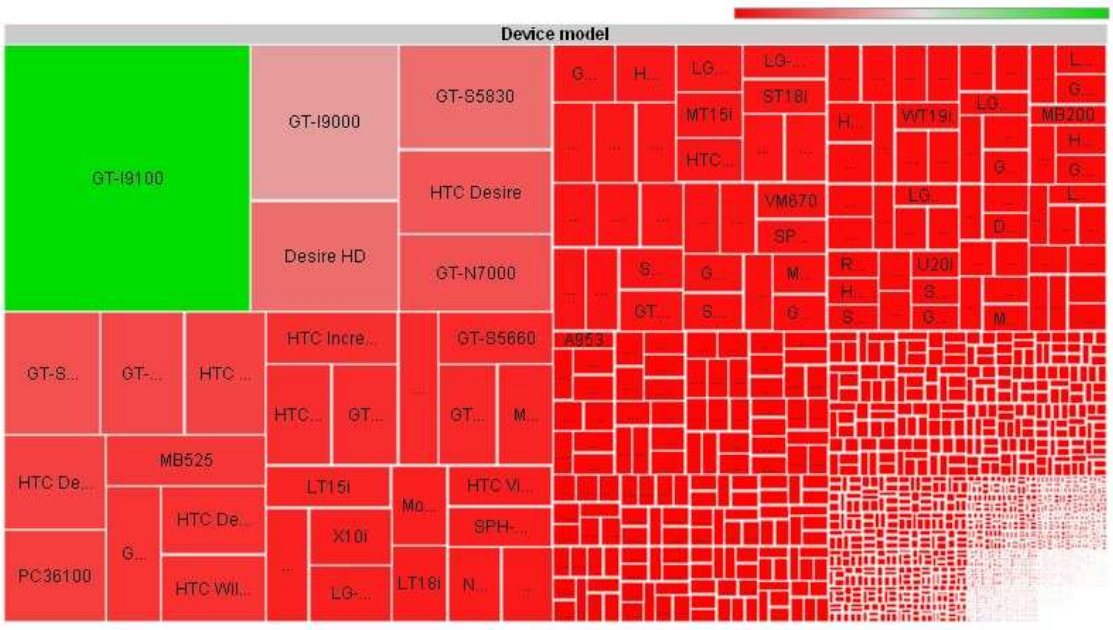

Figura 62. Ilustración del problema de fragmentación en Android. Fuente [58]

En la *appstore* sin embargo, debido a que hay un número mucho menor de dispositivos que en Android, la aplicación sufre un proceso de validación más exhaustivo, siendo testeada en todos los dispositivos iOS. Es por ello que el periodo entre que se solicita la publicación de la aplicación en la *appstore* hasta que ésta se encuentra finalmente disponible es de aproximadamente diez días. No obstante, es muy importante destacar que **Apple es la que tiene la decisión final sobre si publica la aplicación en su tienda o no.** Así pues, si durante el testeo se observa algún fallo en la misma la aplicación, o incluso por otro motivo diferente, la aplicación puede ser rechazada y no se encontrará disponible en la *appstore*.

Del mismo modo sucede para las actualizaciones. Si un desarrollador ha implementado una nueva versión de su aplicación y quiere publicarla en el *market* correspondiente, debe repetir este proceso con el retraso que ello conlleva en el mejor de los casos. Mientras que por el contrario, mediante el empleo de HTML5, es el propio desarrollador el encargado de todo el proceso de publicación de la aplicación. Una vez que la aplicación se encuentre alojada en el servidor, será accesible para todos los usuarios, sin tiempos de espera por intermediarios. **Esto es una enorme ventaja de HTML5 frente a las aplicaciones nativas.**

Además, no debemos olvidar que por cada aplicación descargada del *market* o de la *appstore* la compañía se lleva un porcentaje simplemente por hacer de intermediario.

Luego si se tiene suficiente visibilidad en el mercado para ser conocido sería deseable poder ahorrarse este porcentaje en el coste de distribución de la aplicación y con HTML5 somos capaces de conseguirlo.

Por todo lo comentado en este punto, debemos tener claro que la mayor fortaleza de HTML5 reside en el aspecto económico y en la libertad que otorga al desarrollador para implementar el modelo de negocio que él crea más conveniente en lugar de restringirse al modelo que obligan los *markets*. Un ejemplo de este modelo de negocio puede ser diseñar nuestra página y cobrar por acceder a ella una única vez, emplear one click payment [59] o cualquier otro modelo que se nos pueda ocurrir.

Para concluir con este punto, comentar que algunos periódicos como el financial times [60] o aquí en España la Vanguardia [61] han decidido apostar fuertemente por HTML5 y para dispositivos iOS ya se encuentra disponible su versión en formato HTML5 en lugar de la aplicación nativa.

5.3 Visión de futuro HTML5.

Por todo lo visto en los dos puntos anteriores podemos augurar un gran futuro para HTML5 debido fundamentalmente a las mejoras económicas que nos proporciona.

Debemos tener permanentemente en cuenta que HTML5 es una tecnología muy joven y que se encuentra en proceso de estandarización hasta 2014 [62]. Por lo tanto podemos presumir que buena parte de las debilidades encontradas hasta la fecha en el empleo de esta tecnología irán siendo subsanadas a medida que HTML5 gane cuota de mercado.

También hay que destacar que, como vimos en la figura 58 extraída de la comparativa de [53], la diferencia de soporte en Android 4.0 frente al soporte que ofrecen los dispositivos con versión 2.3 es bastante grande, lo cual nos impulsa a ser optimistas respecto a si las grandes compañías Apple y Google van dar mayor soporte a esta tecnología o si por el contrario van a tratar de disminuir su presencia.

Por su parte, los navegadores no nativos también se encuentran en proceso continuo de actualización y en cada una de ellas se proporciona soporte para nuevas

características de HTML5 no disponibles en la versión anterior, como es el caso de la reciente actualización del navegador Firefox para los dispositivos Android [63].

Respecto al rendimiento, hay que notar que hay ciertos aspectos de la tecnología que se encuentran suficientemente maduros como para implementar ciertas aplicaciones sin ningún tipo de problema pero, sin embargo, hay otros que no hacen posible diseñar para dispositivos móviles aún.

Es por ello que empresas como la Vanguardia o el Financial Times han optado por HTML5 debido a que les aporta unos mayores beneficios que las aplicaciones nativas y además HTML5 les proporciona, ya en la actualidad, el soporte necesario para todos aquellos aspectos que necesitan realizar.

De esto, podemos concluir que, actualmente, el empleo de HTML5 es presumiblemente para aplicaciones ligeras que no requieran de mucha carga computacional o de grandes gráficos ya que pueden ser implementadas con total funcionalidad en HTML5, proporcionando grandes ventajas como la inmediatez en la actualización de versiones de la misma o, incluso, una mayor rapidez de funcionamiento en la misma frente a una aplicación nativa.

Por el contrario podemos afirmar que, en aquellas aplicaciones que requieran de un tratamiento exhaustivo de imágenes y necesiten una tasa mayor a 5fps [64], el empleo de HTML5 está desaconsejado puesto que aún no se encuentra suficientemente soportado a esos niveles por los dispositivos móviles. No obstante, el mercado de dispositivos móviles está en constante renovación surgiendo continuamente dispositivos con procesadores y RAM que crecen exponencialmente respecto al anterior modelo, luego, no es descabellado afirmar que en un año desde ahora pueda haber dispositivos en el mercado con suficiente potencia en su motor Javascript para poder mostrar hasta 30fps [64].

Aún así, si se quiere optar por el empleo de HTML5 para el diseño de aplicaciones que requieran de estas características tales como juegos, hay formas diferentes de lograrlo con el soporte actual. Como ejemplo, la empresa Ideateca [65] consigue desarrollar sus juegos en HTML5 proporcionando las mismas características de un juego desarrollado de forma nativa. Para ello, diseñan un navegador web reduciendo sus funcionalidades al máximo para simplemente mostrar su juego e integran este

navegador en la vista de la aplicación, así la aplicación correrá mediante el navegador que ellos han diseñado específicamente para ese fin traduciéndose esto en una mejora del rendimiento en canvas de hasta 1000% como ellos mismos afirman.

Así pues, las líneas de trabajo futuro en nuestro caso no dependen de la propia tecnología HTML5 si no del soporte que le proporcionen a la misma los diferentes dispositivos. No obstante hemos visto ejemplos de cómo un buen desarrollador web puede diseñar aplicaciones que solventen los problemas surgidos del empleo de HTML5 y que le permitan aprovechar todas las ventajas que esta tecnología ofrece.

Está en la mano del desarrollador tomar ventaja de lo que esta tecnología ofrece y, a medida que pase el tiempo, el soporte y el rendimiento de HTML5 serán mayores en los diferentes dispositivos reduciendo en gran medida los problemas patentes en la actualidad.

CAPITULO 6. Conclusiones.

Finalmente en este capítulo vamos a resumir las conclusiones obtenidas del desarrollo de este proyecto.

6.1 Conclusiones.

Durante el desarrollo de este proyecto hemos utilizado la tecnología de reciente aparición HTML5 con el fin de orientarla al desarrollo de páginas web que puedan sustituir en un futuro a las aplicaciones móviles. Se han explorado diversas características relevantes de la misma como la posibilidad de interactuar con un servidor basado en Wordpress, dotar de un aspecto visualmente atractivo a la par que minimalista y funcional para la página o la capacidad de hacer persistir los datos entre diferentes sesiones en la misma.

Este proyecto nos ha servido para reforzar los conocimientos teóricos adquiridos en la asignatura estudio tecnológico [1] y ver mediante un ejemplo práctico la importancia de los diferentes *frameworks* Javascript y cómo pueden ayudar a los desarrolladores en el empleo de sus funciones. Además nos ha servido para coger confianza ante lo desconocido y para aprender la importancia de recabar toda la información posible antes de tomar decisiones respecto a cómo desarrollar o enfrentarnos a un problema.

También ha sido de utilidad el empleo de la tecnología HTML5 unida al *framework* Dojo en el diseño de una aplicación práctica para percatarnos de que ambas herramientas nos permiten funcionar de manera suficiente para la aplicación llevada a cabo, pero aún así, por ser una tecnología tan joven y encontrarse aún en fase de desarrollo, la funcionalidad que nos permiten emplear es bastante más reducida que la que se puede lograr si se trabaja nativamente en el lenguaje del dispositivo para el que se desarrolla.

La fase de testeo de la aplicación nos ha sido de gran relevancia para comprobar de primera mano el problema de la no estandarización que existe respecto a esta tecnología y al soporte que proporcionan de la misma los diferentes navegadores. Hemos sido conscientes de que no todos los navegadores funcionan de la misma manera y ha sido

necesario realizar modificaciones en nuestro diseño para adaptarnos al mayor número posible de los mismos. Cabe destacar además que, debido al soporte que nos proporciona Dojo Mobile, la aplicación funcionará sin ningún tipo de error en los navegadores basados en Webkit mientras que su funcionalidad se verá mermada en aquellos que no lo sean.

Por otro lado, es importante notar el ahorro en carga de trabajo que supone emplear una única tecnología en el diseño frente a tener que emplear diversas tecnologías en función del dispositivo para el que se desarrolla, tal y como hemos podido comprobar con la implementación de nuestra página.

Por último, el empleo de HTML5 nos ha servido, aparte de para adquirir conocimientos tecnológicos, para tomar consciencia de otros aspectos externos al propio desarrollo tecnológico de la aplicación pero igualmente intrínsecos y relevantes para el mismo. Estos son la importancia de la reducción de costes en el desarrollo de un proyecto de diseño software y tener claro el papel que juegan en este sector las grandes compañías monopolísticas.

APÉNDICE A. Presupuesto

1.- Autor: David Ortiz Sáez

2.- Departamento: Ing. Telemática

3.- Descripción del Proyecto:

- Titulo: **Aplicaciones Web Móviles con persistencia.**
- Duración (meses) **6**

Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):

17.060,99 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F.	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Díaz Asúa, Esteban		Ingeniero Senior	0,31	4.289,54	1.329,75	
Chambers, Hayden		Ingeniero Senior	0,05	4.289,54	214,45	
Galindo Sánchez, Luis Ángel		Ingeniero Senior	0,2	4.289,54	857,9	
Ortiz Sáez, David		Ingeniero	4	2.694,39	10.777,56	
		Hombres mes	**4,56**	**Total**	13.179,66	

a) 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas) Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPO

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
PC con Win. 7 y Office	800,00	100	6	60	93,33
Router ADSL	50	100	6	60	5,00
				Total	98,33

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Licencia Microsoft Office 2012 Hogar y Estudiante		89,00
Material oficina		50,00
Abono Transportes B2 (6 meses)		405,60
	Total	544,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	13.179,66
Amortización	98,33
Subcontratación de tareas	0,00
Costes Indirectos	3.783,00
Total	17.060,99

APÉNDICE B. Glosario

Android: Es un sistema operativo móvil basado en Linux, que está enfocado para ser utilizado en dispositivos móviles como smartphones tablets y otros dispositivos. Es desarrollado por la Open Handset Alliance, la cual es liderada por Google.

Apache: es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual.

Canvas: es un elemento HTML incorporado en HTML5 que permite la generación de gráficos dinámicamente. Permite generar gráficos estáticos y animaciones. Fue implementado por Apple para su navegador Safari. Más tarde fue adoptado por otros navegadores, como Firefox a partir de su versión 1.5 y Opera.

Chrome: Es un navegador web desarrollado por Google y compilado con base en componentes de código abierto como el motor renderizado WebKit y su estructura de desarrollo de aplicaciones (framework), disponible gratuitamente bajo condiciones de servicio específicas.

CSS3: CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML. El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o los navegadores.

Dojo: Es un framework que contiene APIs y widgets (controles) para facilitar el desarrollo de aplicaciones Web que utilicen tecnología AJAX. Contiene un sistema de empaquetado inteligente, los efectos de UI, drag and drop APIs, widget APIs, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX.

Dolphin Browser: Es un navegador web gratuito para dispositivos Android y iOS creado por la empresa MoboTap Inc. El motor de renderizado de Dolphin es WebKit.

DOM: ('Modelo de Objetos del Documento' o 'Modelo en Objetos para la Representación de Documentos') es esencialmente una interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos. A través del DOM, los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML, que es para lo que se diseñó principalmente.

Excerpt: En Wordpress es en esencia una propiedad auxiliary que puede ser incluida opcionalmente y que indica el resumen o las líneas principales de un post.

FireFox: Es un navegador web libre y de código abierto descendiente de Mozilla Aplication Suite y desarrollado por la Fundación Mozilla.

Framework: En el desarrollo de software un *framework* o *infraestructura digital*, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de *software*concretos, con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado.

Graceful Degradation: Es la propiedad que permite a un sistema (a menudo un sistema computacional) continuar funcionando correctamente incluso si se ha producido un fallo en alguno de sus componentes.

HTML: (Hyper Text Markup Language) Lenguaje de marcado predominante para la elaboración de páginas web. Es utilizado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

iOS: (anteriormente denominado iPhone OS) es un sistema operativo móvil de Apple. Originalmente desarrollado para el iPhone, siendo después usado en dispositivos como el iPod, iPad y el Apple TV. Apple, Inc. no permite la instalación de iOS en hardware de terceros. Tenía el 26% de cuota de mercado de sistemas operativos móviles vendidos en el último cuatrimestre de 2010, detrás de Google Android.

Javascript: Lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Existe también una forma de Javascript del lado del servidor (SSJS).

JSON: Acrónimo de *JavaScript Object Notation*, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

Localhost: En informática, en el contexto de redes TCP/IP, **localhost** es un nombre reservado que tienen todas las computadoras, router o dispositivo independientemente de que disponga o no de una tarjeta de red ethernet. El nombre **localhost** es traducido como la dirección IP de *loopback* **127.0.0.1** en IPv4, o como la dirección **::1** en IPv6.

Markup: Un **lenguaje de marcado** o **lenguaje de marcas** es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación. El lenguaje de marcas más extendido es el HTML “HyperText Markup Language".

MovilForum: Es una comunidad abierta orientada a pequeñas empresas tecnológicas, desarrolladores profesionales y Start-ups, para la creación de nuevas

aplicaciones de movilidad que permitan la integración de las comunicaciones móviles en Internet. La iniciativa está gestionada por Telefónica España.

MVC: Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos. El patrón de llamada y retorno MVC (según CMU), se ve frecuentemente en aplicaciones web, donde la vista es la página HTML.

MySQL: Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones.

PHP: Es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Se usa principalmente para la interpretación del lado del servidor (*server-side scripting*) pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica usando las bibliotecas Qt o GTK+.

Plugin: Es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

Responsive Design: El **diseño web adaptativo** o adaptable (en inglés, *Responsive Web Design*) es una técnica de diseño y desarrollo web que mediante el uso de estructuras e imágenes fluidas, así como de media-queries en la hoja de estilo CSS, consigue adaptar el sitio web al entorno del usuario.

Safari: Es un navegador web de código cerrado desarrollado por Apple Inc. Está disponible para Mac OS X, iOS y Microsoft Windows.

SO: Un **sistema operativo** (SO) es un programa o conjunto de programas que en un sistema informático gestiona los recursos de hardware y provee servicios a los programas de aplicación, ejecutándose en modo privilegiado respecto de los restantes.

WebKit: Es una plataforma para aplicaciones que funciona como base para el navegador web Safari, Google Chrome, Epiphany, Maxthon,Midori,Qupzilla entre otros.

Widgets: En informática, un ***widget*** es una pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de *widgets* o *Widget Engine*. Entre sus objetivos están dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

Wordpress: Es un sistema de gestión de contenido (en inglés *Content Management System*, o CMS) enfocado a la creación de blogs (sitios web periódicamente actualizados). Desarrollado en PHP y MySQL.

XML-RPC: Es un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos y HTTP como protocolo de transmisión de mensajes.

APÉNDICE C. Referencias e hiperenlaces

[1] David Ortiz Sáez. 2011. Apuntes asignatura Estudio tecnológico. Estudio sobre el empleo de la tecnología HTML5 en el diseño de aplicaciones móviles. https://dl.dropbox.com/u/27162282/Estudio%20Tecnol%C3%B3gico.pdf *Obtenido el 22/05/2012.*

[2] Ben Frain. Packt Publishing. 2012. Responsive Web Design with HTML5 and CSS3. *Obtenido el 22/05/2012.*

[3] Chuck Hudson & Tom Leadbetter. Pearson Education. 2012.HTML5 Developer's Cookbook. *Obtenido el 22/05/2012.*

[4] Díaz Asúa, E., 8 Abril 2011. HTML5. El estado de las cosas (I). Blog Movilforum. http://blog.movilforum.com/html5-el-estado-de-las-cosas-i/ *Obtenido el 23/05/2012.*

[5] Díaz Asúa, E., 12 Abril 2011. HTML5. El estado de las cosas (II). Blog Movilforum. http://blog.movilforum.com/html5-el-estado-de-las-cosas-ii/ *Obtenido el 23/05/2012.*

[6] WHATwg Community. http://www.whatwg.org/ *Obtenido el 23/05/2012.*

[7] Franganillo, J., 6 Septiembre 2010. HTML5: El nuevo estándar básico de la Web. http://franganillo.es/html5.pdf *Obtenido el 23/05/2012.*

[8] Campos, H., 15 Febrero 2011. HTML5 será estandarizado hasta 2014. http://blog.neubox.net/index.php/2011/02/html5-sera-estandarizado-hasta-el-2014/ *Obtenido el 23/05/2012.*

[9] Crockford, D., 14 Agosto 2008. The only thing we have to fear is premature standardization. http://yuiblog.com/blog/2008/08/14/premature-standardization/ *Obtenido el 23/05/2012.*

[10] HTML5 para principiantes. http://buenasideaspy.com/2010/10/html5-para-principiantes *Obtenido el 23/05/2012.*

[11] Wikipedia; **"Persistencia (informática)"**; http://es.wikipedia.org/wiki/Persistencia_(inform%C3%A1tica) *Obtenido el 24/05/2012.*

[12] Simms, C., 18 Mayo 2011. HTML5 Storage Wars - localStorage vs. IndexedDB vs. Web SQL. http://csimms.botonomy.com/2011/05/html5-storage-wars-localstorage-vs-indexeddb-vs-web-sql.html *Obtenido el 24/05/2012.*

[13] Wikipedia; **"Webkit"**; http://es.wikipedia.org/wiki/WebKit Obtenido el 07/08/2012 *Obtenido el 24/05/2012.*

[14] Mozilla Developer Network; **"Javascript"**; https://developer.mozilla.org/en-US/docs/JavaScript *Obtenido el 24/05/2012.*

[15] Crockford, D., 3 Marzo 2008. The World's misunderstood programming languaje has become the World's most popular programming language. http://javascript.crockford.com/popular.html *Obtenido el 24/05/2012.*

[16] Canada, L. 24 Septiembre 2011. Evolución de búsqueda del término jQuery en Google. Blog Luiscanada.com. http://luiscanada.com/blog/comparativa-frameworks-javascript/ *Obtenido el 26/05/2012.*

[17] Glenn E. Krasner y Stephen T. Pope; A Description of the Model-View-Controller User; http://www.itu.dk/courses/VOP/E2005/VOP2005E/8_mvc_krasner_and_pope.pdf *Obtenido el 26/05/2012.*

[18] Zyp, K., 24 Julio 2011. jQuery vs Dojo vs Mootools – DOM. http://jsperf.com/jquery-vs-dojo-vs-mootools-dom/17 *Obtenido el 27/05/2012.*

[19] Wordpress. http://es.wordpress.com/ *Obtenido el 27/05/2012.*

[20] Spyrestudios; **"40 WordPress-Powered Websites With Awesome Designs"**; http://spyrestudios.com/wordpress-powered-websites/ *Obtenido el 27/05/2012.*

[21] Jos. 8 Agosto 2010. Instalar Apache + PHP + MySQL en Windows 7. http://josmx.com/apache-php-mysql-en-windows-7 *Obtenido el 27/05/2012.*

[22] MySQL; **"3.3.1 Crear y seleccionar una base de datos"**; http://dev.mysql.com/doc/refman/5.0/es/creating-database.html *Obtenido el 27/05/2012.*

[23] WordPress.org; **"Codex es: instalando Wordpress"**; http://codex.wordpress.org/es:Instalando_Wordpress *Obtenido el 27/05/2012.*

[24] Dojo. Documentation; **"Dojo Mobile"**; http://dojotoolkit.org/reference-guide/1.7/dojox/mobile.html#dojox-mobile *Obtenido el 29/05/2012.*

[25] Wikipedia; **"Trident"**; http://es.wikipedia.org/wiki/Trident_(motor_de_navegaci%C3%B3n). *Obtenido el 29/05/2012.*

[26] Mozilla Developer Network; **"Gecko"**; https://developer.mozilla.org/es/docs/Gecko *Obtenido el 29/05/2012.*

[27] Lindley, C. 29 Marzo 2010. Understanding dojo.require. Sitepen. http://www.sitepen.com/blog/2010/03/29/understanding-dojo-require/ *Obtenido el 29/05/2012.*

[28] Wikipedia; **"SOAP"**; http://es.wikipedia.org/wiki/Simple_Object_Access_Protocol *Obtenido el 29/05/2012.*

[29] XML-RPC.com. http://xmlrpc.scripting.com/default.html *Obtenido el 29/05/2012.*

[30] Wikipedia; **"SMTP"**; http://es.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol *Obtenido el 29/05/2012.*

[31] Wikipedia; **"QoS"**; http://es.wikipedia.org/wiki/Calidad_de_servicio *Obtenido el 29/05/2012*

[32] Rodriguez, A., 6 Noviembre 2008, RESTful WebServices: The basics. IBM developer works. https://www.ibm.com/developerworks/webservices/library/ws-restful/ *Obtenido el 29/05/2012.*

[33]Navarro Marset, R., REST vs Web Services. Universidad Politécnica de Valencia. http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf *Obtenido el 30/05/2012.*

[34] Pautasso, C., Zimmerman, O., Leymann, F., RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. http://www2008.org/papers/pdf/p805-pautassoA.pdf *Obtenido el 30/05/2012.*

[35] Dojo. Documentation; **"Dojo.io.script"**; http://dojotoolkit.org/reference-guide/1.7/dojo/io/script.html *Obtenido el 30/05/2012.*

[36] Wordpress.org. Plugin Directory; **"JSON API"**; http://wordpress.org/extend/plugins/json-api/other_notes/ *Obtenido el 30/05/2012.*

[37] Movilforum España. http://espana.movilforum.com/ *Obtenido el 01/06/2012.*

[38] Blog Movilforum España. http://blog.movilforum.com/ *Obtenido el 01/06/2012.*

[39] Wikipedia; **"Hojas de estilo en cascada"**; http://es.wikipedia.org/wiki/Hojas_de_estilo_en_cascada *Obtenido el 01/06/2012.*

[40] Css Tricks. A web Design Community. http://css-tricks.com/ *Obtenido el 01/06/2012.*

[41] W3SCHOOLS; **"HTML5 style Attribute"**; http://www.w3schools.com/html5/att_global_style.asp *Obtenido el 01/06/2012.*

[42] Wordpress.org; **"Codex Excerpt"**; http://codex.wordpress.org/Excerpt *Obtenido el 01/06/2012.*

[43] Dojo. Documentation; **"Dojox.mobile.button"**; http://dojotoolkit.org/reference-guide/1.7/dojox/mobile/Button.html *Obtenido el 03/06/2012.*

[44] Dojo; **"Get Dojo"**; http://dojotoolkit.org/download/ *Obtenido el 03/06/2012.*

[45] Wikipedia; **"Persistence (computer science)"**; http://en.wikipedia.org/wiki/Persistence_(computer_science) *Obtenido el 05/06/2012.*

[46] W3C; **“HTML5 differences from HTML4”**; http://www.w3.org/TR/html5-diff/#new-apis *Obtenido el 05/06/2012.*

[47] Dive into HTML5; **“Chapter 8. Let’s take this offline”**; http://diveintohtml5.info/offline.html *Obtenido el 05/06/2012.*

[48] IBM Developer Works; **“Enable HTML5 Offline Web Application with Dojo 1.7.”**; https://www.ibm.com/developerworks/mydeveloperworks/blogs/94e7fded-7162-445e-8ceb-97a2140866a9/entry/draft_enable_html5_offline_web_application_with_dojo_1_716?lang=en *Obtenido el 10/06/2012.*

[49] Wikipedia; **“XMLHttpRequest”**; http://en.wikipedia.org/wiki/XMLHttpRequest *Obtenido el 10/06/2012.*

[50] Z.Hemel; Persistence.js; Readme file; https://github.com/zefhemel/persistencejs#readme *Obtenido el 10/06/2012.*

[51] Persistence.js.; **“Schema Definition”**; http://persistencejs.org/schema *Obtenido el 10/06/2012.*

[52] Spaceport PerfMarks Report II. HTML5 Performance on Desktop vs Smartphones. Mayo 2012 http://spaceport.io/spaceport_perfmarks_2_report_2012_5.pdf *Obtenido el 21/06/2012.*

[53] HTML5 test; **“Comparativa navegadores móviles”**; http://html5test.com/compare/browser/android23/android40/ios50.html *Obtenido el 21/06/2012.*

[54] Modernizr. http://modernizr.com/ *Obtenido el 21/06/2012.*

[55] Super Mario Bros HTML5. http://www.florian-rappl.de/html5/projects/SuperMario/ *Obtenido el 21/06/2012.*

[56] Wayner, P. 15 Agosto 2011. 11 hard truths about HTML5. Infoworld. http://www.infoworld.com/d/html5/11-hard-truths-about-html5-169665?page=0,0 *Obtenido el 23/06/2012.*

[57] Siliconnews.es. Apps nativas vs apps HTML5: La guerra ha comenzado. 20 Mayo 2012. http://www.siliconnews.es/2012/05/20/apps-nativas-vs-apps-html5-la-guerra-ha-comenzado/ *Obtenido el 23/06/2012.*

[58] Siliconnews.es. La fragmentación de Android condensada en una imagen. 16 Mayo 2012. http://www.siliconnews.es/2012/05/16/la-fragmentacion-de-android-condensada-en-una-imagen/ *Obtenido el 23/06/2012.*

[59] One-click Payments. Paylane. http://paylane.com/one-click-payments *Obtenido el 27/06/2012.*

[60] Financial Tymes. http://www.ft.com *Obtenido el 27/06/2012.*

[61] La Vanguardia. http://www.lavanguardia.com *Obtenido el 27/06/2012.*

[62] Cone, A., 14 Febrero 2011. HTML5 pushed back until 2014. Neowin.net. http://www.neowin.net/news/html5-pushed-back-until-2014 *Obtenido el 27/06/2012.*

[63] Siliconnews.es. La nueva versión de Firefox ya está disponible para Android. 27 Junio 2012. http://www.siliconnews.es/2012/06/27/la-nueva-version-firefox-ya-esta-disponible-para-android/ *Obtenido el 27/06/2012.*

[64] Wikipedia; **"Frame rate"**; http://en.wikipedia.org/wiki/Frame_rate Obtenido el 27/06/2012

[65] Ideateca; **"Juegos HTML5"**; http://www.ludei.com/ *Obtenido el 27/06/2012.*