Damián De Luca

Apps HTML5 para móviles

Desarrollo de aplicaciones para *smartphones* y *tablets* basado en tecnologías Web

Apoyo en la

Alfaomega

ITMaster
PROFESSIONAL TRAINING

Apps HTML5 para móviles

Damián De Luca

Apps HTML5 para móviles

Desarrollo de aplicaciones para smartphones y tablets basado en tecnologías Web

Damián De Luca

Alfaomega

Buenos Aires • Bogotá • México, D.F. • Santiago de Chile

Bajalibros.com
ISBN 978-607-62-2038-2

De Luca, Damián
Apps HTML5 para móviles. Desarrollo de aplicaciones para smartphones y tablets basado en tecnologías Web- 1a ed. - Buenos Aires : Alfaomega Grupo Editor Argentino, 2014.

ISBN 978-607-62-2038-2
1. Informática. 2. Programación. 3. Teléfonos Móviles. I. Título
CDD 004

Edición: Damián Fernández
Corrección de estilo: Adriana Scaglione
Revisión de armado: Vanesa García
Diseño de tapa: Iris Biaggini

Internet: http://www.alfaomega.com.mx

Paraguay 1307, PB, oficina 11
ISBN 978-607-62-2038-2
Queda hecho el depósito que prevé la ley 11.723

Empresas del grupo:
Argentina: Alfaomega Grupo Editor Argentino S. A.
Paraguay 1307 P.B. “11”, Buenos Aires, C.P. 1057
Tel.: (54-11) 4811-7183 / 0887
E-mail: ventas@alfaomegaeditor.com.ar
México: Alfaomega Grupo Editor S. A. de C.V.
Pitágoras 1139, Col. Del Valle, México, D.F., C.P. 03100
Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396
E-mail: atencionalcliente@alfaomega.com.mx
Colombia: Alfaomega Colombiana S. A.
Calle 62 N° 20-46, Bogotá, D.C.
Tel.: (571) 7460102 - Fax:(571) 2100415
E-mail: cliente@alfaomega.com.co
Chile: Alfaomega Grupo Editor S. A.
Av. Providencia 1443. Oficina 24. Santiago de Chile
Tel. (562) 2235-4248 / (562) 2947-9351 Fax. (562) 2235 5786
E-mail: agechile@alfaomega.cl

A todos los que están a mi lado, en las buenas y en las malas, hoy y siempre. A mis padres, Viviana y Horacio; a mi hermano, Alejandro Arturo; a mis abuelos; a mi mujer Myriam y a mi hijo Gael, que todos los días me ilumina como el Sol con su sonrisa.

Un agradecimiento para todos aquellos que confiaron en mí para llevar adelante y hacer realidad este libro, especialmente a las personas que me permitieron potenciar esta obra con sus consejos.

Mensaje del editor

Los conocimientos son esenciales en el desempeño profesional, sin ellos es imposible lograr las habilidades para competir laboralmente. La universidad o las instituciones de formación para el trabajo ofrecen la oportunidad de adquirir conocimientos que serán aprovechados más adelante en beneficio propio y de la sociedad; el avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos. Cuando se toma la decisión de embarcarse en una vida profesional, se adquiere un compromiso de por vida: mantenerse al día en los conocimientos del área u oficio que se ha decidido desempeñar.

Alfaomega tiene por misión ofrecerles a estudiantes y profesionales conocimientos actualizados dentro de lineamientos pedagógicos que faciliten su utilización y permitan desarrollar las competencias requeridas por una profesión determinada. Alfaomega espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Alfaomega hace uso de los medios impresos tradicionales en combinación con las tecnologías de la información y las comunicaciones (TIC) para facilitar el aprendizaje. Libros como éste tienen su complemento en una página Web, en donde el alumno y su profesor encontrarán materiales adicionales.

Esta obra contiene numerosos gráficos, cuadros y otros recursos para despertar el interés del estudiante, y facilitarle la comprensión y apropiación del conocimiento. Cada capítulo se desarrolla con argumentos presentados en forma sencilla y estructurada claramente hacia los objetivos y metas propuestas.

Los libros de Alfaomega están diseñados para ser utilizados dentro de los procesos de enseñanza-aprendizaje, y pueden ser usados como textos para diversos cursos o como apoyo para reforzar el desarrollo profesional.

Alfaomega espera contribuir así a la formación y el desarrollo de profesionales exitosos para beneficio de la sociedad.

Acerca del autor

Damlán De Luca

Es experto en HTML5 y cuenta con una importante experiencia en el desarrollo de Web Apps y soluciones enfocadas en tecnologías Web. Entre sus trabajos más recientes, se destacan aplicaciones para móviles basadas en HTML5 + PhoneGap/Cordova.

Es profesor de la carrera Programador Web y de los cursos HTML5 para Diseñadores, Programador HTML5 y Mobile Web con HTML5 en ITMaster Professional Training. También ha brindado charlas y capacitaciones en importantes empresas en diferentes ciudades de la región.

Es autor de otros cuatro libros, entre los que se destacan Webmaster Profesional (2009) y HTML5 (2011) de editorial Users. Además, ha participado como autor y editor de colecciones de fascículos, revistas y contenidos online relacionados con informática, diseño y desarrollo Web.

Realiza publicaciones vinculadas con el mundo del desarrollo en su sitio Web personal http://damiandeluca.com.ar y se lo puede seguir a través de su cuenta de Twitter: @damiande.

Acceso al material complementario

Para tener acceso al material complementario del libro Apps HTML5 para móviles, es necesario:

1. Ir a la página: http://libroweb.alfaomega.com.mx/

2. Registrarse como usuario y guarde en un lugar seguro su nombre de usuario y clave de acceso para emplearla para futuros ingresos.

3. Ingrese con su usuario a la sección de libros.

4. Busque y seleccione la imagen correspondiente a este libro para descargar su material complementario como:

- Código fuente
- Fe de erratas

Prólogo

¿Les gusta viajar? Creo que esta pregunta tendrá una respuesta afirmativa de la gran mayoría de los lectores que relacionan los viajes con placer o motivaciones personales positivas, aunque quizás algunos no comprendan aún como esto se vincula con la obra que se encuentra en este momento en sus manos. Paso a explicarlo.

Cuando comencé a pensar la idea inicial que daría vida a esta obra imaginé un viaje, una manera de poder transmitirle al lector una serie de experiencias que lo puedan potenciar como profesional y le permitan comenzar a pensar distinto el mundo móvil. Desde mi punto de vista, viajar es, de alguna manera, un medio para conocer y comprender nuevos aspectos del mundo que nos rodea.

Partiendo de esta premisa, las páginas de este libro nos sumergen en un recorrido por las nuevas tecnologías que están cambiando nuestro tiempo. Nos abre la puerta para aprender nuevas técnicas de desarrollo y repensar lo que veníamos haciendo. Salir de la mecánica absoluta del código repetitivo y animarnos a pensar distinto.

En algunos pasajes de este libro nos pondremos en el lugar del usuario, para tratar de comprender qué piensa y qué necesita. Esta obra resalta la importancia de no perder de vista nunca las necesidades del usuario y, como desarrolladores, comprender la diversidad de personas que pueden acceder a nuestros productos. Hacer el intento de ponernos en los zapatos del usuarios puede resultar un ejercicio muy saludable en pos de llegar a una solución adecuada y efectiva para una necesidad planteada en un proyecto. Usabilidad y accesibilidad no son solo teoría, son esencialmente conceptos que hay que aprender a llevar a la práctica.

A lo largo de las páginas que componen esta aventura, nos situaremos en los puntos clave de este misterioso universo, aquellos que todo profesional necesita conocer a la hora de llevar sus proyectos a los dispositivos móviles. Y en este sendero también descubriremos que no hay un solo camino para llevar al final del recorrido con éxito.

En síntesis, este libro puede ser el boleto que los lleve a pensar el universo del desarrollo de aplicaciones Web para dispositivos móviles de otra forma.

El mundo que conocimos se encuentra en una era basada en cambios permanentes. Las computadoras personales han comenzado a abandonar su reinado, dejando el paso a dispositivos más pequeños y compactos, pero no por eso menos confortables. ¿Cuál es la mejor solución para desarrollar en estas plataformas?

Para conocer la respuesta pónganse cómodos, prepárense para una nueva experiencia y, por sobre todas las cosas, disfruten de este viaje.

Damián De Luca

Alfaomega e ITMaster Professional Training te dan la posibilidad de que certifiques tus conocimientos y experiencias adquiridos como lector de este libro mediante una evaluación gratuita. Su aprobación te permitirá tener la certificación HTML5 para móviles.

Luego de la obtención del certificado, podrás continuar tu formación en la carrera corta de Programador HTML5 y en los cursos cortos PhoneGap, jQuery Mobille, Programación para iPhone y Android para completar tus conocimientos en tecnologías relacionadas con web móvil.

La capacitación podrás realizarla en forma presencial en las sedes habilitadas o a distancia a través de Internet con una capacitación online, en cualquier ciudad del mundo donde te encuentres.

Para dar la evaluación de certificación ingresa en la dirección correspondiente a tu país para recibir mayor información. Para realizar cualquier consulta adicional, se detalla la dirección de correo electrónico que corresponde a tu país

España – http://libros.itmaster.es – info@itmaster.es

México – http://libros.itmaster.com.mx – info@itmaster.com.mx

Argentina – http://libros.itmaster.com.ar – info@itmaster.com.ar

Otro País – http://libros.itmaster.la – info@itmaster.la

Introducción al mundo móvil

1

El mundo de los móviles

Nos encontramos en una época en la que las comunicaciones y el uso de las redes se han transformado en factores clave de nuestra vida. De acuerdo con las estadísticas mundiales de mediados de 2012, publicadas por Internet World Stats (http://www.internetworldstats.com/stats.htm), la penetración de Internet a nivel global supera el 34%. El número de los usuarios que se conectan a esta red se estima cercano a los siete mil millones, y continúa expandiéndose.

En algunas regiones del mundo se registra un alto nivel de penetración, como el caso de Norteamérica (78,6%) y Europa (63,2%). América Latina ostenta un 42,9% de penetración con un importante crecimiento en la última década. Un indicador interesante es el que nos marca que casi el 45% de los usuarios que utilizan Internet lo hacen desde países asiáticos, pero en esa región, debido al gran nivel poblacional, el grado de penetración que tiene la gran red de redes apenas supera el 27%.

Los dispositivos móviles (*smartphone*, *tablets* y demás variantes) son utilizados a diario por millones de usuarios en todo el mundo, convirtiéndose en piezas fundamentales de la vida digital moderna.

Según un informe publicado por el sitio StatCounter, que podemos leer en http://gs.statcounter.com/press/mobile-Internet-usage-is-doubling-year-on-year, de 2009 a 2012, el acceso a Internet desde dispositivos móviles se ha duplicado cada año. Los datos del primer semestre de 2013, indican que más del 17% del tráfico en la Web proviene de dispositivos móviles.
Las perspectivas marcan un crecimiento en el mercado de los móviles. Este sector nuclea a una importante masa de público que demanda contenidos Web de calidad,

pensados especialmente para dichos dispositivos, y también, la llegada de una nueva generación de aplicaciones para estos entornos. En el último lustro se ha comenzado a modificar el mapa de acceso a Internet y los dispositivos móviles se posicionan con mayor fuerza en este mundo.

Ya no podemos pensar en una Web que funciona solo en computadoras de escritorio ni en aplicaciones exclusivas para ciertos entornos. La nueva misión de los desarrolladores es ir más allá y adaptarse a las nuevas reglas de juego, que incluyen diversidad de dispositivos y plataformas.

Para satisfacer las necesidades de los usuarios, debemos comprender que ya no podemos esperar que las personas se ajusten a lo que nosotros les ofrecemos, sino que los desarrollos ahora deben estar preparados para adaptarse a los nuevos entornos que se encuentran disponibles en el mercado.

En este libro, comenzaremos un viaje que arrancará por develar las bases y conceptos de esta nueva realidad y nos llevará a recorrer las posibilidades que tenemos hoy en día para crear aplicaciones para dispositivo móviles.

Breve historia de la Web móvil

Si a una persona mayor de cuarenta años le preguntáramos cuál es su primer recuerdo de un teléfono móvil, es muy probable que su mente le traiga como respuesta la imagen de Maxwell Smart, interpretado por Dom Adams, hablando con el teléfono ubicado en su zapato, en la recordada serie *Get Smart* (conocida como *El Superagente 86* por el público hispanohablante). Y si la referencia fuera hacia las *tablets*, quizás muchas personas recordarían los modelos con perillas y luces que se veían en la serie de ciencia ficción *Star Trek* (*Viaje a las estrellas*), creada también en los años sesenta.

Lejos de los gratos recuerdos de las series de televisión, los primeros teléfonos móviles se crearon en la década de los cuarenta, en plena Segunda Guerra Mundial y funcionaban utilizando ondas de radio. Eran bastante distintos a los que conocemos ahora.

El primer éxito comercial de los teléfonos móviles en el mundo se remonta a la década de los ochenta. Estos primeros móviles masivos hoy son denominados como primera generación o (1G). En esos tiempos eran dispositivos de gran tamaño, comparados a los de hoy en día, y ofrecían funciones de telefonía básica ya que la Web llegaría algunos años después a este mundo.

En los noventa llega la era 2G y la digitalización, que permite una mejor calidad en las llamadas. Los dispositivos comienzan a reducir su tamaño y el éxito global de los teléfonos móviles continúa su importante crecimiento, imponiendo el estándar GSM. A partir de esta generación, llega la posibilidad de enviar mensajes de texto (SMS).

Internet comienza su andar en los móviles hacia fines de la década del noventa, gracias a la primera versión protocolo de aplicaciones inalámbricas (WAP). Si bien puede considerarse una opción limitada para lo que hoy en día estamos acostumbrados a ver en la Web móvil, de la mano del WML (Wireless Markup Language) encontramos el primer lenguaje de marcado para dispositivos móviles

Con 2.5G se experimenta un incremento en la velocidad de transmisión de datos, gracias a tecnologías como GPRS y EDGE. También se destaca por un sistema de mensajes mejorado (EMS) y la opción de mensajes multimedia (MMS).

En el año 2004, con WAP 2.0, se gestaría una gran renovación en la Web móvil, respaldada por el uso de XHTML Mobile Profile (XHTML-MP), ofreciendo mejores opciones para el maquetado de la de las páginas, trabajo con estilos y contenidos con imágenes.

Presente y futuro de la Web móvil

La tecnología 3G presenta una evolución imprescindible para los usuarios de dispositivos móviles, que comenzaban a requerir el uso de opciones de telefonía y servicios de Internet de una manera integrada, cómoda y fluida, desde cualquier lugar.

La línea evolutiva continúa con 4G, la generación que comienza a imponerse como alternativa para una experiencia más completa y potente en los dispositivos móviles y *tablets*. Además del uso de datos y voz, son pilares de esta generación la transmisión de video HD y la televisión móvil en alta definición, entre otras características.

Con el notable crecimiento que ha experimentado la Web móvil, mejorando sus opciones de conectividad, y con navegadores que pueden mostrar los sitios pensados para pantallas y equipos más grandes, la pregunta que muchas personas se hacen es: "¿Sigue siendo necesario crear versiones para móviles?".

Aquí la respuesta resulta más que evidente: si nuestra intención es brindarles a todos los usuarios de *smartphones* y *tablets* una experiencia acorde al contexto y expectativas que pueden tener cuando utilizan estos dispositivos, definitivamente tendremos que "pensar en móvil". Y esto no quiere decir solo tratar de que las piezas encajen en una pantalla más pequeña, sino que también implica analizar las necesidades y posibilidades de una manera diferente de la habitual para implementaciones *desktop*.

El mundo móvil tiene sus propias reglas y un estilo bien definido. Tanto desarrolladores como diseñadores deben aprender a crear sitios y aplicaciones para estas plataformas, ofreciendo a los usuarios la mejor experiencia de uso que sea posible.

Para comprender cómo planificar y resolver estas nuevas necesidades, primero debemos conocer la variedad reinante en cuanto a dispositivo, sistemas, navegadores y demás características. Sobre esto profundizaremos en el presente capítulo.

Cuando pensamos en dispositivos móviles que cuentan con capacidad de navegación por Internet, además de los teléfonos y las *tablets*, podemos destacar también los reproductores multimedia (por ejemplo iPod Touch) y los E-book Reader (como el caso de Amazon Kindle).

Los dispositivos

El universo de los dispositivos móviles se caracteriza por la diversidad. Si lo observamos desde el punto de vista usuario, podríamos pensar que es una ventaja, ya que se abre un amplio abanico de alternativas a la hora de elegir. Pero desde la mirada del desarrollador, la fragmentación existente en el mercado es un gran desafío, ya que nos obliga a estar informados y a contar con las herramientas y conocimientos necesarios para afrontar nuestros proyectos. Esto, además del software de desarrollo y el dominio de las diversas tecnologías, también se refiere a tener los dispositivos adecuados para realizar el *testing* de nuestros trabajos.

Otro aspecto que no debemos olvidar cuando hablamos de *smartphone*s y *tablet*s es que, por lo general, encontraremos dispositivos más limitados en cuanto a hardware (procesador, memoria RAM y espacio de almacenamiento) que en los equipos de escritorio. Si bien es verdad que estas características se mantienen en constante evolución, aun hoy debemos contemplar que en este universo los recursos de hardware pueden ser más limitados, al igual que algunas opciones de conexión a Internet.

La primera gran división que podríamos hacer entonces, es establecer dos grandes grupos, conformados por *smartphone*s y *tablet*s. Si bien entre ambos existen similitudes, también hay diferencias de tamaño, opciones de telefonía y también en algunas aplicaciones desarrolladas especialmente para alguno de estos grupos.

Si analizamos los dispositivos en cuanto a funcionalidades, capacidad y calidad de hardware los podríamos sectorizar en baja, media y alta gama. Entre los usuarios de gama media y alta, encontraremos la mayor adhesión al uso de Internet y también a adquirir aplicaciones para el dispositivo.

Otra forma de analizar el mercado es agrupando por marcas y modelos. Esto influye tanto en las características técnicas de hardware de los dispositivos como también del sistema operativo que viene preinstalado en cada uno. Algunos de los fabricantes con mayor presencia en el mercado de *smartphone*s son: Nokia, Apple, Samsung, BlackBerry, LG y HTC. En cuanto al universo de las *tablets*, podemos mencionar a: Apple, Motorola, Samsung, Sony, BlackBerry y ASUS. Vale la pena

destacar que además de las empresas mencionadas, en ambos rubros, podremos encontrar gran variedad de otras marcas que participan de este mercado.

A continuación vamos a profundizar sobre los puntos clave que nos ayudarán a comprender de qué manera está conformado el ecosistema de dispositivos móviles y, a su vez, cuáles son los aspectos principales para comenzar a observar en función de nuestros próximos proyectos para estas plataformas.

Los sistemas operativos para móviles

Así como encontramos diversidad en los modelos de dispositivos, también encontraremos variedad en los sistemas operativos para móviles.

El sistema operativo en los móviles define varias características importantes para un desarrollo, ya que pueden cambiar las herramientas y el lenguaje con el cual se generan las aplicaciones nativas. Si pensamos en el desarrollo Web para móviles, podremos observar que cada sistema operativo cuenta con un navegador que nos llega de fábrica como predeterminado, y que nos encontramos con la posibilidad de optar por instalar alternativas de otros desarrolladores, si lo deseamos.

A continuación veremos un breve detalle sobre los principales sistemas operativos para *smartphones* y *tablets*:

- **Android:** su desarrollo está en manos de Open Handset Alliance, alianza con más de ochenta empresas, con Google como actor principal de este proyecto. Android es el sistema operativo más utilizado en la actualidad por *smartphones* y *tablets*, con más de la mitad del mercado. Cuenta con adaptaciones para dispositivos de diversos fabricantes. Un dato para pensar si vamos a desarrollar para Android es las versiones que aún están vigentes y el API Level de cada una de ellas:

  (http://developer.android.com/about/dashboards/index.html).

- **iOS:** es el sistema que Apple creó para sus dispositivos móviles, originalmente para iPhone, y luego lo extendió para iPod Touch y también para iPad. Es el segundo sistema operativo móvil más utilizado, con algo más de un cuarto del mercado. Es importante destacar que iOS es un sistema que solamente funciona en dispositivos Apple y no se encuentra disponible para otros fabricantes.

- **Symbian:** es un sistema con una larga tradición en el mundo de los móviles. Nokia, Sony Ericsson y Motorola desarrollaron por años muchos dispositivos exitosos en el mercado que se han apoyado en la confiabilidad de Symbian. Sin embargo, algunas decisiones estratégicas como la de Nokia eligiendo Windows Phone para sus nuevos dispositivos están marcando el principio del fin para Symbian.

- **Windows Phone:** es el sistema que actualmente desarrolla Microsoft para dispositivos móviles. Ha sido adoptado por dispositivos de diversas empresas de móviles en todo el mundo, entre ellas: Acer, Dell, HTC, Nokia, Samsung y LG. Previo al lanzamiento de Windows Phone, Microsoft desarrolló otros sistemas tales como Windows CE y Windows Mobile. Estas versiones antiguas ya no cuentan con muchas unidades en el mercado activo y sus aplicaciones no son compatibles con las del nuevo sistema operativo.
- **BlackBerry OS:** es la denominación del sistema operativo instalado en la línea de teléfonos Blackberry. Su nacimiento data de fines de los años noventa y sus diferentes versiones acompañaron la evolución de los *smartphone*s de RIM, empresa que desde el año 2013 pasó a llamarse simplemente BlackBerry. La llegada de la *tablet* PlayBook, en el año 2011, también marcó el arribo de un nuevo sistema operativo: BlackBerry Tablet OS.
- **webOS:** es un sistema creado por Palm en el año 2009. Posteriormente pasa a manos de Hewlett-Packard bajo el nombre de HP webOS. En 2011, se anunció que no se continuaría con la fabricación de dispositivos que incluyan este sistema operativo, sin embargo se mantiene el soporte y se ha modificado su modalidad de licencia para que sea software libre.
- **Firefox OS:** es el nombre con el que se ha lanzado el sistema operativo preparado por Mozilla Corporation. Es desarrollado bajo el modelo código abierto, se basa en núcleo Linux, apoyándose en las virtudes del motor Gecko (el mismo que emplea Firefox). Se destaca por ser una opción liviana para dispositivos de hardware limitado. Su fortaleza se basa en los estándares Web y su capacidad de correr aplicaciones creadas en HTML5.

En el mercado, podremos encontrar otros sistemas operativos para móviles, como el caso de MeGoo (sucesor de Maemo y Moblin) y Bada. Vale la pena destacar también que existen distribuciones de Linux adaptadas como sistemas operativos para móviles. Esta última opción se puede ver con mayor presencia en mercados asiáticos, como los casos de China y Japón, por citar algunos ejemplos.

Fig. 1.1. Aplicaciones móviles corriendo en iPhone y en iPad.

Los navegadores

En el mundo de las computadoras de escritorio, las preferencias de los usuarios en cuanto a navegadores se encuentran enfocadas principalmente en Internet Explorer, Google Chrome, Mozilla Firefox y Apple Safari. Si bien existen otras opciones, estos son los nombres que dominan el mercado.

En el caso de los móviles nos encontramos con muchas opciones diferentes. En la actualidad existen más de cuarenta alternativas, entre lo que se conoce como *browsers* y *pseudo-browers* para smartphones y *tablets*.

También podremos hacer una división entre los navegadores que se encuentran preinstalados junto a los sistemas operativos móviles y aquellos que pueden ser instalados por el usuario. Veamos las principales opciones:

- **Android Browser:** es el navegador disponible en Android desde sus primeras versiones hasta la versión 4. Desde sus inicios ha evolucionado en el soporte de nuevas tecnologías, pero finalmente deja paso a Google Chrome, como su sucesor en plataformas móviles.

- **Google Chrome:** disponible en Google Play para sistemas Android 4 o superior, se convierte en el *browser* por defecto a partir de Android 4.1. Vale la pena señalar que también se puede obtener de manera gratuita en el Apple Store, para dispositivos con iOS, pero en este caso Chrome utiliza UIWebView para realizar la representación de las páginas Web, es decir, actúa como un *pseudo browser*.
- **Safari para iOS:** es el navegador que encontraremos configurado por defecto en iPod Touch, iPhone y también en iPad. Las estadísticas lo señalan como el navegador más utilizado dentro del mundo móvil. Este dato es muy interesante, ya que iOS se encuentra por detrás de Android en lo que se refiere a cantidad de usuarios a nivel global.
- **BlackBerry Browser:** es el navegador preinstalado de los *smartphone*s fabricados por RIM. A partir de la versión 6 del sistema, este navegador tuvo una importante renovación en su interior y cambió su motor a Webkit. Anteriormente utilizaba Mango.
- **Internet Explorer Mobile:** es el navegador instalado de manera predeterminada en los dispositivos que cuentan con Windows Phone. Las primeras versiones de Internet Explorer para dispositivos de mano aparecieron en la segunda mitad de la década de los noventa. Pocket Internet Explorer vio la luz en 1996, y fue desarrollado para dispositivos de mano (anteriores a los smartphones) con Windows CE. Internet Explorer Mobile aparece en el 2008 para Windows Mobile, y luego encontramos la versión destinada a Windows Phone.
- **Opera Mobile:** desarrollado por una empresa noruega, que también ofrece versiones para sistemas operativos de escritorio, este navegador es considerado como la principal alternativa que los usuarios eligen en lugar de los navegadores que vienen preinstalados en el sistema.
- **Opera Mini:** es la versión de Opera para dispositivos móviles de menor potencia. Se basa en un motor de *render online*, que procesa y comprime la información para entregarla al usuario. Esto beneficia la performance en *smartphone*s de bajos recursos de hardware, pero también presenta algunas limitaciones en las características disponibles en la navegación.
- **Firefox:** en el año 2010, Mozilla lanzó la primera versión de su navegador para *smartphone*s y *tablet*s compatible con Maemo y un año después llega la versión para Android. Este navegador puede ser instalado por los usuarios en móviles con Android 2.2 o superior (la lista de compatibilidad se encuentra en https://wiki.mozilla.org/Mobile/Platforms/Android). Entre sus características ofrece sincronización con sistemas de escritorio y la posibilidad de incorporar add-ons. El enfoque de Mozilla es que la Web es la plataforma del futuro.
- **webOS Browser / Isis Browser:** los sistemas webOS de HP cuentan con un navegador propio instalado por defecto. Con el cambio de licencia propuesto

del sistema hacia un modelo *open source*, el navegador pasa a llamarse Isis y utiliza QtWebkit (http://trac.webkit.org/wiki/QtWebKit).

- **Amazon Silk:** es el navegador que encontramos en el Kindle Fire de Amazon. Cuenta con características para aceleración en la carga de páginas, aprovechando la robustez de Amazon Web Services (AWS). De esta manera, suma poder a la capacidad de representación del dispositivo, compartiendo la labor y reduciendo la latencia en la carga.

Un dato interesante a considerar es que cuando pensamos en navegadores para dispositivos móviles, debemos tener en cuenta que sus características y funcionalidades pueden ser más limitadas que las de los de sistemas de escritorio y, aunque lleven el mismo nombre que sus "hermanos mayores", en algunos casos, pueden tener un soporte distinto a nuevas tecnologías. Esto quiere decir que si, por ejemplo, sometemos a pruebas de compatibilidad a las diversas versiones de Chrome, tanto en plataformas *desktop* (Windows, Linux y Mac OS X) como en plataformas móviles (Android e iOS), seguramente tendremos resultados con algunas diferencias.

Otro dato que resulta interesante para analizar es que varios de los navegadores mencionados en el listado (Android Browser, Safari para iOS, BlackBerry Browser, webOS Browser y Amazon Silk) guardan un punto en común: emplean como componente principal el motor de Webkit para representar contenidos Web. A partir de 2013 Google anuncia el desarrollo de Blink, un motor basado en WebKit que adoptan los navegadores que utilizan Chromium, como el caso de Google Chrome y Opera (que discontinúa Presto).

Como comentábamos, la lista de navegadores para móviles es realmente muy extensa. Además de los mencionados, podemos destacar también las siguientes alternativas para instalar: Dolphin Browser, Skyfire y Maxthon.

Pantallas

Gracias al éxito de las interfaces táctiles, en el mundo de los móviles la pantalla se ha transformado en protagonista casi exclusivo tanto para visualizar como para interactuar con el contenido.

Son varias las características que definen a las pantallas de los dispositivos móviles. Podremos encontrar diferencias en las tecnologías empleadas para su fabricación y también en cuanto a su calidad y durabilidad.

Entre los dispositivos táctiles, se distinguen modelos que cuentan con pantallas resistivas y otros que incorporan pantallas capacitivas. Las primeras son una

alternativa económica con menor brillo y mayor grosor. Por otra parte, las capacitivas ofrecen una mejor respuesta a las acciones que pueda realizar el usuario, ya que hasta un suave deslizamiento del dedo puede disparar eventos. Son adecuadas para dispositivos con características multitáctiles.

Para nuestra labor como desarrolladores, en lo referente a pantallas, nos enfocaremos especialmente en su tamaño, resolución, *aspect ratio* (relación aspecto) y tipo de interfaz con la que puede interactuar el usuario.

Tamaño de pantalla, resolución y aspect ratio

El tamaño de la pantalla de los teléfonos móviles y *tablets* se mide en pulgadas, de manera diagonal, de igual forma que en los televisores o monitores de computadoras de escritorio o laptops. Aquí deberíamos marcar claramente las diferencias entre *smartphone* y *tablets*:

- **Teléfonos inteligentes:** sus pantallas que pueden ir desde 2,5" en modelos de baja gama hasta las 4,8" en algunos modelos lanzados en el último tiempo.
- ***Tablets*:** sus medidas arrancan, por lo general, en 7" y podemos encontrar modelos que superan las 11".

La resolución de la pantalla no está necesariamente fijada por el tamaño del *display*. En los teléfonos más básicos podemos encontrar desde 128 x 128 px, llegando hasta 2048 x 1536 px o 2560 px x 1600 px en *tablets* que se ubican en la gama High Densitiy y Extra High Density.

La tendencia del mercado actual de teléfonos inteligentes de alta gama tiene una clara inclinación hacia el desarrollo de pantallas cada vez más grandes. Algo que se puede notar claramente en dispositivos como iPhone 5 (1136 x 640 px en 4" de pantalla) o en el Samsung Galaxy S III (1.280 x 720 px en una pantalla de 4,8 pulgadas), solamente por citar algunos ejemplos.

Por otra parte, el *aspect ratio* de las pantallas se refiere a las proporciones que guardan entre su ancho y su alto. En monitores se popularizaron en la década de los noventa, los modelos VGA y SVGA con una relación de 4:3. Hoy en día los dispositivos *wide* han ganado el mercado y pueden utilizar proporciones de 16:9 o 16:10.

En *smartphones* y *tablets* podremos encontrar diferentes relaciones. Por ejemplo, en iPhone (hasta el 3GS) la relación ha sido de 2:3 o 3:2 (según la orientación) con una resolución de 480 x 320 px. Con la llegada de las pantallas retina, en iPhone 4 y 4S, se pasa a 640 x 960 px, pero igualmente se mantiene la proporción. Como se decidió mantener el tamaño de pantalla, la resolución mayor se logra gracias a que se modificó la densidad de píxeles (*pixel density*) que pasó de 163 ppi a 326 ppi.

El iPhone 5 promueve una evolución en el tamaño de pantalla (pasa de 3,5" a 4"), y por consecuencia, la resolución (640 x 1136 px) y en el *aspect ratio* (16:9).

Si observamos en caso del iPad, encontraremos que la resolución de los modelos originales es de 1024 x 768 px en 9,7" y una densidad de píxeles de 132 ppi. En los iPad con pantalla retina encontramos una resolución 2048 x 1536 px, manteniendo, hasta ahora, la relación 4:3 y el tamaño 9,7", pero con una densidad de píxeles mayor: 264 ppi.

En los dispositivos Android, la variedad de pantallas, resolución y demás características es realmente amplia. Para teléfonos las resoluciones más utilizadas son: 240 x 320 px, 320 x 480 px y 480 x 800 px; mientras que en *tablets* encontramos 600 x 1024 px, 720 x 1280 px y 800 x 1280 px, entre otras. Cabe destacar que los modelos de High Densitiy y Extra High Density ya están superando estas resoluciones de pantalla. Por su parte, Microsoft estableció para Windows Phone 7 una resolución de pantalla de 480 x 800 px. Windows Phone 8 además de la mencionada resolución suma soporte para dispositivos con 720 x 1280 px y 768 x 1280 px. En los *smartphones* de BlackBerry tenemos dos grandes grupos: Hight Resolution (480 x 320 px y 480 x 360 px) y Low Resolution (240 x 320 px, 240 x 260 px y 320 x 240 px).

Interfaces y tecnologías

La navegación por cursor permite que los usuarios naveguen por la pantalla moviéndose mediante el teclado físico del dispositivo con una especie de puntero para seleccionar o acceder a elementos de los documentos Web.

Muchos dispositivos modernos nos ofrecen interfaces táctiles, para interactuar con nuestros dedos. También podremos encontrar dispositivos multitáctiles que nos permiten la interacción con varios elementos de la página o aplicación Web al mismo tiempo con nuestros dedos, abriendo la posibilidad del uso de gestos (con los dedos).

El reconocimiento de voz es una opción que comienza a encontrar su lugar a la hora de navegar o realizar búsquedas en Internet. Un ejemplo interesante para destacar en este campo es el sistema SIRI incluido por Apple en iOS a partir del lanzamiento de iPhone 4S. Mediante este software, el usuario puede realizar consultas o pedidos por voz que el dispositivo intentará resolver y brindar una respuesta o resultado a través de los servicios Web vinculados con esta característica.

Las aplicaciones Web: ¿qué es una Web App?

Web App es la manera de llamar habitualmente a una aplicación web, en referencia a su denominación en idioma inglés: Web Application. La importancia de comprender su naturaleza es uno de los ejes fundamentales de este libro, ya que existen diversos enfoques a la hora de definir qué es (y qué no es) una Web App. En principio deberíamos saber que son aplicaciones desarrolladas para funcionar desde un navegador Web, algunas de los cuales pueden trabajar del lado cliente o bien conectarse e interactuar con tecnologías del lado servidor, para intercambiar datos o realizar otras operaciones.

Si observamos un poco la historia de las aplicaciones Web encontraremos que a partir de mediados de la década del noventa, con la aparición del lenguaje JavaScript y ciertas tecnologías que comienzan a integrarse en los navegadores surgen los primeros ejemplos de cómo se puede crear interacción mediante *scripts* desde el lado cliente.

Casi una década después, AJAX ataría todos los cabos, combinando las técnicas existentes, para permitir la creación de aplicaciones Web de la talla de Gmail y Google Maps, entre otras. También abriría las puertas para funciones indispensables para redes sociales como Facebook, Flickr o Twitter.

A la par del nacimiento de JavaScript, desde una empresa llamada Macromedia surge Flash, un producto que luego sería adquirido por Adobe. Esta herramienta permite a diseñadores y desarrolladores crear aplicaciones ricas en contenido multimedia, entre ellas: aplicaciones y juegos o presentaciones de gran impacto visual.

Es importante resaltar que Flash no es un lenguaje, sino que es un software que permite generar archivos empaquetados, que pueden funcionar en el navegador gracias a un *plugin*. El gran éxito de esta plataforma fue respaldado con la gran aceptación de los usuarios, en equipos *desktop*. Pero con la evolución de los equipos móviles, llegada del iPhone mediante, Flash encontró una barrera: algunos dispositivos no incluían el *plugin* por defecto, y en otros no sería posible incorporarlo (como el caso de los dispositivos móviles de Apple). Adobe finalmente decidió discontinuar las versiones de Flash Player para móviles, manteniendo por ahora la línea de reproductores Flash para navegadores de sistemas operativos del mundo *desktop*.

El nuevo actor de esta historia es conocido como HTML5. Sobre sus características y ventajas hablaremos a lo largo del libro, pero por ahora lo importante es saber que es un conjunto de nuevas tecnologías que nos permiten crear aplicaciones que funcionan de manera nativa, tanto en navegadores de escritorio como en móviles sin necesidad de instalar *plugins* o incluir agregados de terceros.

Como podemos darnos cuenta, HTML5 abre una nueva era para las aplicaciones Web con características que van desde las posibilidades multimedia y el 3D hasta el almacenamiento local del lado cliente y la posibilidad de trabajar de manera *offline*.

El W3C ofrece documentación con las características que están en definición para la creación de aplicaciones Web para móviles. El detalle lo podremos encontrar en la siguiente dirección:

http://www.w3.org/wiki/Standards_for_Web_Applications_on_Mobile

Aplicaciones para móviles: ¿nativas, Web o híbridas?

Aquí llegamos a otro de los interrogantes clave al que se enfrentan muchas personas a la hora de crear aplicaciones para móviles: "¿Es conveniente hacer aplicaciones nativas o es una mejor opción crear aplicaciones Web?".

La respuesta a esta pregunta puede variar dependiendo de las necesidades, recursos disponibles, previsión de actualizaciones futuras y perspectiva de escalabilidad del proyecto. En consecuencia, la respuesta llegará luego de realizar un análisis profundo sobre las características y alcances que tendrá el desarrollo que se esté planificando.

Una aplicación nativa cuenta con las ventajas de permitir una adaptación pensada para cada plataforma. Esto también nos da la posibilidad de crear alternativas distintas, si fuera necesario, por ejemplo para *tablets* y *smartphones*, o para las versiones existentes del sistema operativo en cuestión. El acceso a hardware está disponible y nos permite una manera eficiente de interactuar con los recursos del dispositivo.

Para desarrollar de manera nativa deberemos contar con el conocimiento del lenguaje de programación adecuado para cada plataforma. Esto implica que si vamos a crear soluciones para diferentes sistemas, deberemos dominar varios lenguajes.

Si desarrollamos una aplicación Web hospedada en Internet contamos con la ventaja de tener mayor control y facilidades para la actualización sin que el cliente deba realizar ninguna acción. Dispondremos del servidor para realizar estos procesos y es posible cambiar toda la aplicación sin necesidad de modificar nada en el equipo del usuario.

Una novedad que introduce HTML5 a favor de las aplicaciones Web es la posibilidad de que funcionen *offline*, y también de utilizar almacenamiento local. De esta manera, podremos sumar estas características a nuestros proyectos si lo deseamos.

En caso de trabajar con aplicaciones Web pensadas para móviles, algunos sistemas nos permiten crear un acceso a los sitios en la pantalla inicial, con el resto de las aplicaciones. Esta alternativa puede servir tanto para el funcionamiento *online* u

offline de sitios Web. Incluso tendremos la posibilidad de incluir el ícono que deseemos para la pantalla principal, agregar una pantalla de inicio a la aplicación y utilizar el modo *fullscreen*.

Dentro de la variedad de alternativas, también existen las aplicaciones denominadas híbridas. Éstas son aquellas que se escriben con los lenguajes empleados en el mundo Web (HTML, CSS y JavaScript), y que luego se empaquetan como nativas para plataformas móviles. Pueden funcionar a pantalla completa, utilizando un visor incluido en el sistema y sin que se vean las barras y controles del navegador. Otro punto a favor es que se pueden publicar y distribuir desde las tiendas *online*, al igual que un desarrollo realizado en lenguaje nativo del móvil. El usuario las podrá descargar e instalar de la misma manera que cualquier otra aplicación que hayan adquirido en la tienda.

La idea de este modelo de aplicación se apoya en el concepto de escribir el código base una vez, y luego poder emplearlo en diversas plataformas con modificaciones mínimas que ahorran muchas horas de desarrollo, y pueden permitir reducir costos en los proyectos. Este tipo de solución es una buena opción para salvar el dilema de la fragmentación de sistemas operativos y dispositivos existentes en el mercado.

Apache Cordova (originalmente conocido como Phonegap) es una de las opciones que encontramos para realizar el empaquetado de nuestras aplicaciones. Es posible obtener este paquete para realizar compilaciones en línea de comando, agregar *plugins* en nuestros programas de desarrollo (Eclipse o Xcode) o tener la integración del servicio dentro de las opciones que ofrece Adobe Dreamweaver (como Adobe Phonegap).

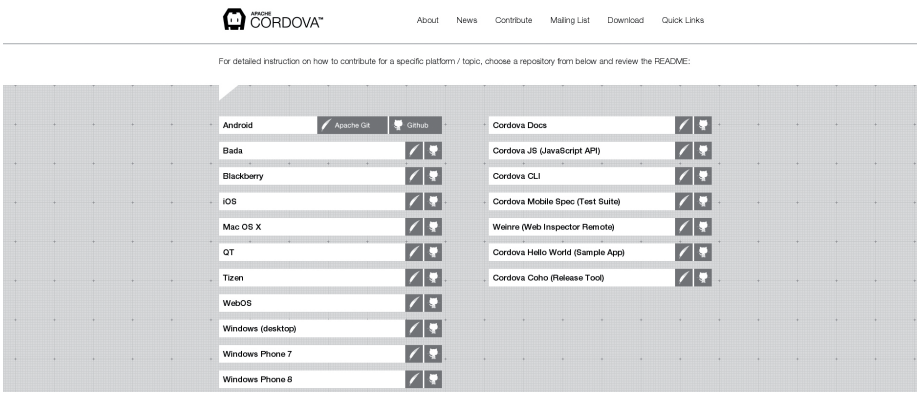

Fig. 1.2. En http://incubator.apache.org/cordova/ encontraremos detalles de Apache Cordova y podremos observar las posibilidades que nos ofrece para diversas plataformas.

Existen otras alternativas para la creación de Web Apps a partir de HTML5 y JavaScript, como el caso de Appcelerator Titanium:

(http://www.appcelerator.com/)

Este producto, a diferencia de Apache Cordova que utiliza un visor del sistema, trabaja sobre el proyecto realizando una "trascodificación" que permite lograr una aplicación nativa para cada plataforma.

Otros *frameworks* que nos ayudan en el desarrollo multiplataforma para dispositivos móviles son:

- Appspresso (http://appspresso.com/)
- Application Craft (http://www.applicationcraft.com/)
- MoSync (http://www.mosync.com/sdk)

La optimización de los recursos es fundamental a la hora de encarar desarrollos para móviles. Es importante buscar siempre la fluidez y buen desempeño en nuestras aplicaciones, contemplando las limitaciones que tienen muchos móviles en cuanto a hardware, y aprovechando sus virtudes, en especial, la portabilidad y las ventajas que brindan las interfaces táctiles.

El proyecto

En el camino del aprendizaje, es muy valioso adquirir conocimientos, asimilarlos, y luego poder aplicarlos en situaciones reales. El marco teórico juega un rol clave en nuestra ruta, pero la hora de la verdad llega cuando ponemos manos a la obra y llevamos a la práctica lo aprendido.

En el libro, mientras profundizamos sobre los conceptos más importantes que nos ayudarán en el desarrollo de soluciones para móviles, también llevaremos adelante un proyecto que nos mostrará la parte práctica de esta labor.

A continuación, analizaremos la idea inicial que llevaremos adelante en el proyecto y los objetivos que buscaremos alcanzar al completar la obra.

Idea y objetivos

Nuestro primer desafío en este proyecto es comenzar a pensar en *mobile*, es decir, distinguir las diferencias que existen entre lo que conocíamos del mundo de soluciones para computadoras de escritorio, y empezar a enfocarnos en dispositivos móviles.

Este primer capítulo, nos ha brindado un pantallazo que nos introduce en un nuevo ecosistema, donde el tamaño no es el único factor importante, sino que es parte de una serie de características que debemos conocer y manejar. Una filosofía diferente, si estábamos acostumbrados al desarrollo en otras plataformas.

A partir del próximo capítulo, analizaremos los aspectos que nos permitirán comenzar con la planificación de nuestra aplicación Web para móviles. Realizaremos el *mockup* o boceto del prototipo de nuestro proyecto, que será nuestra guía inicial para darle vida a la interfaz de nuestra aplicación.

En el tercer capítulo, veremos los lenguajes y tecnologías elegidos para llevar a cabo el desarrollo. Los programas y las librerías que nos acompañarán en este proyecto serán objeto de análisis en el cuarto capítulo. Comenzaremos el maquetado en el quinto capítulo, y nos apoyaremos en las ventajas que nos ofrece jQuery Mobile para crear nuestra estructura y aplicarle efectos. En el capítulo 6, aprenderemos a empaquetar nuestra Web App, para luego, en el capítulo final de esta obra, conocer las opciones de distribución disponibles.

Nuestra aplicación será desarrollada con las técnicas más modernas del mercado. Veremos cómo hacerla funcionar desde un servidor y también generaremos un empaquetado para que pueda ser instalada, igual que una aplicación nativa. Contará con opciones que permitirán ver contenidos de Internet, y también tendrá características que aprovecharán las nuevas opciones de almacenamiento local de HTML5.

En resumen, la aplicación que crearemos será capaz de obtener novedades de una fuente ubicada en Internet, y le permitirá al usuario crear sus propias notas que serán almacenadas localmente en el equipo.

La idea final es aprender a integrar diversas técnicas que se utilizan para la creación de Web Apps, y a su vez, tener la posibilidad de verlas aplicadas de un modo práctico en un mismo proyecto.

Planificación de una aplicación

2

Pensar en móvil

En el capítulo anterior, destacábamos la importancia de comenzar a pensar en móvil para iniciar un viaje que nos permita desarrollar aplicaciones destinadas a estos dispositivos. A fin de enfocarnos en este mundo, antes de poner manos a la obra en el rol de desarrolladores es necesario recorrerlo como usuarios. Puede ser una buena idea intentar despojarnos de lo que sabíamos del mundo Web, ése que solemos navegar desde un equipo de escritorio. Podemos aprender mucho si realizamos una recorrida intuitiva por diferentes sitios desde nuestro móvil. Es recomendable hacer este ejercicio desde un *smartphone* que cuente con un *browser* moderno como el que nos ofrecen las últimas versiones de iOS o Android.

A esta propuesta podríamos sumarle la tarea de realizar el mismo procedimiento desde una *tablet*, para tratar de identificar las diferencias que salten a la vista, respecto a la recorrida realizada desde un *smartphone*. Seguramente notaremos que no es solo una cuestión de tamaño, sino que también la experiencia nos resultará distinta. Pero vayamos por partes para comprender mejor esta situación.

Seguramente comprobaremos al recorrer sitios y al utilizar aplicaciones para *smartphones* y *tablets* que los patrones de diseño son diferentes. En este nuevo mundo es fundamental adaptarse al contexto. Algunos sitios cuentan con versiones especialmente optimizadas para móviles y *tablets*.

Al efectuar diferentes pruebas con sitios y aplicaciones *online*, un aspecto interesante que notaremos es que si cambiamos la orientación del dispositivo, en los sistemas que lo soporten, el navegador se adaptará al giro junto con el contenido.

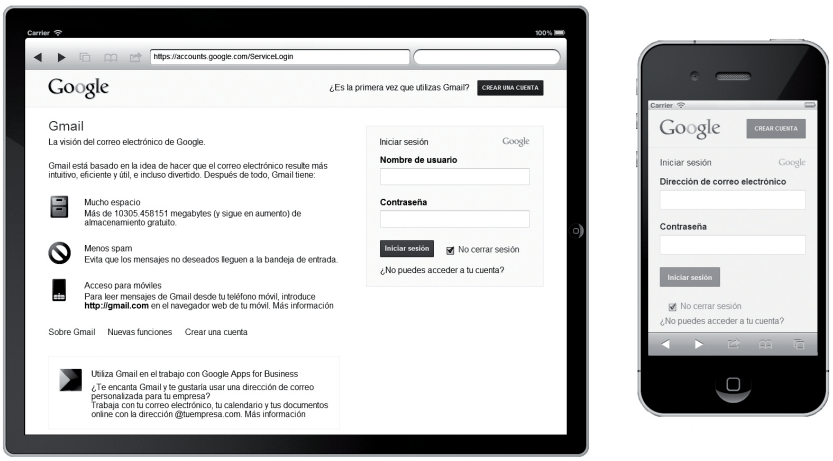

Fig. 2.1. Pantalla de inicio de sesión de Gmail, vista en una *tablet* (en modo vertical) y en un *smartphone* (en modo horizontal).

Otro factor clave al que deberemos prestar mucha atención es al tiempo de carga inicial y a la *performance* general. En lo que se refiere a las aplicaciones móviles, el tiempo de arranque es oro. En caso de que el inicio de la aplicación pueda demorarse, hay que tratar de indicarle al usuario que el proceso está en marcha y que debe esperar. Claro que si lo hacemos esperar demasiado, también podrá pensar que algo no está funcionando bien y cerrará la aplicación.

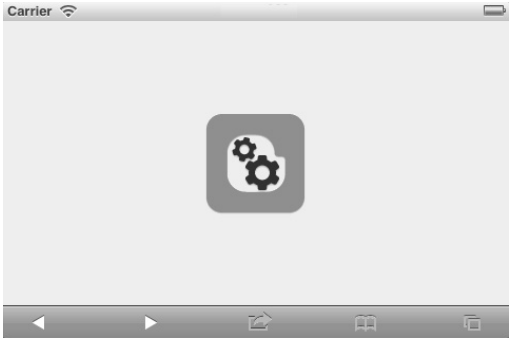

Fig. 2.2. Para mejorar la experiencia de usuario, si un sitio o aplicación demora en la carga, es importante mostrar un indicador sobre esta situación.

Contamos con diversas técnicas para medir la eficacia y los tiempos de respuesta de un sitio Web o de una Web App. Con estos datos en la mano, podremos evaluar si debemos simplificar u optimizar algunos recursos. En el mundo móvil, es fundamental que cada página o pantalla sea liviana, tanto para beneficiar la carga

rápida como también para permitir que el usuario se pueda desplazar con agilidad y no sienta que es “pesada”.

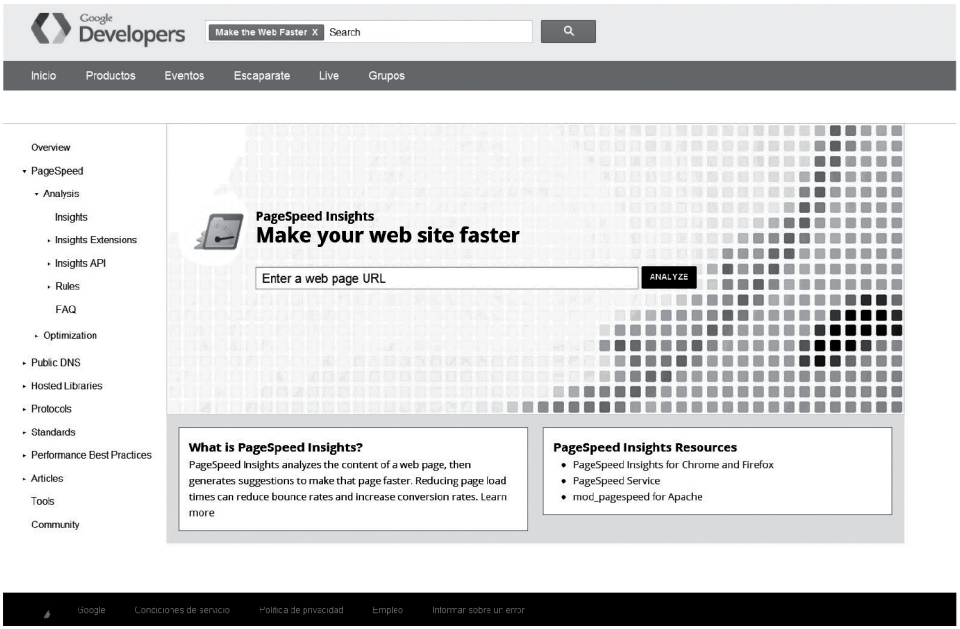

Fig. 2.3. Entre las herramientas en línea que existen para evaluar el tiempo de carga de una página, se destaca *PageSpeed Insight* (https://developers.google.com/speed/pagespeed/insights). Con este test, obtendremos recomendaciones para optimizar la *performance* de nuestros desarrollos.

Arquitectura de la información

Esta disciplina comenzó a mencionarse en el mundo de la informática hacia fines de la década de los cincuenta, pero recién se puso en práctica de manera efectiva a partir de los años setenta. En la actualidad, cumple un rol importante a la hora de planificar grandes sitios de Internet y también de diagramar el flujo de información en aplicaciones que corren en sistemas móviles, entre otros casos de uso.

Al definir cómo se mostrará una página, los aspectos estéticos deben complementarse con la organización eficaz del flujo de información que se necesita ofrecer en ese espacio. ¿A qué apunta esta afirmación? A que no solo debemos buscar el atractivo visual, sino que también debemos lograr ser efectivos y cumplir con la misión de cada pantalla: el usuario debe tener facilidades para poder encontrar lo que vino a buscar. En esta parte es donde nos debemos enfocar en la

arquitectura de la información, la manera en la que estructuraremos el contenido en cada página de nuestro proyecto.

Definir un orden jerárquico adecuado nos ayudará a organizar de una manera eficaz la información en el reducido espacio que tenemos disponible para así proyectar las interfaces en aplicaciones y sitios Web para móviles.

Arquitectura de información en una Web App

Enfocándonos ahora sí en el mundo de las Web Apps, la arquitectura de la información nos permite determinar qué elementos estarán disponibles en cada pantalla, si serán siempre visibles o si estarán, por ejemplo, en menús o submenús. También permite plantearnos si existen elementos innecesarios que podríamos retirar para beneficiar a la interfaz. El objetivo es ser simples y efectivos. Lo que encuentre el usuario al ingresar debe ser claro a primera vista, reduciendo al máximo la curva de aprendizaje que necesitará sortear cada persona para comprender cómo recorrer o utilizar la aplicación.

Ante todo, es necesario fijar jerarquías para organizar los elementos en cada pantalla para comenzar a definir la estructura. Luego, llegará el turno de establecer el lugar que le tocará ocupar a cada elemento. No podemos tirar los dados y ver qué números salen para decidir el orden y lugar que tendrán las cosas. Si bien puede haber cierta libertad para diseñar y crear, también debemos saber que existen algunas reglas y recomendaciones que harán que nuestra interfaz resulte mucho más efectiva. Identificar de antemano las necesidades y expectativas del usuario es fundamental para el proyecto. Ubicar todos los elementos en el lugar adecuado será una labor que tendremos que emprender en cada una de las pantallas de nuestra aplicación.

Lo importante es comprender cómo llevar esto a un terreno práctico. Si nos encontramos frente al desarrollo de una aplicación que nos ha solicitado un cliente, deberemos tener la suficiente destreza para combinar lo que nos ha solicitado con las necesidades de los usuarios finales del producto. En muchos proyectos, éste es uno de los momentos de mayores idas y vueltas, hasta confirmar cómo quedará todo. Es fundamental que el debate se produzca en esta etapa para que la base esté bien definida y acordada, ya que los cambios luego pueden ser un problema. Frases como: “¿Podríamos mover este menú aquí?” o “Necesitamos agregar algunos campos a ese formulario” no deberían aparecer a pocas horas de lanzar un producto.

Aunque pueda resultar redundante decirlo, el flujo de la información debe estar bien planeado desde el principio. Si queremos aplicar estos conceptos en las etapas finales, quizás resultaría mejor empezarlo todo de nuevo. El contenido es el rey, y nosotros debemos aprender a ubicarlo en el lugar que le corresponde cuando realicemos la planificación del proyecto que estamos llevando adelante.

Usabilidad

La usabilidad está en todas partes: en nuestra vida cotidiana, en los productos que utilizamos a diario, y en los dispositivos que llevamos en nuestro bolsillo o bolso de mano. No es solamente un concepto de la Web o del mundo informático. Aunque muchas veces no reparamos en ello, las cosas que usamos a diario pueden llegar a tener muchas horas de planeamiento y estudio previo, antes de haber llegado a la forma definitiva con que los encontramos al utilizarlas.

En la vida real, en el mundo físico por así llamarlo, los elementos tienen un objetivo. Por citar un ejemplo, en una biblioteca colocamos los libros y en el armario guardamos la ropa. No doblamos la ropa para que entre en la biblioteca o apilamos libros creando torres para que se guarden en el armario. Esto no es solamente porque uno sabe que debe ser así, sino que las características que tienen cada uno de estos objetos han sido pensadas para que funcionen de esa forma y sean fáciles de usar por las personas.

Si llevamos esa idea al mundo de la informática, nos daremos cuenta de que existen algunas reglas que, aunque no las hayamos leído en un manual, las comprendemos por intuición. Hay interfaces concebidas con una eficacia y simpleza tan grande que un niño de corta edad las puede comprender, sin necesidad de que se le explique nada, sin siquiera saber leer.

La curva de aprendizaje también cuenta. Aunque a veces la olvidamos, no todas las personas que hoy en día se enfrentan a una computadora o dispositivos electrónicos, son nativos digitales. Esto suma otra barrera que debemos ayudar a vencer a la hora de utilizar un sitio Web o software informático. Por otro lado, es evidente que los niños nacidos en la era digital y "multitáctil" son como esponjas a la hora de absorber conocimientos nuevos, en especial si algo les llama poderosamente la atención como una *tablet* o un *smartphone*.

Este principio debería poder aplicarse en realidad a cualquier persona. Si hace falta agregar un manual de instrucciones para utilizar lo más básico de la interfaz de usuario o explicar cómo movernos en la aplicación, deberíamos reflexionar sobre si la manera en que hemos planificamos el desarrollo ha sido la correcta.

A continuación, veremos un breve listado con algunas recomendaciones para que nuestros proyectos respeten las buenas prácticas de usabilidad:

- Al adaptar para móviles, es muy importante dividir el contenido en porciones más pequeñas de las que estábamos acostumbrados en *desktop*. Debemos pensar en partes más reducidas antes que en bloques extensos.
- El diseño a una sola columna suele ser el indicado sin pensamos en usabilidad de cara a los usuarios de *smartphones*.
- Al crear interfaces para móviles debemos pensar el área donde el usuario puede pulsar de una manera diferente. Por ejemplo, los botones o listas de enlaces no deberían ser menores a 44 px x 44 px.

- Los cuadros de diálogo y ayudas deben ser simples y de fácil compresión. El lenguaje que le exponemos al usuario (visual y escrito) debe ser claro y conciso.
- Los mensajes de error tienen que ser descriptivos, y si es posible, con algún tipo de asistencia para que el usuario pueda salir o resolver el problema.
- En las aplicaciones y sitios para móviles es importante evitar el *scroll* horizontal, salvo que la aplicación lo requiera específicamente.
- En formularios para móviles, es importante que el usuario escriba lo menos posible. En el caso de las búsquedas, podemos ayudarlo, por ejemplo, con opciones tales como las que puede ofrecer un *autosuggest*. En casos de registración tenemos que apuntar a economizar la información solicitada y ayudar al usuario a que deba digitar la menor cantidad de palabras posibles sin dejar de cumplir, por supuesto, con las necesidades del formulario
- Debemos tratar siempre de ayudar al usuario. Por ejemplo, si la persona está por abandonar una ventana sin guardar una configuración o si está por cerrar un documento sin guardarlo, la aparición de un mensaje advirtiendo de esta situación puede evitar más de un dolor de cabeza. Pero recordemos siempre usar el sentido común. Si abusamos de las advertencias en una aplicación, el usuario seguramente terminará por no prestarles atención a ningún aviso.
- En el mundo móvil es importante tener en cuenta que muchos sistemas y dispositivos no cuentan con *plugins* para reproducir contenidos de Flash y Silverlight. Por esta razón, es importante preferir desarrollos Web que utilicen lenguajes nativos del *browser*. HTML5, CSS3 y JavaScript no permiten crear aplicaciones Web ricas de muy buena calidad y compatibilidad con móviles.

Para saber si estamos siguiendo el camino correcto, podremos encontrar diversos test de usabilidad que se pueden proponer a grupos de personas seleccionadas para este fin. La idea es que el grupo elegido se aproxime al perfil de usuario "previsible" para la aplicación que se está desarrollando. En estas pruebas se puede medir la cantidad de errores que cometen los usuarios y el tiempo que les lleva realizar los pasos indicados.

Aquí no es cuestión de juzgar la destreza de las personas que realizan la evaluación, por el contrario, el objetivo es tratar de detectar las dificultades que ofrece nuestro desarrollo para reducirlas, y ofrecer así una mejor experiencia a los usuarios finales que utilizarán la aplicación.

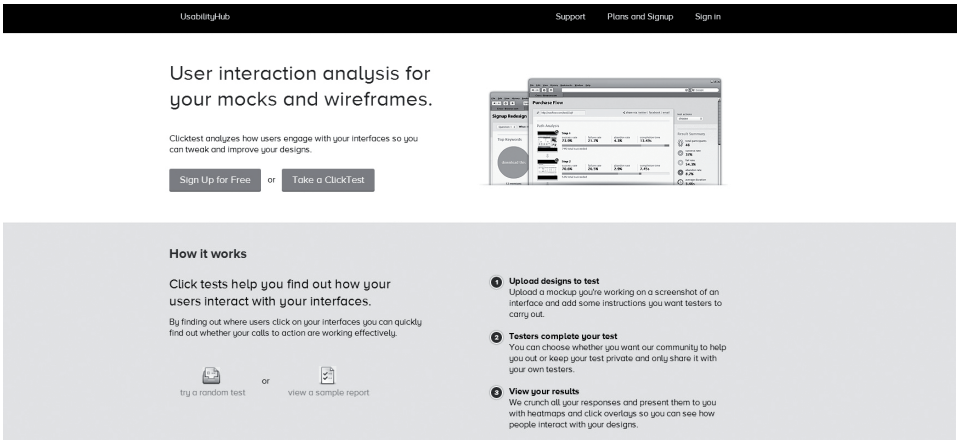

Fig. 2.4. ClickTest es una herramienta *online* que permite analizar de qué manera los usuarios interactúan con la interfaz de un sitio o aplicación Web. Está disponible en: http://theclicktest.com/.

Jakob Nielsen es un referente en el mundo de la usabilidad Web. Nacido en Dinamarca en 1957, ha escrito numerosos libros y artículos sobre esta materia, que nos ayudan a comprender lo que los usuarios buscan y necesitan a la hora de navegar un sitio Web. Si nos interesa profundizar aún más sobre este tema, encontraremos más información sobre Jakob Nielsen en: http://www.useit.com/jakob/.

Accesibilidad

El concepto de accesibilidad apunta a que un objeto o medio pueda ser utilizado sin problemas por una persona, más allá de sus habilidades, aptitudes y contexto. Estas consideraciones se aplican a todos nosotros al margen de nuestra edad, capacidades de aprendizaje, y dispositivo desde el que accedemos, que podrá ser más chico, más grande, con pantalla táctil, teclado físico o cualquier otra característica.

Gracias al esfuerzo de muchas personas, en el mundo informático, y particularmente en Internet, desde hace tiempo se está hablando de la importancia de la accesibilidad. De cierta manera, el concepto se ha profundizado por la expansión y las características de nuevos dispositivos, que han logrado destacar aún más la importancia que tiene todo lo relacionado con accesibilidad en el mundo actual.

La palabra accesibilidad está íntimamente ligada al término inclusión. Existen millones de personas en el mundo que experimentan alguna dificultad física, por ejemplo, auditiva o visual. Ellos también son potenciales usuarios de nuestros

desarrollos. Por eso muy importante considerar a este público, y que nuestro contenido pueda ser accesible y cómodo también para ellos.

En los últimos años, se han creado normas que favorecen la inclusión de todas las personas mediante el correcto uso de las reglas de accesibilidad en software y sitios Web. Es posible encontrar legislación internacional (por ejemplo, en la Unión Europea) y también local en varios países de Latinoamérica y en España.

Si nos trasladamos por un momento al mundo móvil, es fundamental conceptualizar la diversidad de posibilidades que podemos encontrar. Cuando realizamos nuestros desarrollos desde un cómodo sillón, con una computadora veloz, un monitor Full HD de más de veinte pulgadas, altavoces 5.1 y una conexión a Internet de 5 Mbps, deberíamos también tener presente que no todos los usuarios accederán a nuestra aplicación desde un contexto similar, y eso podría implicar un cambio en la manera de plantear la aplicación.

Veamos el caso contrario: una persona viaja de pie en un transporte público, con ruido ambiente, y necesita acceder a una información rápidamente desde su *smartphone* a través de una red 3G. Este escenario, que resulta igual de válido que el descripto anteriormente, nos demuestra que las posibilidades de acceso a un contenido pueden ser muy variadas y que son muchos los factores que inciden.

Si pensamos detenidamente, es probable que el último ejemplo sea bastante habitual para usuarios de dispositivos móviles. Y dada la situación planteada, resulta muy probable que la persona de nuestro ejemplo no pueda tener la atención total sobre lo que está buscando en ese momento. Nuestro desafío en estos casos será que el usuario pueda encontrar lo que desea en la menor cantidad de pasos posibles. Sin distracciones innecesarias de parte de nuestra aplicación y con mucha claridad en la señalización de opciones y/o pasos a efectuar.

La legibilidad en los textos y la claridad en las consignas son factores clave. Resulta también importante que los elementos de nuestra interfaz de usuario sean fácilmente reconocibles. Por ejemplo, los botones y los enlaces deben ser identificables a primera vista. Las zonas que permiten interacción pueden estar señalizadas o destacadas de alguna forma en nuestro diseño, el usuario no debería estar adivinando dónde puede realizar un clic o *touch*. Esto debe ser claro para todos.

Fig. 2.5. La experiencia de usuario es muy importante. Apple, por ejemplo, ofrece varias guías destinadas a ayudar a los desarrolladores en esta tarea. Las encontraremos en: https://developer.apple.com/.

Accesibilidad en el mundo móvil

Comprendido el concepto de accesibilidad, en este paso nos enfocaremos en lo que ocurre en el mundo de los móviles y las *tablets*. La pregunta que nos haremos en este caso es: ¿cuáles son las buenas prácticas que debemos seguir para mejorar la accesibilidad de un sitio Web o una aplicación creada para ser utilizada en dispositivos móviles?

En primer lugar, debemos tener siempre presente que llevar un sitio o una aplicación al formato de un *smartphone* o de una *tablet* no significa simplemente tratar de achicar y concentrar todo para que entre en un espacio más pequeño. Por el contrario, debemos fijar en nuestra mente que el objetivo es que ese contenido se adapte de la mejor manera posible al medio, y que le permita a los usuarios tener una experiencia adecuada y agradable.

Los espacios son los que permiten darle un poco de aire a los elementos que conforman cada pantalla. Aunque muchas veces son utilizados como "variables de ajuste" cuando es necesario agregar algo; pero, en realidad son muy importantes. Hay que considerarlos de la misma manera que a otras características de las pantallas. Si lo deseamos, podemos apoyarnos en la ayuda que nos brindan las grillas y guías para definir con precisión los espacios en nuestros prototipos.

Hay un gran número de recomendaciones valiosas que nos permiten lograr aplicaciones accesibles. Para completar este apartado, veremos una lista de tareas que nos ayudará a verificar que nuestro desarrollo respete las reglas de accesibilidad:

- No todas las interfaces de dispositivos móviles son táctiles, ni todas las interfaces táctiles ofrecen el mismo nivel de precisión para los usuarios.

Debemos evitar que los elementos que tienen acciones programadas, al ser pulsados o seleccionados, se encuentren solapados o demasiados cerca. De esta manera, ayudaremos a evitar que puedan accionarse de manera involuntaria por el usuario.

- Es vital pensar en *layouts* simples, que puedan ser comprendidos de un vistazo. Una de nuestras misiones consiste en evitar sobrecargar el poco espacio con el que contamos en las pantallas de los móviles. Los íconos minimalistas suelen ser muy efectivos en el mundo móvil, pero ante todo debemos tener presente las recomendaciones de accesibilidad para imágenes, en estos casos.
- Cuando se crean contenidos basados en HTML, mirar el contenido de una página "desnuda" de CSS y JavaScript nos puede mostrar si está correctamente estructurada. Existen opciones en los navegadores para efectuar este tipo de pruebas. En algunos casos se puede optar por agregar un *plugin*.
- Reducir la cantidad de pasos al máximo para cada operación. Como hemos visto anteriormente, el usuario puede encontrarse en un contexto que resulte muy poco cómodo para pasar por varias pantallas hasta llegar a su objetivo. Todo debe ser liviano y sin muchos rodeos en el mundo móvil.
- La correcta combinación de colores es fundamental. Tengamos en cuenta que aproximadamente un diez por ciento de los usuarios tiene dificultadas para distinguir entre algunos colores. Este problema suele producirse especialmente con el rojo y el verde.
- Cuando pensemos en el tamaño del texto, no deberíamos utilizar fuentes menores a 14/16 px. Las tipografías demasiado pequeñas dificultan la lectura. Ser minimalistas no implica usar fuentes tan pequeñas que se pierden en la pantalla.
- Excepto en el caso de íconos, logotipos y carteles especiales, el texto no debería ubicarse en imágenes u otros formatos que no permitan accesibilidad. El atributo `alt`, texto alternativo, es obligatorio para la etiqueta de imagen `<img>`.
- El uso de recursos multimedia potencia el nivel de nuestro sitio o aplicación. Desafortunadamente no todos esos contenidos pueden verse en la totalidad de los dispositivos. Debemos plantear soluciones en nuestro desarrollo para aquellos que no puedan mostrar dicho material.
- Cualquier señal sonora que resulte fundamental para la aplicación, también debería tener un indicador visual. Esto puede aplicarse tanto para los alertas que emita como también para los mensajes que brinden información al usuario.
- En los videos, el uso de los subtítulos es un factor esencial como opción de traducción de contenidos y como posibilidad de inclusión de personas con dificultades auditivas.

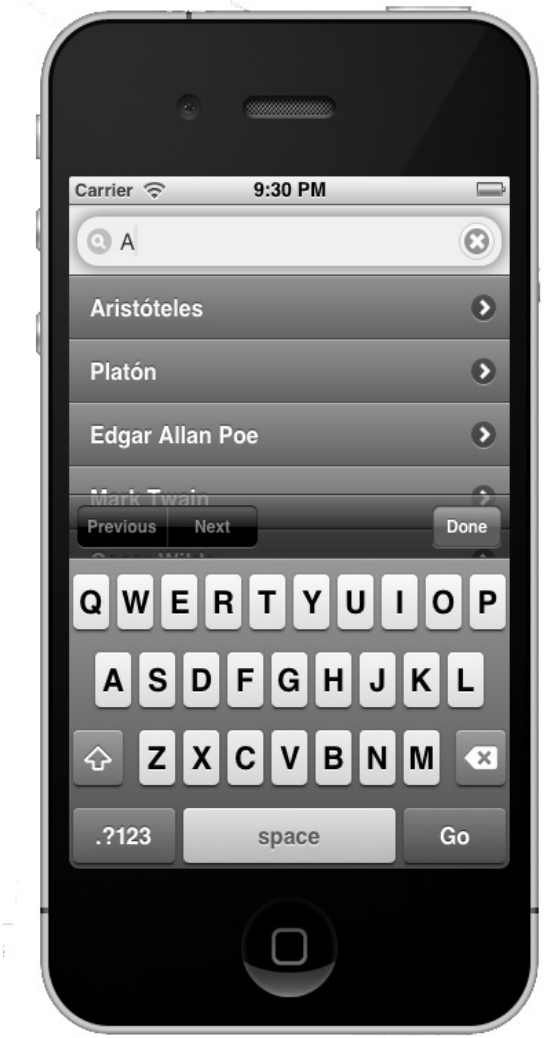

Fig. 2.6. El teclado virtual de los móviles es útil para escribir textos cortos, pero puede resultar tedioso para algunos usuarios si los obligamos a tener que ingresar muchos datos.

Ponernos en el lugar del otro, en este caso, de la persona que utilizará nuestra aplicación, puede ayudarnos a comprender las dificultades que puede enfrentar al intentar usarla. Para comenzar a innovar, primero debemos comprender cómo funcionan las cosas actualmente. Aprender sobre su naturaleza, analizar sus pros y sus contras. Luego de esto, con una buena cuota de creatividad de nuestra parte,

estaremos listos para comenzar a cambiar lo que a los usuarios no les resulta cómodo o fácil de usar.

Fig. 2.7. El uso de traductores *online*, los lectores de texto en voz alta o el caso de ingreso de texto mediante micrófono, son muy buenos ejemplos de la importancia de generar contenidos accesibles que facilitan su uso para todos los usuarios.

El concepto de navegabilidad aplicado a una Web App

Comprender la perspectiva del usuario es un factor fundamental a la hora de crear un sistema exitoso. Siempre debemos suministrar la información que necesita para que pueda ubicarse, utilizarlo y/o recorrerlo con la menor complejidad posible. Convertirse en un callejón sin salida es lo peor que le puede pasar a una página que debería estar vinculada con otras que conforman un sitio Web. El mismo rótulo se le podría aplicar a una pantalla de una aplicación que no le ofrece al usuario la posibilidad de regresar o pasar a otra pantalla, una vez terminado el proceso que fue a realizar allí.

¿Cómo evitamos esto? Primero, ofreciendo siempre al usuario una referencia clara para que sepa en qué lugar está. Por ejemplo: el nombre de la página o la sección en la que se encuentra. En segundo término, debemos asegurarnos que tenga un lugar en el cuál continuar su estadía o bien (si fuera necesario, porque ya completó todos los pasos) la opción de cerrar la sesión o regresar al inicio.

Informarle al usuario el estado del sistema también es fundamental. Y no solo hablamos de cuando se presenta una falla o un error, sino también de cuando se está realizando una operación que pueda demorar más de lo esperado.

Un paso fundamental en nuestro proyecto es la creación de un mapa de sitio (Sitemap) o, en nuestro caso, un mapa de la aplicación con la conexión de las pantallas. Allí podremos definir una estructura general, especificar cómo se conectan las pantallas, y, también nos ayudará a detectar las inconsistencias que

hubiera. Tal vez podemos encontrar que nos falta alguna conexión o que debemos agregar alguna pantalla que actúe como nexo a fin de facilitar la navegación natural que pueda necesitar el usuario.

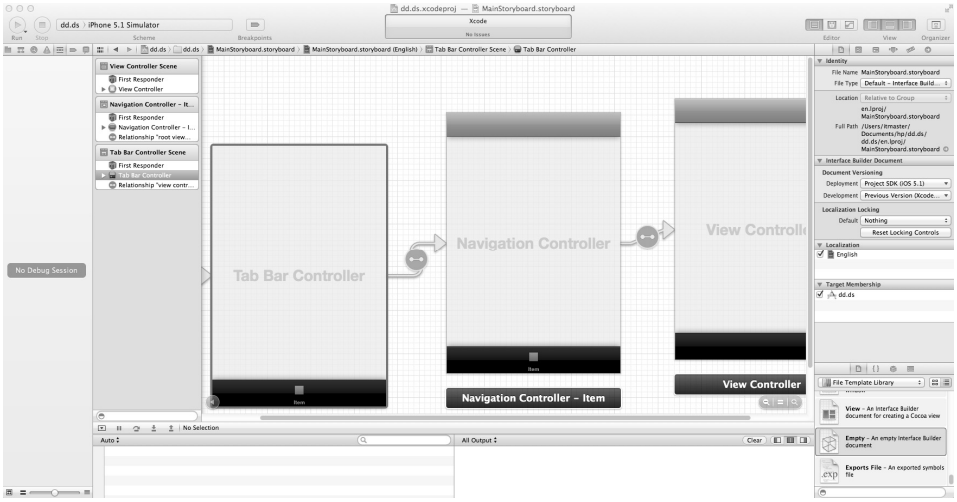

Fig. 2.8. En Xcode, herramienta para el desarrollo de aplicaciones nativa de iOS, se ha incorporado el concepto de *Storyboard* a la hora de crear y conectar las pantallas que conformarán la app.

Cómo orientar al usuario

Como ya hemos mencionado, el usuario en todo momento debe saber en qué lugar está parado. Incluso, de ser posible, deberíamos brindarle la jerarquía del nivel en el que se encuentra para que pueda desplazarse. Es un elemento central que pueda volver al lugar del cual viene o a la pantalla de inicio. El usuario siempre debe tener opciones para continuar navegando.

Para quienes están acostumbrados a los navegadores de escritorio, las flechas para ir hacia atrás o hacia adelante en el historial son ya habituales. Estas opciones también existen en los navegadores móviles. En una aplicación, esto por lo general no es necesario, aunque puede igualmente ser interesante para los efectos del desarrollo contar con una API de historial como la que nos brinda HTML5. Pero ese es un tema que analizaremos más adelante en el libro.

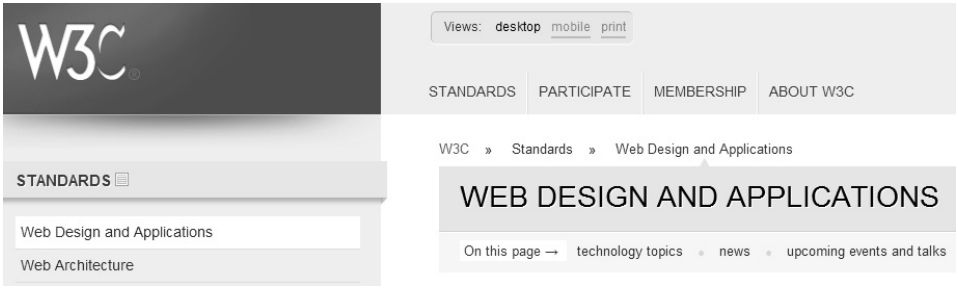

Fig. 2.9. Incluir una navegación con *breadcomb* (migas de pan) ayuda a los usuarios a ubicarse en el lugar donde están. El ejemplo del W3C citado en esta figura ubica a los usuarios en la sección: W3C » Standards » *Web Design and Applications*.

Mejora progresiva

La mejora progresiva (*progressive enhacement*) busca unir varios de los conceptos que analizamos en este capítulo (accesibilidad, usabilidad y arquitectura de la información) sumados a otros que posibilitan crear una solución que pueda ser vista por usuarios en diferentes contextos, es decir, dispositivos, sistemas y/o velocidades de conexión. La idea es partir de un modelo básico sobre el cual trabajaremos hasta mejorarlo. De esta manera comenzamos a pensar las opciones que nos ofrezcan mayor compatibilidad para subir el nivel desde nuestro planteo inicial.

Entre las grandes ventajas de optar por esta metodología en el desarrollo Web, se destacan los beneficios en lo que se refiere a SEO (*Search Engine Optimization*), y la posibilidad de lograr un contenido accesible. Esto nos permite mantener diferentes partes separadas y trabajarlas de una manera eficiente y modular.

La mejor forma de encarar esta técnica es trabajar en capas, en las cuales añadiremos las características que formarán parte de la aplicación como, por ejemplo, en marcado semántica, los estilos y luego las características de programación del lado cliente. Nuestra recomendación es tratar de que las capas puedan manejarse por separado y evitar que el código de programación, la llamada a eventos y los estilos se mezclen con el marcado que define la estructura semántica de una página.

Si lo llevamos a nuestro universo del diseño y desarrollo Web, podríamos decir entonces que las etiquetas HTML nos proporcionan la estructura; las propiedades CSS, la representación; y el código JavaScript nos provee de comportamientos, interacciones y efectos adicionales. Recordemos que JavaScript en su esencia se basa en eventos.

Vale la pena aclarar que la mejora progresiva no apunta a agregar elementos sobre las pantallas definidas previamente bajo pretextos como: “Queda lugar, podríamos

añadir…" o "Para ver qué tal queda, movemos ese elemento a la izquierda…". Por el contrario, la mejora progresiva es una manera de enfocar nuestro desarrollo para fortalecer lo que se define en cada etapa y estructurar el proyecto.

El método inverso se conoce como degradación agraciada (*graceful degradation*). Aquí el planteo arranca desde la opción más avanzada y menos compatible para luego realizar este proceso. Es decir, optar por comenzar utilizando todas las tecnologías modernas, y luego trabajar para buscar el equilibrio de compatibilidad con navegadores o sistemas más limitados.

Los patrones de diseño nos ayudan a resolver situaciones del diseño de una interfaz de manera efectiva sobre la base de experiencias aplicadas con anterioridad. Nos permiten crear estructuras, relaciones y aprovechar la reutilización de elementos, mediante la estandarización y el uso de un lenguaje común. Los patrones de diseño se introdujeron en el mundo informático en la década de los noventa.

El diseño adaptado al medio

En el diseño Web Adaptivo (*Responsive Web Design*), confluyen diversas técnicas que permiten a los diseñadores Web crear estructuras que se ajustarán a diferentes sistemas, plataformas y resoluciones de pantalla. Como hemos señalado anteriormente, la diversidad en resoluciones y tamaños de pantalla nos sitúa frente al desafío de pensar en un diseño que se pueda adaptar al dispositivo del usuario.

Dentro de las innovaciones que llegan de la mano de HTML5 y CSS3 se destaca Media Queries, una característica que nos permite crear reglas condicionales dependiendo de algunas características de la pantalla, como su resolución u orientación del dispositivo.

El concepto más significativo que debemos tener en claro en esta etapa es que si vamos a crear una solución para diferentes medios, debemos olvidarnos del *pixel perfect* y comenzar a pensar en uno de tipo "adaptivo". Más adelante profundizaremos sobre cómo podemos aplicar el *Responsive Web Design* a nuestro proyecto para que pueda verse de manera correcta en los diferentes medios y dispositivos que utilizan habitualmente los usuarios.

La estructura básica inicial

Existen opciones variadas para comenzar con las primeras etapas de trabajo: flujo de la aplicación, estructura de alambre, boceto, etcétera. Si lo deseamos, podemos trabajar con las aplicaciones de diseño con las que estamos habituados o bien optar por algunas alternativas específicas para crear este tipo de prototipos.

Si buscamos armar un flujo de información con un muy buen aspecto visual para presentar al cliente, podemos revisar lo que nos ofrece **Lovely Charts** (http://www.lovelycharts.com). Esta alternativa cuenta con versiones para Windows, Mac OSX, Linux, iPad, y también posee una versión *online*. Esta herramienta brinda diferentes tipos de licencia de pago, pero también nos permite descargar una versión de prueba para evaluar sus beneficios. Además de la facilidad de uso que nos brinda la interfaz, y la buena calidad gráfica, esta opción se destaca por las posibilidades de compartir y exportar nuestros trabajos a otros programas y/o plataformas.

Balsamiq Mockup es una alternativa muy atractiva que nos facilitará el trabajo de crear Mockups. Este producto está disponible en versiones para Windows, Mac OSX y Linux. Es un software pago que cuenta con versiones para diferentes tipos de usuarios y volumen, y también ofrece un trial por tiempo limitado.

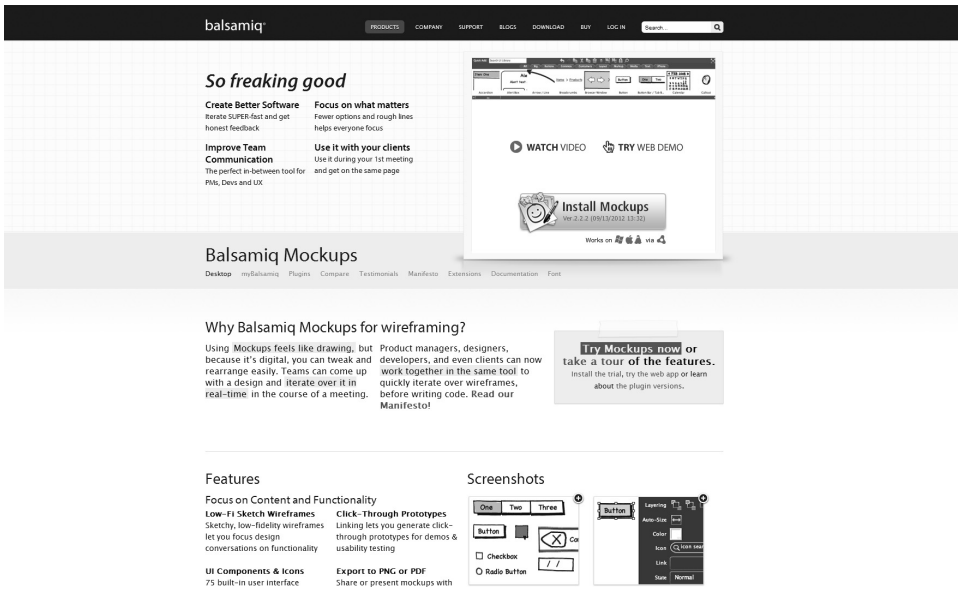

Fig. 2.10. El sitio de Balsamiq Mockup es: http://www.balsamiq.com/products/mockups.

Lo que destaca del Balsamiq Mockup es su facilidad de uso y las opciones que nos brinda para ayudar a distribuir elementos que están predefinidos dentro de la aplicación. Estas posibilidades comprenden a los mensajes emergentes, las barras de progreso o las ventanas modales, solo por citar algunos ejemplos.

Las cuadrículas y las líneas guías resultan de gran utilidad para realizar este proceso, ya que podemos agrupar elementos, moverlos y organizarlos con gran facilidad. La interfaz nos permite editar y personalizar los textos a nuestro gusto. Las

ayudas visuales que nos ofrece también son muy valiosas para simplificar nuestra tarea, en especial, cuando apenas nos estamos familiarizando con la herramienta.

Axure RP es otra aplicación muy potente que nos permite crear prototipos de nuestros proyectos. Este producto es pago y se encuentra disponible en versiones para sistemas Windows y Mac OSX. También ofrece una versión *trial* para probarlo.

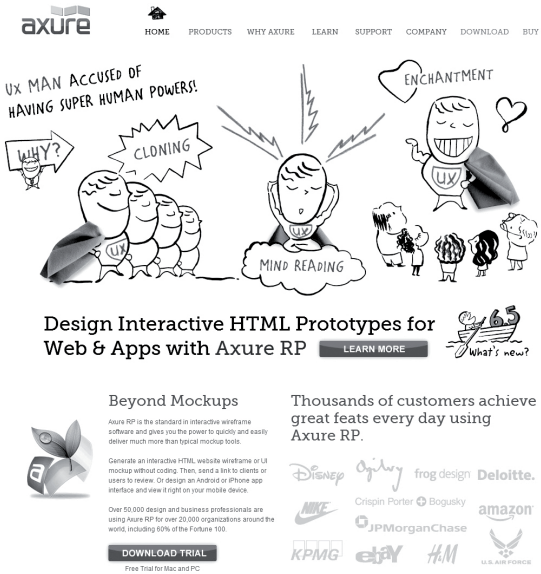

Fig. 2.11. Axure RP es utilizado por numerosas empresas para crear sus *mockups* . Este producto se encuentra disponible en: http://www.axure.com/.

¿Cuáles son las ventajas de Axure? Es una herramienta muy útil para pensar nuestro proyecto como un sistema, y definir el flujo de información y el detalle de las páginas, que serán las pantallas que conformarán nuestra aplicación. A esto se le suma el muy buen nivel gráfico que nos posibilita crear *wireframes* (prototipos de alambre), y también modelos de gran calidad visual para presentar o hacer demostraciones.

Dentro de los aspectos sustanciales de Axure, destaca el ser una excelente opción para trabajar en equipo. Podremos exportar nuestro modelo para que el cliente u otro miembro del equipo de trabajo tengan la posibilidad de verlo. En este rubro tendremos gran flexibilidad de posibilidades, incluyendo opciones interactivas que emularán el funcionamiento de la aplicación.

Planificación del flujo

Debemos intentar tener siempre en cuenta que para llevar un proyecto a buen puerto es clave establecer objetivos claros para así ser capaces de medir su éxito, y de construir parámetros en el futuro. Por ejemplo, se deben estimar tiempos, estipular previamente los recursos necesarios, definir la cantidad de personas involucradas y sus roles, entre otros factores.

Ahora pondremos manos a la obra con la aplicación que planteamos en el final del capítulo anterior. En este paso definiremos el flujo que tendrá el sistema. La idea aquí no es entrar aún en el diseño, sino pensar en qué pantallas necesitaremos crear y cómo se conectarán entre sí. Este diagrama nos permitirá describir de una manera gráfica cómo estará dispuesto el flujo y cómo será la interacción entre las partes que componen la aplicación que estamos planificando.

Este es el momento de llevar lo que está en nuestra mente al papel o al soporte elegido para volcar nuestras ideas. Sin entrar en detalles "finos" de diseño, sí debemos trazar las bases de usabilidad, funcionalidad y navegabilidad. Esta pieza sirve para permitir que los diversas partes puedan sumar sus ideas y las lleven a un punto en concreto.

A la hora de hacer bocetos es clave limitar la cantidad de propuestas. De lo contrario, se puede demorar o estancar el proyecto, aumentando innecesariamente las horas de trabajo invertidas para pasar de etapa.

Diagrama de flujo de la aplicación

Comenzaremos ahora con la diagramación de la aplicación. Luego pasaremos a idear las pantallas que lo componen. Para esto, en nuestro ejemplo utilizaremos Axure RP, aunque podríamos trabajar con cualquiera de las herramientas mencionadas anteriormente, o con otras con las que ya tengamos experiencia y nos resulten útiles para esta finalidad.

Axure nos permite crear diferentes tipos de esquemas, y cuenta con librerías propicias para cada una de las opciones. Además podremos añadir elementos a nuestro gusto.

Empezaremos por crear un documento nuevo en Axure. En el panel de la izquierda, buscamos en la sección **Sitemap** y hacemos un clic derecho sobre **Home**. En el menú desplegable, escogemos dentro de **Diagram Type** la opción **Flow**. Para tener disponibles solo los elementos relacionados con el diagrama de flujo, vamos a la solapa **Widgets** y elegimos **Flow**.

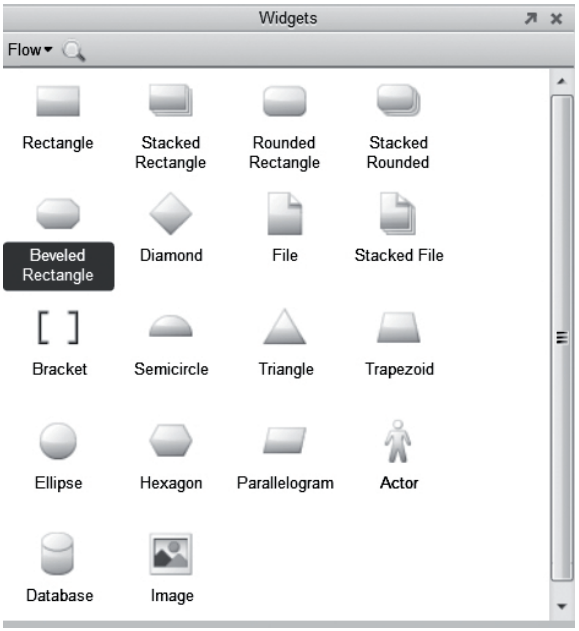

Fig. 2.12. Al seleccionar la opción de *Flow* en el panel de Widgets se encontrarán los símbolos que permitirán definir la lógica del flujo de un sistema.

Para colocar un elemento en el escenario, simplemente, lo arrastramos con el mouse y lo posicionamos en el lugar que deseamos. Las grillas nos ayudarán para ubicar el contenido y la alineación, cuando tengamos varios elementos. Podremos agregar o cambiar el texto del elemento, modificar la tipografía, la alineación del texto, el tamaño, el color, y también añadir un hipervínculo si lo creemos conveniente.

Ahora es el momento de comenzar a crear el mapa. Si repasamos el final del capítulo anterior, podremos ver que quedaron planteadas algunas de las características que tendrá nuestro proyecto: será una app que mostrará noticias a los usuarios y también les permitirá guardar notas personales. Para ampliar la idea original, ahora decidiremos algunos aspectos que definen el enfoque del proyecto

El flujo partirá desde una pantalla de inicio que permitirá al usuario acceder a las diferentes opciones y secciones disponibles. También encontrará las posibilidades de escribir sus notas personales, leer las notas que haya guardado con anterioridad, configurar la aplicación y acceder a las opciones de redes sociales.

Para conectar las cajas, podremos usar la flecha de conexión, ubicada en el menú superior o presionar la tecla **F11** para que habilite esta opción. Luego, generamos la conexión directamente sobre el escenario, vinculando las pantallas.

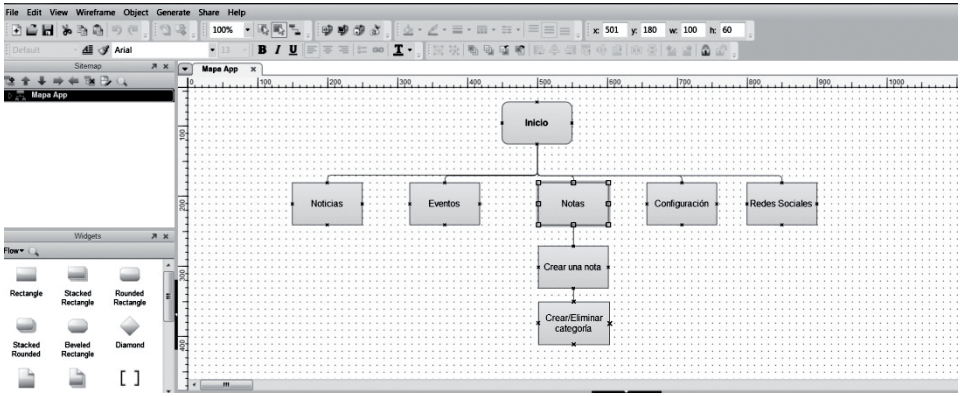

Fig. 2.13. El diagrama de flujo de nuestra aplicación en esta etapa del desarrollo.

Cuando creamos una sección, debemos identificarla de una forma clara y uniforme a lo largo de la aplicación, ya que el usuario debe reconocerla siempre con facilidad.

Una ventana de ayuda es una posibilidad que se puede agregar a las aplicaciones que tienen ciertos procesos que consideramos que pueden ser complejos. De cualquier manera, deberíamos asegurarnos que la aplicación por sí sola pueda ser navegada y utilizada por el usuario, y que la ayuda sea una opción y no un requerimiento si nuestra app no justifica una curva de aprendizaje importante.

Cuando los usuarios ingresan por primera vez a un sitio o a una aplicación, por lo general, tratarán de comprenderla sin recurrir a ayudas, usando los conocimientos que han adquirido anteriormente al usar sistemas similares. Esto debemos recordarlo a la hora de crear una app, ya que si proponemos un cambio muy profundo en las estructuras, puede ocurrir que los usuarios se sientan perdidos o no comprendan qué deben hacer. Ese es el peor de los escenarios, y podría atentar contra el éxito de nuestro proyecto.

Vale la pena mencionar que Axure nos permite trabajar con ***guides*** (guías de página o globales) y ***grids*** (grillas ajustables que podremos mostrar u ocultar cuando deseemos) para ubicar los elementos de una manera simple y natural en nuestro esquema. Estas opciones facilitan nuestro trabajo al permitir que los objetos, imágenes y líneas ubicadas en cada página se ajusten con facilidad.

Las pantallas de nuestra aplicación

Ya definido el diagrama de flujo que tendrá nuestra aplicación, ahora podremos dedicarnos a producir los prototipos de las pantallas que componen el desarrollo que estamos planificando.

Con esta herramienta no podremos simular los giros o los efectos del diseño adaptado al medio. Esto quiere decir que no tendremos la posibilidad de trabajar con porcentajes en las medidas que apliquemos. Si deseamos simular diferentes resoluciones de pantalla, tendríamos que construir un prototipo para cada una de la misma manera que lo haríamos si partiéramos de una maqueta en productos como Adobe Photoshop o Adobe Illustrator.

En el panel de la izquierda de Axure, encontraremos el apartado **Sitemap** donde veremos que ya está creado el diagrama de nuestra app, y debajo, por defecto, encontraremos tres páginas. Allí tendremos la posibilidad de agregar más, crear páginas hijas o renombrarlas. En primera instancia, haremos clic con el botón derecho del mouse, y vamos a renombrar la primera a **Inicio**. Con un doble clic del botón principal del mouse, se abrirá dicha página en el modo *wireframe*. Ahora ya tenemos una nueva pestaña donde podremos crear el prototipo de la pantalla inicial de nuestra aplicación.

Un aspecto que vale la pena subrayar de Axure es que cuenta con muy buenas opciones de expansión, con librerías y otros agregados que se obtienen en Internet. En http://www.axure.com/download-widget-libraries podremos acceder a diferentes tipos de librerías. Algunas de ellas están pensadas especialmente para realizar prototipo de interfaces, que serán utilizadas en aplicaciones para dispositivos móviles como: iOS Wireframe Library, iPad Widget Library, iPhone UI Elements, Windows Phone 7 Library o Touch Screen Hand Gestures. También tenemos algunas enfocadas en facilitar la realización de cualquier tipo de *mockups*, como el caso de Sketchy Widget Library.

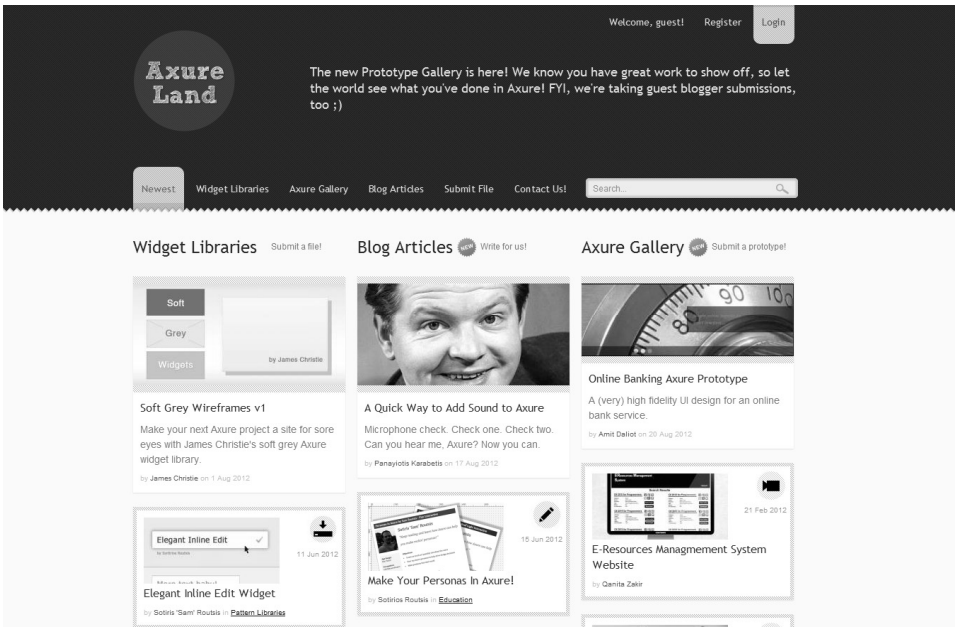

Fig. 2.14. Axure Land es un muy buen lugar para conseguir recursos adicionales para Axure. Ingresando en http://axureland.com se puede acceder a librerías de *widgets*, galerías y también a artículos relacionados con esta herramienta. Dentro de estas opciones se encuentra la librería de jQuery Mobile.

Las librerías de Axure que descargamos de Internet pueden contener un único archivo de extensión **.rplib.** También podremos encontrar algunos sets de recursos y ejemplos que se encuentran disponibles en un archivo comprimido (por ejemplo, un **.zip**).

Para comenzar a utilizarlas, las librerías de Axure deben copiarse al directorio **My Axure RP Libraries** dentro de **Mis Documentos** en el equipo local.

Axure es una herramienta que se destaca por su versatilidad. Entre sus características encontramos el permitir la creación de estilos que luego podremos aplicar a diferentes elementos. Nos brinda la posibilidad de definir opciones que van desde la alineación hasta efectos de *sketch* o bocetado.

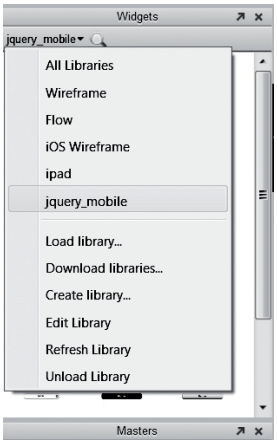

Fig. 2.15. Una vez copiadas las librerías, estarán disponibles dentro de Axure desde el desplegable del Panel de Widgets. Es posible mostrar solo alguna librería en particular o recurrir a la opción *All Libraries* para que estén todas juntas.

Para optimizar el proceso de creación de prototipos, Axure nos ofrece la posibilidad de crear **Masters**. Con esta opción, ubicada en la parte inferior de la barra lateral izquierda, podremos crear le cabecera y el pie de la aplicación, y luego aplicarlos en el resto de las pantallas. En la solapa de **Masters**, pulsamos sobre el segundo ícono que representa la opción **Add Master**, y lo llamaremos **Header**. El encabezado tendrá el botón para regreso a la *home* sobre el lado izquierdo y las opciones de configuración al lado derecho. En el centro ubicaremos el logo o nombre de la aplicación que estamos creando.

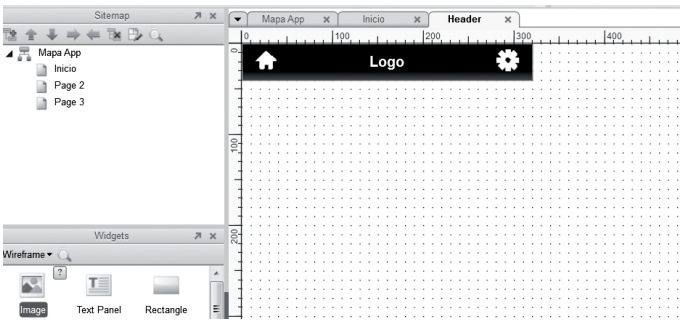

Fig. 2.16. Axure permite incorporar imágenes desde Wireframe (en el panel de Widgets).

A continuación, podremos avanzar sobre el Master que conformará el pie de la aplicación. Lo llamaremos **Footer** y, en él, incorporaremos los accesos a las secciones de Eventos, Noticias, Notas y a Redes Sociales.

Definidos los Masters, podremos comenzar a crear la pantalla inicial de la aplicación. Allí ubicaremos la última noticia y el último evento publicado. Debajo, dispondremos un espacio para que se puedan mostrar alertas o novedades publicadas en redes sociales.

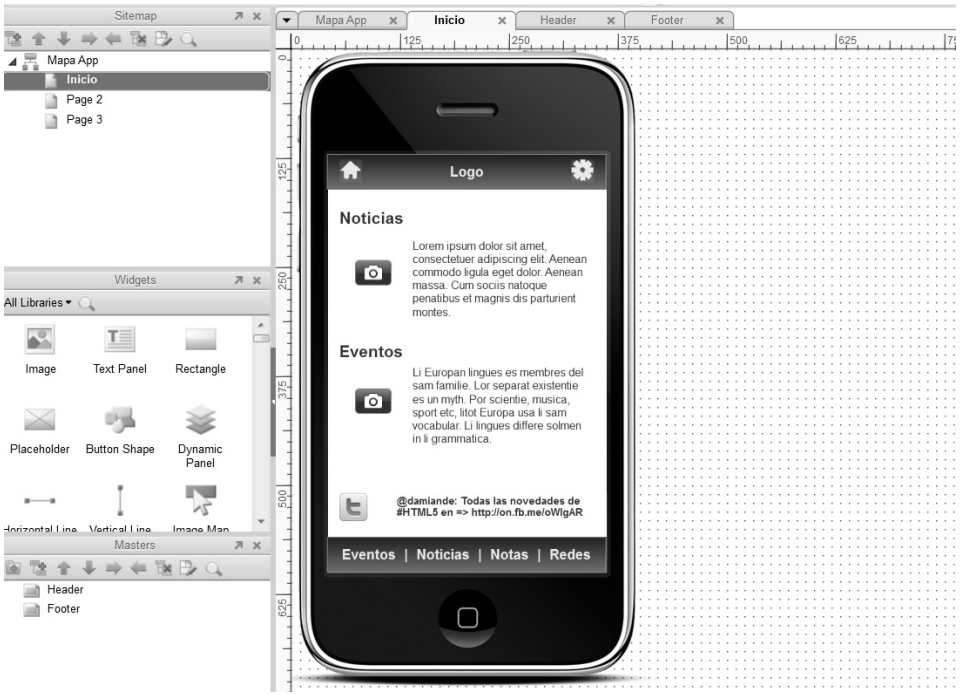

Fig. 2.17. En la pantalla destinada a ventana de inicio, para este ejemplo, se ha colocado el cuerpo de un iPhone, y además de los elementos propios, se han agregado la cabecera y el pie definidos en el Master del proyecto.

Las ventanas de Noticias y Eventos son muy similares en su diagramación, ya que mostrarán información de una fuente de Internet que ofrece a los usuarios las últimas noticias publicadas y los eventos promocionados por el sitio vinculado con esta aplicación.

En esta vista para móviles, dispuesta en formato vertical, colocaremos tres noticias por pantalla e incorporamos una opción de paginado para que el usuario pueda acceder a otras páginas en el caso de que exista más contenido. Otra opción que puede aplicarse es la de generar un paginado que muestre el resto del contenido a medida que el usuario hace *scroll*. Esta opción se hizo muy popular desde la época de AJAX y es usada por muchas redes sociales.

Fig. 2.18. En las secciones de *Noticias* y *Eventos* se incorporará un elemento, en la parte superior de cada uno de ellos, para que el usuario sepa dónde se encuentra. Este *breadcumb* se aplicará también al resto de las pantallas del interior de la aplicación.

En la sección de **Notas,** les brindaremos a los usuarios la posibilidad de que ingresen sus propios textos, los cuales serán almacenados en el móvil.

De esta manera, en la sección **Notas**, en primer lugar, mostraremos las últimas ingresadas por el usuarios y facilitaremos el acceso para agregar otras nuevas. Las notas no llevarán foto o imagen relacionada, pero sí tendrán una sección y un enlace opcional, que podrá ser especificado por el usuario para cada una de ellas.

El almacenamiento local de información en el dispositivo es una de las principales características que se introduce a partir de HTML5, y abre un abanico de nuevas posibilidades para los desarrolladores. Aún no nos introduciremos en su funcionamiento, pero es interesante saber que hay diversos métodos para hacerlo y que puede funcionar incluso si la aplicación está *offline*. En próximos capítulos aprenderemos más sobre estas características.

Fig. 2.19. Desde la pantalla de notas se agrega, debajo de la barra de *breadcomb*, la opción de acceso a leer notas y a crearlas. Por defecto, al acceder a esta pantalla arrancamos en la alternativa de leer notas.

Cuando el usuario elige la opción Crear una nota, pasará a una pantalla de formulario donde podrá seleccionar una categoría, ingresar el texto de la nota y añadir un enlace relacionado en el campo definido para la URL.

Junto a la opción que permite elegir una categoría, se ofrecer un enlace que llevará a otra ventana, donde se podrá agregar o quitar categorías para las notas personales que guardará el usuario en el dispositivo. En esa pantalla, veremos un campo de texto que permitirá anexar una nueva categoría y la lista de las categorías ya existentes, que contarán con un botón de borrado por si el usuario quisiera eliminarlas.

Cada acción de agregar o borrar mostrará un mensaje al usuario cuando la operación se ejecute de manera exitosa o si se produce un error. Ofreceremos también la opción de ver más categorías en caso de ser numerosas (a manera de paginado), y también la posibilidad de regresar a la ventana de creación de notas, luego de que el usuario realice todas las operaciones que deseaba en esta pantalla.

Fig. 2.20. La pantalla para crear una nota nueva se complementa con la de agregar o borrar categorías, ya que están íntimamente ligadas a la lógica de acción que puede realizar un usuario al efectuar estas operaciones.

La faceta social es fundamental para nuestra aplicación. Las redes sociales concentran hoy en día buena parte del tráfico que circula en Internet. La mayoría de las personas que accede a la gran red de redes usa alguna de las redes sociales más populares, y suele compartir contenidos en ella. También, muchos usuarios las utilizan para leer o visualizar material multimedia.

Para hablar sobre datos concretos, estadísticas difundidas en el 2013 indican que Facebook tiene más de 1100 millones de usuarios registrados, 750 millones de los cuales accede desde dispositivos móviles. Los números son claros y marcan una subida en la cantidad de usuarios que frecuentan las redes sociales *top* del momento. Si observamos datos de años anteriores, por ejemplo, Facebook tenía 100 millones de usuarios en 2008, y si bien su nivel de crecimiento en la actualidad no es tan alto como en los primeros años, sigue siendo la red social de mayor audiencia. En el caso de Twitter, en el 2013, supera los200 millones de usuarios activos, con un promedio de 400 millones de mensajes diarios, y una buen dosis de tuits escritos directamente desde *smartphones* y *tablets*.

Ahora sí, regresando a nuestra aplicación, en la pantalla de **Redes Sociales** colocaremos una lista con las direcciones y enlaces para acceder a las redes sociales relacionadas con la aplicación. En el ejemplo que estamos desarrollando

enlazaremos con las cuentas de Facebook, Twitter, Google+, YouTube y Flickr vinculadas con las app.

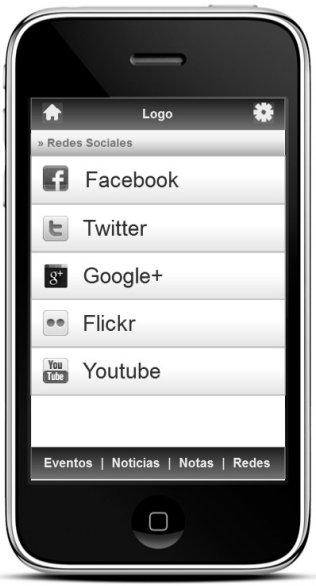

Fig. 2.21. La pantalla de redes sociales quedará resuelta con una lista de enlaces.

Nos queda ahora la tarea de crear la pantalla de configuración. En esa sección, el usuario podrá establecer algunas preferencias: combinación de colores de la aplicación, ubicación geográfica (para ofrecerle eventos relacionados con su zona), borrar notas (permite borrar todas las notas guardadas por el usuario en el equipo), ayuda y acerca de (información de la aplicación, versión y datos del creador).

Si bien la ubicación geográfica (como veremos más adelante) puede ser obtenida mediante técnicas de geolocalización que llegan de la mano de HTML5, en este caso optaremos por que el usuario elija su ubicación.

Esta decisión, vinculada particularmente con un objetivo de este desarrollo, apunta a ofrecer una lista de lugares, ya que no se ofrecerán eventos en todo el mundo. La lista podrá tener países y/o ciudades, pero también podría estar conformada por regiones en particular u ofrecer una opción como “resto del mundo”.

Está claro que la pantalla de configuración puede ser más amplia, dependiendo de las necesidades de cada aplicación, y puede ofrecer la opción de regresar a la opción original o a los valores de fábrica, como aparece en algunos dispositivos.

Fig. 2.22. Desde la pantalla de configuración, tanto los botones de *Ayuda y Acerca de* abren una ventana de diálogo con un texto correspondiente a cada uno de ellos.

Una vez que tenemos todas las pantallas listas, podemos volver sobre el Sitemap e indicar la relación de las pantallas con cada uno de los elementos que allí definimos. Para lograr esto debemos hacer clic con el botón derecho del mouse sobre el elemento del mapa que deseamos asociar con una ventana. Veremos un menú desplegable en el que debemos seleccionar **Edit Flow Shape**. Dentro del menú que se despliega a continuación, haremos clic sobre **Edit Reference Page** para elegir la página con la cual lo referenciaremos.

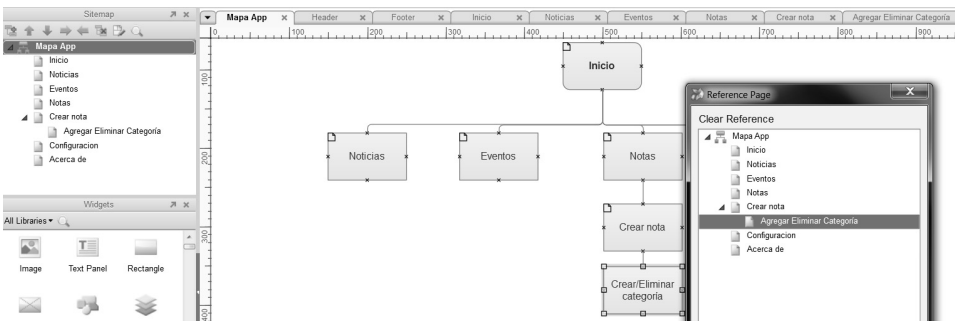

Fig. 2.23. En la ventana *Reference Page* se mostrarán todas las páginas creadas.

Cada página relacionada desde el panel Sitemap mostrará una pequeña página en el ángulo superior izquierdo del elemento que lo representa. No debemos olvidar que tendremos que realizar este procedimiento para todas las páginas que conforman nuestra aplicación.

El diagrama de flujo en Axure también puede ser generado de una forma automática cuando las páginas ya están creadas. Para lograr esto hay que dirigirse a las opciones de Sitemap (ubicado en el panel de la izquierda) y hacer clic con el botón derecho del mouse. En el desplegable, se debe elegir Generate Flow Diagram, y en la ventana siguiente, hay que pulsar sobre Standard.

Generar el prototipo con Axure

Una vez completado todo el proceso de diagramación y conexión de pantallas, estamos listos para crear el prototipo. Dentro de Axure nos dirigimos al menú superior y hacemos clic en **Generate**. Dentro de las opciones que se despliegan elegimos **Prototype...** (También es posible acceder a ella mediante la tecla **F5**).

La ventana que sigue a continuación nos permitirá indicar la carpeta donde se guardará el prototipo que estamos originando. También podremos especificar con qué navegador se abrirá una vez que esté ya generado en nuestro equipo (en algunos casos se puede solicitar habilitar un complemento en el navegador).

Es posible establecer si vamos a generar todas las páginas o seleccionaremos solo algunas de ellas. También, se puede señalar si se incluirán las notas, además de establecer algunos parámetros de configuraciones específicas para pantallas de dispositivos móviles, entre otras opciones avanzadas.

La opción de generar prototipos puede resultar muy útil en nuestro proyecto para compartir este material con el equipo de trabajo, directivos o directamente para enviarlo al cliente. Vale aclarar que si bien esta generación nos brindará archivos HTML, CSS y JavaScript, estos no serán los finales, ya que esta versión interactiva será una muestra y no contendrá nuestra estructura de código definitiva, que veremos cómo programar en los próximos capítulos.

Fig. 2.24. El prototipo ofrece el Sitemap y las notas en un panel a la izquierda; en el centro se verán las páginas que se seleccionen desde el panel. Tendrán interactividad y se podrá acceder de una manera básica, por ejemplo, desplegando opciones de formulario.

Axure también puede exportar a otros formatos. Por ejemplo, podremos generar las imágenes individuales de las páginas a formato JPG, PNG o GIF o BMP. Otra posibilidad sería exportar el proyecto a un documento compatible con Microsoft Word 2007 o superior. El archivo que se crea en este caso es un DOCX. Claro que con esta opción no tendremos el mismo nivel de interactividad que encontraremos con el formato HTML, creado al generar el prototipo.

A partir del próximo capítulo, conoceremos las ventajas que nos brindan HTML5, CSS3 y las API relacionadas con estas tecnologías.

Lenguajes y tecnologías

3

Una nueva etapa para los desarrolladores

Como vimos en los primeros capítulos de este libro, la Web ha atravesado diferentes eras. Los cambios que se han experimentado en este ámbito, por lo general, han sido graduales, aunque existen varios puntos de quiebre o, para decirlo en otras palabras, fechas clave para la evolución de la gran telaraña que domina el mundo de Internet y se expande hacia otros ámbitos.

Luego de transitar varios años por caminos que se podrían imaginar previsibles, sobre el final de la primera década del siglo XXI, el mundo del desarrollo Web comenzó a experimentar una revolución que trajo consigo importantes novedades para el trabajo que realizan desarrolladores y diseñadores.

HTML5 surge, entonces, como el principio de solución para muchas de las necesidades que las nuevas tecnologías exigían. El impulso de HTML5 tiene uno de sus epicentros más importantes en el mundo móvil, y es allí donde se pueden aprovechar al máximo varias de sus características más interesantes.

Pero antes de avanzar en el tema, vale la pena definir qué es HTML5 y derribar algunos mitos que circulan a su alrededor.

En primer lugar, HTML5 es un nombre que identifica una nueva etapa del desarrollo de aplicaciones basadas en Web. Se escribe todo junto como una sola palabra, como un concepto. A diferencia de HTML 4, la cuarta versión del estándar HyperText Markup Language, HTML5 representa a un espectro mucho más amplio de cosas.

En esta historia, es importante comprender que HTML5 no es solamente una actualización de las etiquetas HTML. Si bien esta versión trae algunos cambios e

incorporaciones en las etiquetas de HTML 4 y XHTML, también nos ofrece muchas cosas más a través de las APIs que acompañan a este nuevo estándar. También CSS3, el nuevo nivel de las hojas de estilo en cascada, puede considerarse como parte de las características que llegan con HTML5.

Podríamos resumir, entonces, las características más importantes que promueven HTML5 y los estándares asociados a estas tecnologías, con los siguientes enunciados:

- Semántica.
- Nuevas características para estilos.
- Animaciones.
- Multimedia.
- Acceso a hardware.
- Geolocalización.
- Almacenamiento local.
- Funcionamiento *offline*.
- Conectividad en tiempo real cliente-servidor.
- Capacidad para generar contenidos 3 D.

En este capítulo, profundizaremos acerca de estas características y nos centraremos especialmente en aquéllas que utilizaremos en el desarrollo de nuestra aplicación.

La evolución de HTML

Tim Berners-Lee (http://www.w3.org/People/Berners-Lee/) es una de las personalidades más importantes de la historia de la informática moderna. Su labor lo identifica como un hombre clave en la historia de la Web, ya que su trabajo permitió establecer las bases para la estandarización de varios de los servicios, lenguajes y protocolos que usamos a diario al utilizar Internet.

A principios de la década de los noventa, Tim Berners-Lee escribió y publicó el primer documento con los elementos iniciales de HTML (HTML *Tags*). En 1993, llegaría la primera especificación impulsada por el IETF (*Internet Engineering Task Force*). Recordemos que en esa época aún no existía Internet Explorer ni la mayoría de los navegadores que conocemos en la actualidad. La Web móvil tal como la conocemos hoy era en ese tiempo solo materia de ciencia ficción. En aquella época, el contenido podía ser visualizado por Mosaic, uno de los primeros navegadores Web gráficos junto a ViolaWWW. Algunos nostálgicos podrán recordar que Mosaic contaba con versiones para sistemas Window y Mac, pero con el tiempo fue cediendo su espacio a navegadores más modernos.

La siguiente versión de HTML fue conocida originalmente como HTML+. Estableció un avance interesante en este lenguaje de etiquetas, pero nunca llegó a estandarizarse como HTML 2.

En 1994, Tim Berners-Lee crea el W3C y la versión 3.0 de HTML aparece como borrador en 1995, bajo la supervisión e impulso de este nuevo Consorcio. En 1997, llega HTML 4, y posteriormente, la especificación HTML 4.01 logra su forma de recomendación definitiva el 24 de diciembre de 1999.

XHTML surge posteriormente como una alternativa más estricta para la construcción de la estructura del lenguaje, con reglas más estrictas para escribir las etiquetas y los atributos.

A diferencia de HTML, que desciende de SGML, XHTML se basa en XML. Vale la pena mencionar que a pesar de esto, las etiquetas de HTML 4 y XHTML son en su mayoría las mismas.

A continuación veremos algunas reglas que diferencian XHTML de HTML:

- En XHTML, los elementos vacíos deben cerrarse (en HTML no es obligatorio).
- En XHTML, el orden de los elementos anidados debe respetar siempre el orden de apertura y cierre.
- En XHTML, el valor de los atributos debe llevar siempre comillas de apertura y cierre (en HTML si es una sola palabra o número el valor puede ir sin comillas).
- En XHTML, las etiquetas y los atributos deben escribirse en letra minúscula (en HTML es válido escribirlos en mayúsculas).

Existen también otras reglas que diferencian XHTML de HTML, como por ejemplo, que algunos elementos que no son obligatorios en HTML o su cierre es opcional. Sin embargo, hay que tener cuidado, porque en muchos casos hay reglas que establecen cuándo está permitido o no hacerlo. Para tener un panorama completo sobre las diferencias entre HTML y XHTML, se recomienda leer el siguiente documento publicado por el W3C:

(http://www.w3.org/TR/xhtml1/diffs.html)

El momento de HTML5

La última especificación de HTML 4 data del año 1999, un largo tiempo si se tiene en cuenta la velocidad con la que ocurren los cambios en el mundo informático y, en especial, en el universo de Internet.

El dilema que se planteó a la hora de definir la evolución de HTML fue precisamente si se continuaba el rumbo por el lado de lo que sería XHTML 2 o si se enfocarían los esfuerzos en la quinta versión de HTML.

Si bien se dieron algunos pasos en torno a la definición de XHTML 2.0 (http://www.w3.org/TR/xhtml2/), luego se decidió discontinuar el avance sobre esta

alternativa y se optó por apoyar lo que sería HTML5. La novedad de esta versión es que el concepto de HTML se enriquece no solo con nuevas etiquetas, sino también con la combinación de otras tecnologías, para ofrecer una solución a la altura de las necesidades del desarrollo Web moderno.

La estandarización de esta nueva versión se encuentra en manos del W3C, y sus documentos avanzan en su desarrollo. Un aspecto importante para destacar es que no todas las tecnologías mencionadas en este libro son parte de lo que se podría considerar el núcleo de HTML5. Existen especificaciones propias de algunos fabricantes, y otras que han sido parte del estándar que son aplicables a los desarrollos actuales, y que, por tal motivo, es importante conocerlas.

Los primeros bosquejos para definir el "Timeline" de HTML5 fueron propuestos por Ian Hickson, en 2006, aunque luego tendrían algunas modificaciones en las fechas, fue Hickson quien condujo las primeras etapas de la estandarización. Posteriormente, Hickson se inclinó por enfocar sus esfuerzos en el grupo WHATWG, para continuar con el desarrollo de HTML Living Standar.

HTML5 reconoce compatibilidad con HTML4, quitando del estándar solamente aquellas etiquetas que ya no resultan útiles (obsoletas), modificando los elementos que necesitan actualizarse y agregando todo un nuevo mundo de posibilidades. Como veremos más adelante, muchas de las etiquetas que conocíamos siguen vigentes en esta versión.

Con HTML5, llega también una nueva etapa en lo que respecta a compatibilidad multiplataforma y con las diferentes versiones de los navegadores. Esto nos permitirá identificar y conocer uno de los aspectos más significativos e importantes que debemos tener en cuenta a la hora de determinar si es factible aplicar o no, algunas de las nuevas características en un desarrollo.

Es importante resaltar que no todos los documentos se encuentren en el mismo estado de evolución, ya que algunos están más avanzados y pueden tener mayor soporte en los navegadores modernos, gracias a que hace tiempo que cuentan con una forma medianamente estable y han podido implementarse. Los primeros avances de HTML5 en los navegadores más populares comenzaron a registrarse entre 2009 y 2010, con una importante evolución en el último tiempo.

En las próximas páginas vamos a recorrer y conocer los elementos y características que componen HTML5.

Los elementos de HTML5

Dentro de las características y renovaciones que ofrece HTML5, encontramos elementos que se mantienen en el estándar, nuevos elementos, y otros que se modifican y son retirados. Es importante destacar también, que algunas de las nuevas características de HTML5 no están directamente vinculadas con una etiqueta en particular, sino que son APIs de JavaScript. Algunos ejemplos de esto

podrían ser WebWorkers, WebSockets o Geolocalizacion, aunque esta última no es una API HTML5, generalmente la encontraremos de la mano de esta tecnología.

A continuación, nos centraremos en los elementos y atributos más importantes que ya podemos utilizar de HTML5, y más adelante, nos dedicaremos a recorrer las ventajas que introducen las APIs.

Estructura del documento y semántica

Uno de los objetivos de HTML5 es hacerle la vida más fácil a diseñadores y desarrolladores, por lo cual veremos simplificados algunos códigos que conocíamos hasta aquí. Además, recordemos que HTML5 es HTML, por lo cual es más flexible con algunas reglas (respecto a XHTML). Por esta razón, ahora la declaración del tipo de documento de nuestra paginas HTML5 es:

```
<!DOCTYPE html>
```

Las etiquetas que permiten definir la cabecera (`<head>`) y el cuerpo del documento (`<body>`) no son obligatorias en HTML como sí lo son en XHTML. Por tal motivo será opcional utilizarlas, aunque si venimos trabajando desde hace tiempo con XHTML, nos puede resultar más ordenado continuar aplicándolas.

La etiqueta título (`<title>`) es obligatoria, y las metaetiquetas (`<meta>`) que utilicemos dependerán de las necesidades de nuestro proyecto, ya que ahora tendremos algunas opciones específicas que nos ayudaran al enfocarnos en móviles. Para nuestro proyecto podremos incorporar la meta definida para ajustar el *Viewport*. Nos permitirá trabajar con `width` (ancho), `height` (alto), `initial-scale` (escala inicial), `minimum-scale` (escala mínima), `maximum-scale` (escala máxima) y `user-scalable` (permite indicar si será posible que el usuario haga *zoom* en la aplicación). A continuación, veremos un ejemplo de cómo utilizarlo:

```
<meta name="viewport" content="width=device-width, initial-
scale=1, maximum-scale=1">
```

Otra novedad que puede resultar interesante conocer es que si utilizamos las etiquetas `<style>` y `<script>` no será obligatorio especificar el `type`, siempre y cuando estemos trabajando con CSS y JavaScript (las opciones por defecto para estos elementos).

Estructura del cuerpo del documento

Durante varios años, el maquetado de sitios con marcos (*frames)* y tablas fue tendencia en el mundo de Internet. Posteriormente, el uso de la etiqueta `<div>` marcó una instancia superadora en el maquetado de los sitios Web.

A partir de HTML5, el uso de `<framset>` sale de la recomendación, mientras que las tablas solo deben utilizar para crear contenido tabulado, es decir, tablas (salvo casos excepcionales, como en el diseño de *newsletters*).

En la nueva estructura HTML5, la etiqueta `<div>` se define como un contenedor genérico, y ahora es acompañada por nuevas etiquetas semánticas que nos permitirán estructurar cada página de un sitio o una ventana de una aplicación de una manera más adecuada y precisa de la que teníamos hasta el momento.

Etiqueta	Descripción
`<header>`	Permite definir un bloque de encabezado. Por lo general, lo tendremos aplicado en la parte superior de la página y de cada contenido que lo requiera.
`<hgroup>`	Agrupa un conjunto de títulos, por ejemplo un título y un subtítulo, que podrían estar representados por <h1> y< h2>, contiguos.
`<section>`	No es un contenedor genérico, sino que debe aplicarse a contenido que define una sección dentro de la página o en un artículo. La pregunta que debemos hacernos a la hora de utilizarlo es si el contenido debería tener por lógica un título.
`<article>`	Se aplica para contenido que puede funcionar de manera independiente como, por ejemplo, artículos de un periódico o de un blog.
`<aside>`	Se puede aplica para información que guarda relación con el contenido principal, pero que si se quitara no afectaría la comprensión para el lector.
`<footer>`	Permite definir el pie de la página o de un contenido.

Fig. 3.1. Nuevas etiquetas semánticas de HTML5.

Estas etiquetas aceptan los mismos atributos globales que el `<div>`, que a partir de HTML5 cuentan con incorporaciones:

Atributo	Descripción
accesskey	Tecla para enfocar el elemento.
class	Clase que se aplica al elemento.
contenteditable	Permite indicar que el elemento puede ser editado por el usuario.
contextmenu	Permite modificar el menú contextual.
dir	Dirección del texto para el contenido del elemento.
draggable	Permite indicar si el elemento puede ser arrastrado.
dropzone	Zona donde se puede arrastrar y soltar el elemento

	establecido como *draggable*.
hidden	Se utiliza para indicar que un elemento no es relevante y el navegador no lo mostrará.
id	Id del elemento.
lang	Idioma del contenido del elemento.
spellcheck	Permite indicar si el elemento tendrá ortografía y gramática marcada.
style	Permite incorporar estilos CSS *inline*.
tabindex	Se emplea para especificar el *tab order* de los elementos.
title	Posibilita incluir un título para el elemento, que actuará como información sobre él.
translate	Se puede utilizar para marcar si un elemento debería o no ser traducido en caso de aplicarse un proceso de traducción.

Fig. 3.2. Atributo globales para las etiquetas semánticas de HTML5.

Para trabajar con fechas y hora, se ha propuesto la etiqueta `<time>`. Ofrece los atributos `datetime` (fecha y hora en formato accesible por un agente informático) y también `pubdate` (permite indicar que es una fecha de publicación). Veamos un ejemplo a continuación para comprender su uso en la fecha de publicación de una entrada o noticia de un blog:

```
<time datetime="2012-08-10T15:52:27+00:00" pubdate="">10 de
Agosto de 2012</time>
```

En los navegadores modernos, el soporte para los elementos semánticos de HTML5 es alto. Para navegadores antiguos, por ejemplo: Internet Explorer 8 y anteriores, se pueden utilizar algunos Polyfills, como HTML5Shiv (http://code.google.com/p/html5shiv/) (distribuido bajo licencia MIT y GPL).

Trabajo con títulos y texto

Para el texto, en HTML5 podremos utilizar la mayoría de los elementos que conocíamos, algunos de los cuales tienen ligeras modificaciones.

Para estructurar los niveles de título de un documento, se mantienen vigentes las etiquetas `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>` y `<h6>`. La de mayor jerarquía es `<h1>` y las demás siguen el orden descendente. Es importante recordar que debemos utilizar estas etiquetas de acuerdo a su jerarquía, ya que la representación la podremos manejar, como siempre, desde CSS.

Para definir un párrafo utilizamos la etiqueta <p> y dentro de ella podemos envolver texto utilizando las etiquetas:

- <i>: siempre fue utilizada como itálica, ahora cambia su significado semántico (su representación por defecto sigue siendo la misma). A partir de HTML5 debe emplearse como una voz alternativa en el texto. Un ejemplo de uso es un texto en otro idioma, en cuyo caso se recomienda especificar el atributo lang, para indicar la lengua.
- <b>: históricamente se lo aplicaba para obtener una negrita en el texto. A partir de HTML5 su resultado visual es el mismo, pero su significado semántico apunta a definir un texto que cambia el estilo de la narración o texto. Podría utilizarse para palabras clave en el texto, nombres de marcas o productos, entre otras opciones.
- : texto con énfasis que por defecto se representa en itálica.
- : es un texto destacado que en la mayoría de los navegadores se muestra en negrita.
- : es un contenedor genérico en línea. Se puede utilizar, por ejemplo, para dar estilo a un texto.
- <mark>: texto resaltado (por defecto con el color de fondo amarillo). Es una opción nueva de HTML5.
- <abbr>: permite definir una abreviatura, por lo general, trabaja con el atributo title.

Para realizar citas largas de texto contamos con la etiqueta <blockquote>; y para realizar citas cortas dentro del texto, se puede aplicar la etiqueta <q>. Mediante el atributo cite tendremos la posibilidad de incluir la URL del texto citado (es solo información para el agente informático, no genera un *link* para el usuario).

La etiqueta <small>, que anteriormente se definía como un texto más pequeño, ahora se aplica para definir un *Small Print* o lo que podríamos definir como un pie de imprenta para el sitio (usualmente lo que se puede ubicar en el <footer>). Su contraparte, <big>, pasa a ser una etiqueta desaconsejada por el estándar.

La etiqueta <a>, utilizada como un ancla para realizar hipervínculos entre la propia página u otros enlaces externos.

Trabajo con listas

Las listas nos ofrecen una amplia gama de soluciones en la estructuración de un sitio o aplicación Web, ya que además de permitirnos listar contenidos, también nos pueden resultar útiles para crear menús u otras alternativas que puedan contener varios ítems.

Contamos con tres tipos de listas: listas ordenadas <ol>), listas desordenadas (<ul>), y listas de definiciones (<dl>). El primer caso nos permite contar con

numeración; y el segundo, probablemente el más utilizado, nos da la posibilidad de especificar distintos tipos de *bullets* y personalizar su representación mediante CSS. Ambos tipos de listas trabajan en conjunto con ítems definidos con la etiqueta `<li>`.

En HTML5, no es obligatorio cerrar los `<li>`, siempre y cuando sean delimitados por el siguiente ítem o por el cierre de la lista (`</ol>` o `</ul>`).

Para `<ol>` se agrega la posibilidad de utilizar el atributo `reversed`, el cual nos permite hacer que la lista se muestre de manera descendiente. Los atributos `start` y `type` siguen siendo soportados, mientras que *compact* se retira del estándar.

Para `<ul>`, en HTML5 ya no están soportados los atributos *compact* y *type* (debemos usar las propiedades de CSS en su reemplazo).

En el caso de las listas de definiciones, en lugar de trabajar con `<li>`, tendremos como elementos hijos a la definición del ítem (`<dt>`) y a la descripción del ítem (`<dd>`).

En nuestro proyecto, trabajaremos con listas desordenadas para crear menús como en el ejemplo que veremos a continuación.

```
<ul>
      <li>Elemento 1 del menu
      <li>Elemento 2 del menu
      <li>Elemento 3 del menu
      <li>Elemento 4 del menu
</ul>
```

Trabajo con tablas

Como veíamos anteriormente, las tablas continúan dentro del estándar. En el diseño Web general, deben utilizarse para organizar contenido tabulado y ya no para el maquetado. Solo en casos excepcionales aún se emplean tablas para estructurar contenido, como el caso de una *newsletter* o cuando se desarrolla para dispositivos muy antiguos, cuyos navegadores no soportan otras nuevas características.

En HTML5, no se introducen cambios significativos en el mundo de las tablas. Se retiran del estándar los atributos fuera de uso la etiqueta `<table>`, ellos son: `align`, `bgcolor`, `cellpading`, `cellspacing`, `frame`, `rules`, `summary` y `width`. El atributo `border` sigue siendo soportado, pero ahora sus valores posibles son `"1"` o `""` (para que tenga borde o no).

A continuación, vamos a repasar las principales etiquetas que podemos utilizar para estructurar una tabla:

- `<thead>`: es el elemento que se emplea para crear el encabezado de la tabla.
- `<tbody>`: se utiliza para definir el cuerpo de la tabla.
- `<tfooter>`: es el elemento indicado para crear el pie de la tabla.
- `<tr>`: permite definir filas en la tabla.
- `<th>`: se emplea en celdas de encabezado.
- `<td>`: se puede utilizar para crear celdas en las filas de la tabla y se puede anidar como hijo de `<tr>`, tanto en el cuerpo como en el pie de la tabla.

Los elementos de las tablas tienen ahora soporte para los atributos globales de HTML5.

Si bien no están estrictamente prohibidas, las tablas no son recomendadas para el desarrollo de móviles moderno (y mucho menos anidarlas). Teniendo en cuenta las posibilidades de presentación, en *smartphone*, muchas veces es recomendable ofrecer la representación de los resultados en listas, una opción que puede ser más eficaz, en estos casos.

Gráficos y multimedia

La posibilidad de brindar contenidos multimedia era uno de los puntos flojos del estándar HTML, ya que no existía ninguna opción nativa para mostrar otras cosas que no fueran texto e imágenes.

Hasta aquí, estas posibilidades estaban resueltas generalmente mediante la incorporación de agregados que requerían un *plugin* para funcionar en el navegador, como el caso de Flash o Silverlight.

HTML5 incorpora el soporte nativo de audio y video a través de etiquetas especificas, y también nos trae, con Canvas, nuevas opciones para la creación de gráficos, juegos y aplicaciones que abren la puerta a una nueva gama de posibilidades.

Audio y video HTML5

El soporte nativo para audio y video era una de las características más esperadas en el último tiempo, y HTML5 trae la solución que hacía falta para cubrir este espacio.

Mediante la etiqueta `<audio>` ahora es posible acceder a un reproductor de archivos de sonido en el navegador. Los formatos que se pueden utilizar son MP3, OGG y WAV, aunque no todos los navegadores los soportan por igual.

Los atributos disponibles para utilizar con esta etiqueta son: `autoplay`, `controls`, `loop`, `preload` y `src`.

Mediante la etiqueta `<source>` podremos definir varias fuentes para ofrecer una solución de compatibilidad con los diferentes formatos.

La etiqueta `<video>` nos permite incorporar *clips* en nuestros sitios, sin necesidad de *plugins* en los navegadores compatibles. Los formatos soportados son MP4 (códec de video H.264 y de audio AAC), WEBM (códec de video VP8 y audio Vorbis) y OGG (códec de video Theora y audio Vorbis).

Aquí nuevamente nos encontramos con diversidad de soporte por temas de compatibilidad, y es recomendable recurrir a la etiqueta `<source>` para incluir el mismo video en diferentes formatos.

Son requeridos los atributos `width` y `height` para indicar el tamaño del video, y podremos incluir de manera opcional los atributos `autoplay`, `controls`, `loop`, `muted`, `preload`, `src` y `poster`. Este último nos permite indicar la ruta de una imagen que actuará como miniatura del video. Algunas de estas características, como `autoplay`, pueden no estar disponibles en todos los navegadores, especialmente en móviles, donde algunos fabricantes han decidido no habilitarla para evitar que el video comience a reproducirse y consuma ancho de banda del usuario.

También vale la pena mencionar que tanto para audio como video, el reproductor nativo en cada navegador y sistema puede ofrecer un aspecto diferente. Existen técnicas mediante CSS y JavaScript que nos permiten personalizar el aspecto del reproductor para lograr una alternativa homogénea en todos los *browsers*. En estos casos, puede ser recomendable recurrir a un *plugin* o *polyfill* que nos provea de una solución probada y nos simplifique el trabajo.

Para quienes necesitan realizar procesos avanzados sobre el audio, el W3C está trabajando sobre una nueva especificación que lleva el nombre de Web Audio API. Esta API se describe como una opción de alto nivel que permite mediante JavaScript trabajar sobre características de sintetización de audio en aplicaciones Web. La documentación se encuentra disponible en: http://www.w3.org/TR/webaudio/.

Acceso a cámara y micrófono

Otras novedades que se introducen en HTML5 respecto de multimedia es la capacidad de acceder a la cámara y el micrófono del dispositivo para capturar contenido. Esto introduce un nuevo potencial para las aplicaciones Web, que podrán ofrecer sin necesidad de *plugins* ni agregados, comunicaciones en tiempo real, soluciones para videovigilancia y grabación de contenidos desde el medio, entre otras alternativas.

La clave de esta especificación llega de la mano de `getUserMedia()`. Para obtener video y mostrarlo en un documento, en primer lugar, podemos escribir:

```
<video width="640" height="480" id="camara"></video>
```

A continuación, escribimos el *script* para obtener el *stream* de video y colocarlo dentro de la etiqueta correspondiente:

```
navigator.getUserMedia({video: true, audio: true},
function(localMediaStream) {
    var camara =
document.getElementById('camara');
    camara.src =
window.URL.createObjectURL(localMediaStream);
  }, function() {
     console.log("Algo no ha salido bien");
  });
```

Al correr el código en el navegador, se solicitará un permiso a través de un cartel. Esto es un control de seguridad para que el usuario pueda elegir si desea autorizar el acceso a cámara y micrófono. Para poder lograr compatibilidad *crossbrowser* debemos trabajar también con:

- `navigator.webkitGetUserMedia` (navegadores con webkit)
- `navigator.mozGetUserMedia` (navegadores con Mozilla Firefox)
- `navigator.msGetUserMedia` (futuras versiones de Internet Explorer)

En navegadores de escritorio, esta característica es compatible en Firefox a partir de la versión 17; en Google Chrome, desde la versión 21; y en Opera en la versión 12. En móviles puede ser utilizada desde el navegador en dispositivos con BlackBerry 10.

Encontraremos más información sobre esta especificación en el documento del W3C Media Capture and Streams: http://www.w3.org/TR/mediacapture-streams/.

Para trabajar con streaming, algunos dispositivos ofrecen compatibilidad con el protocolo RTSP (Real Time Streaming Protocol). Mediante este estándar se puede establecer la conexión entre dos puntos y controlar el flujo. Es extensible, seguro y puede utilizarse para soluciones profesionales. Más información sobre esta especificación en: http://www.ietf.org/rfc/rfc2326.txt.

Canvas

HTML5 nos trae un poderoso lienzo para dibujar: Canvas. Esta característica es una de las de mayor soporte de HTML5, y también una de las más poderosas, ya que nos permite dibujar en un área determinada la página mediante *scripts*, y también generar interacción con el usuario mediante una API.

Para ser más precisos, vale la pena mencionar que Canvas fue introducido por Apple en WebKit Mac OSX, en 2004; y luego también, adaptado por otros navegadores como Firefox. El grupo de trabajo del W3C decidió incorporar el elemento Canvas en el estándar de HTML5, impulsando la estandarización de una API que permite dibujar en un área del sitio de la página y, a su vez, permitir interacción. Esto posibilita el desarrollo de aplicaciones y juegos que antes requerían el uso de otras tecnologías.

Canvas se incorpora en el documento HTML mediante la etiqueta `<canvas>`, es obligatorio definir su ancho y alto con los atributos `width` y `height`. También es recomendable indicar la id para poder seleccionar el elementos, que luego trabajaremos desde su API con JavaScript.

A continuación, veremos cómo definir la etiqueta en el documento:

```
<canvas id="juego" width="640" height="320">
      Mensaje para navegadores no compatibles
</canvas>
```

Desde el *script* debemos inicializar el Canvas de la siguiente manera:

```
var canvas = document.getElementById('juego');
if (canvas.getContext){
  var ctx = canvas.getContext('2d');
  // Código para el Canvas.
} else {
  // Código para navegadores no compatibles.
}
```

En la siguiente tabla, encontraremos un detalle de los principales métodos que nos permiten crear trazos, dibujar, aplicar rellenos e incorporar imágenes en el Canvas de HTML5:

Método	Descripción
beginPath()	Comienza un *path*.
moveTo()	Mueve el puntero de dibujo hasta un punto indicado (recibe coordenadas de x e y), pero no dibuja el trazo.
lineTo()	Agrega un punto a línea en el trazo para permitir dibujar el *path* entre los puntos especificados (recibe coordenadas x e y).

closePath()	Crea una línea entre el punto actual y el comienzo del *path*, para poder cerrarlo.
stroke()	Dibuja el *path* definido.
arc()	Permite crear arcos y sirve también para definir formas circulares. Recibe coordenadas de x, y, radio, ángulo inicial y ángulo final.
arcTo()	Se puede emplear para crear un arco o una curva entre dos tangentes. Para describirlo recibe dos coordenadas de x, y; la quinta representa el radio: arcTo(x1,y1,x2,y2,r).
bezierCurveTo()	Permite definir una curva Bezier.
quadraticCurveTo()	Permite definir una curva Bezier cuadrática.
fill()	Llena la figura.
strokeRect()	Dibuja el rectángulo sin relleno.
fillRect()	Dibuja un rectángulo con relleno.
clearRect()	Recibe como parámetros los valores de x, y, ancho y alto, para limpiar (o borrar) los píxeles indicados en un rectángulo dibujado.
createLinearGradient()	Crea un *gradient* lineal.
createRadialGradient()	Crea un *gradient* radial.
addColorStop()	Se utiliza para trabajar en conjunto con los métodos para crear *gradients*. Permite indicar los pasos que tendrá el *gradient*.
createPattern()	Crea un *pattern*.
drawImage()	Dibuja una imagen en el Canvas (también puede trabajar con video).
createImageData()	Crea un nuevo objeto ImageData.
getImageData()	Permite conseguir la información del objeto ImageData, obteniendo la información de cada píxel de información.
putImageData()	Se emplea para colocar la información de un objeto ImageData en el contexto del Canvas.

Fig. 3.3. Métodos para crear trazos, figuras, imágenes y rellenos en Canvas.

Canvas soporta un conjunto de propiedades que nos permiten aplicar estilos, colores, sombras y fondos. En la siguiente tabla, podremos observar estas propiedades y conoceremos para qué se pueden utilizar.

Propiedad	Descripción
fillStyle	Color, *gradient* o relleno que llena una figura del Canvas.
strokeStyle	Color, *gradient* o relleno para el trazo de la figura.
shadowColor	Color de la sombra.
shadowBlur	Nivel de desenfoque de la sombra.
shadowOffsetX	Movimiento (*offset*) en el eje X de la sombra. Por defecto su valor es 0.
shadowOffsetY	Movimiento (*offset*) en el eje Y de la sombra. Por defecto su valor es 0.
lineCap	Estilo de la tapa o final de la línea. Puede recibir los valores *butt* (valor por defecto), *round* o *square*.
lineJoin	Estilo de la línea cuando se junta con otra. Puede recibir los valores *round*, *bevel* o *miter* (valor por defecto).
lineWidth	Ancho de la línea.
miterLimit	Longitud del ángulo máximo. Por defecto su valor es 10.

Fig. 3.4. Propiedades soportadas por Canvas para estilos, colores, sombras y fondos.

Canvas ofrece algunos métodos pensados especialmente para realizar transformaciones:

- `scale()`
- `rotate()`
- `translate()`
- `transform(`
- `setTransform()`

Cuando se trabaja, por ejemplo, con transformaciones en animaciones o desarrollo de juegos, pueden resultar muy útiles los métodos `save()` y `restore()`, que permiten guardar y restaurar (respectivamente) el estado de un contexto. El estado se almacena en una pila (*stack*), y al recuperarse se devuelve al contexto.

Abarcar todas las características que brinda la API de Canvas puede ser muy extenso y podría demandar un capitulo completo o, incluso, hasta un libro dedicado al tema. En este caso, hemos abordado los conceptos básicos para que el lector pueda conocerlo y saber cómo aplicarlo, y luego profundizar con otros contenidos específicos.

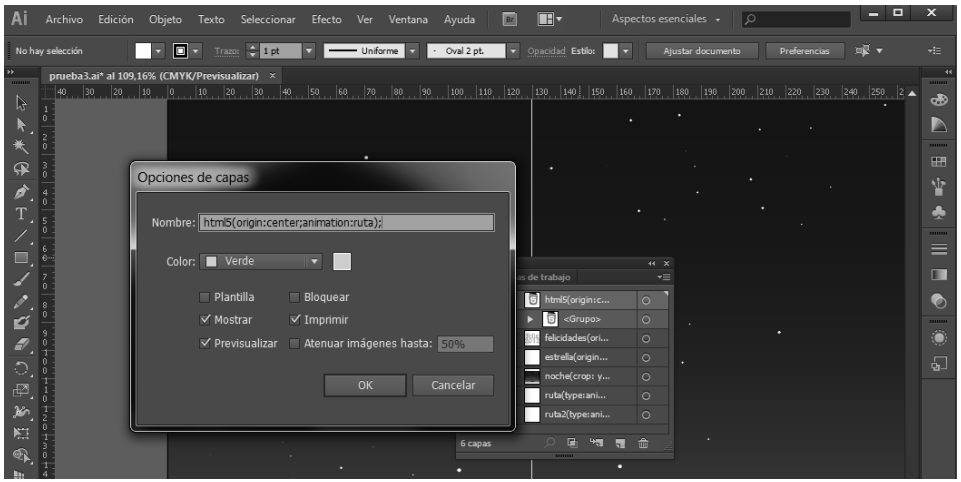

Fig. 3.5. AI2Canvas (http://visitmix.com/labs/ai2canvas/) es un plugin que se puede incorporar a Adobe Illustrator para exportar a HTML5 Canvas. Mediante esta extensión, es posible exportar dibujos estáticos o animados, trabajando directamente sobre las capas de Adobe Illustrator.

Encontraremos más información sobre cómo trabajar con el contexto 2D de HTML5 Canvas en el documento del W3C: http://www.w3.org/TR/2dcontext/. Allí se especifica cómo se puede programar con la API de Canvas para crear líneas, *paths*, textos, figuras, rellenos y transformaciones, entre otras características de este elemento. También se explican y muestran ejemplos de buenas prácticas.

El soporte de Canvas es uno de los más antiguos de HTML5 y está presente en la mayoría de los navegadores modernos para Desktop (Internet Explorer 9 o superior, Chorme, Firefox, Safari y Opera), y en Mobile (iOS Safari 3.2 o superior, Android 2.1 o superior, BlackBerry Browser 7 o superior, Opera Mobile 10.0 o superior, y también en otros modelos de navegadores como Google Chrome for Android y Firefox for Mobile).

WebGL introduce la posibilidad de mostrar e interactuar con gráficos 3D directamente en el navegador. Esta característica cuenta con una API en JavaScript y aprovecha el procesamiento del GPU para representar los gráficos. WebGL se encuentra bajo la órbita de Khronos Group
(http://www.chromeexperiments.com/webgl/)
En HTML5, se puede utilizar mediante el elemento Canvas.

SVG

Scalable Vector Graphics (SVG) es una especificación que ha llegado a ser recomendación del W3C, y que permite definir gráficos vectoriales mediante un lenguaje de etiquetas (basado en XML).

Se diferencia de Canvas, principalmente, porque trabaja con vectores en lugar de píxeles y, además, porque SVG permite definir gráficos mediante etiquetas y atributos. SVG también acepta el uso de estilos, y permite la manipulación de sus nodos y atributos mediante un lenguaje *script* (se puede trabajar directamente con ECMAScript).

La posibilidad de integrar estos gráficos con un documento HTML ya existían hace tiempo. La novedad que introduce HTML5 es la posibilidad de incorporar SVG *inline* en un documento HTML5 con la etiqueta <SVG> (característica soportada por navegadores modernos). A continuación, analizaremos las opciones con las que contamos para incorporar contenidos en formato SVG en nuestra página o aplicación Web.

- Embebido con la etiqueta <embed>. Con buena compatibilidad y soporte de *scripts* es una opción válida en HTML5 para SVG, pero era desaconsejado en HTML4/XHTML:

```
<embed src="prueba.svg" type="image/svg+xml" />
```

- Embebido con la etiqueta <object>. Ofrece buena compatibilidad y admitido por HTML4, XHTML y HTML5. El problema es que no soporta *scripts*:

```
<object data="prueba.svg"
type="image/svg+xml"></object>
```

- Embebido mediante la etiqueta <iframe>. Brinda buen soporte crossbrowser, pero entre las desventajas se suman que no es recomendado en HTML4 y XHTML, además tendrá las limitaciones que pueden tener los iframes:

```
<iframe src="prueba.svg"></iframe>
```

- Colocado en el cuerpo del documento HTML Para esto se emplea la etiqueta <SVG>, que es soportada solo por navegadores modernos:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect
  width="200" height="200"
  style="fill:blue;stroke-width:2;stroke:red"/>
</svg>
```

Para conocer más sobre el formato SVG, es recomendable acceder a la documentación que ofrece el grupo de trabajo en:

http://www.w3.org/Graphics/SVG/.

Raphaël (http://raphaeljs.com/) es el nombre de una librería JavaScript que nos facilita el trabajo con gráficos vectoriales en la Web. También nos ofrece una muy buena solución crossbrowser, ya que podremos lograr buena compatibilidad con navegadores antiguos, utilizando SVG y también VML.

Formularios

Luego de la importante evolución que experimentaron los formularios gracias a AJAX, se comenzaron a buscar otras soluciones para potenciar su uso y cubrir algunas necesidades básicas de una manera más simple. HTML5 incorpora características nativas que potencian a los formularios y nos ayudarán a minimizar el uso de agregados externos para su creación. Algunas de estas características están vinculadas con la validación de datos de manera nativa, y otras se encuentran relacionadas con nuevos controles u opciones que se incorporan para definir tipos de campos.

Para la etiqueta `<form>` se incorporan los atributos `autocomplete` (para habilitar o deshabilitar el autocompletado de los campos) y `novalidate` (atributo que permite indicar que no se validen los datos). También contamos con atributos que pueden aplicarse a los controles para definir si se permite el autocompletado y la validación. Veamos un ejemplo:

```
<form action="procesar.php" name="formulario"
autocomplete="on" novalidate>
  <label>Nombre:</label>
  <input type="text" name="nombre">
  <label>Apellido</label>
  <input type="text" name="apellido">
  <label>E-mail:</label>
  <input type="email" name="email">
  <label>Tarjeta de crédito</label>
  <input type="text" name="tarjeta"
  autocomplete="off">
  <input type="submit">
</form>
```

Dentro de los controles de formulario, `<input>` es el que recibe la mayor actualización en HTML5. A continuación, veremos los principales atributos que se incorporan para este control:

Nuevo atributo para <input>	Descripción
autocomplete	Se puede activar el autocompletado con el valor *on* o desactivar con *off*.
autofocus	Indica que el campo debe recibir el foco al cargar la página.
form	Permite indicar el identificador (id) del formulario al que pertenece el control.
formaction	Se puede utilizar para indicar la URL del recurso que procesa el control del formulario. Puede ser utilizado para los tipos *submit* o *image*.
formenctype	Permite indicar como la información será codificada al ser enviada. Puede ser utilizado para los tipos *submit* o *image*.
formmethod	Método de envío, puede ser *get* o *post*. Se puede utilizar para los tipos *submit* o *image*.
formnovalidate	Si se incluye indica que el campo no debe ser validado al enviarse el formulario.
formtarget	Puede ser usado con los tipos *submit* o *image* y recibe los valores _blank, _self, _parent o _top. Es el destino de la respuesta que se recibe al enviar un formulario.
placeholder	Se puede utilizar para incorporar un texto temporal que el usuario verá hasta que escriba en ese control. Recibe como valor un texto y puede utilizarse también en otros controles, como en <textarea>.
Required	Al incluirse este atributo, se indica que el campo es obligatorio y debe completarse.

Fig. 3.6. Nuevos atributos para la etiqueta <input>.

Otra característica interesante para destacar en los formularios es que a partir de HTML5 es posible validar los campos con expresiones regulares, por medio del atributo `pattern`, que puede trabajar en conjunto con el atributo `required`. A continuación, veamos un ejemplo de uso para validar que un campo de texto solo permita que el usuario ingrese valores alfanuméricos:

```
<input type="text" name="alfanum"
pattern="[a-zA-Z0-9]+" required />
```

Dentro de la importante renovación e incorporación de características que HTML5 introduce para la etiqueta `<input>`, también se destacan los nuevos

valores que se pueden utilizar para el atributo `type`. A continuación, veremos las principales opciones que se incorporan:

Nuevo valor de type para <input>	Descripción
date	Formato de fecha (sin la hora).
datetime	Fecha y hora.
week	Número de semana del año.
month	Mes y año.
time	Hora, minuto, segundo y fracción.
datetime-local	Fecha y hora sin la zona horaria.
number	Permite ingresar un número y ofrece un control para subir y bajar en la numeración. Soporta un valor inicial (*value*), un mínimo (*min*), un máximo (*max*) y un valor para el paso o intervalo (*step*).
range	Posibilita seleccionar un número mediante un deslizador. Al igual que con *number*, es posible usar los atributos para valor inicial (*value*), mínimo (*min*), máximo (*max)* y paso (*step*).
tel	Número de teléfono.
search	Campo de búsqueda.
url	Dirección de un sitio o recurso Web (URL).
email	Dirección de correo electrónico.
color	Permite elegir un color dentro de un cuadro cromático. Devuelve su valor en formato hexadecimal.

Fig. 3.7. Valores del atributo type incorporados en HTML5 para la etiqueta <input>.

Dentro de los elementos nuevos que se introducen para formularios, destacamos:

- `<datalist>`: esta etiqueta trabaja en tándem con un `<input>`. Se vinculan mediante la `id` del `<datalist>` y el valor del atributo `list` del campo `<input>`. El `<datalist>` permite especificar una lista de valores, definidos cada uno como `option`. Esa lista de valores se mostrarán como sugerencias para el usuario cuando acceda al *input* vinculado con este `<datalist>`.
- `<output>`: permite mostrar los resultados de una operación o cálculo. Se vincula con los campos de entrada, y puede emplearse para operaciones matemáticas, cálculo, subtotales, etcétera.
- `<keygen>`: es un campo que permite definir un generador de claves. Al ser enviado, la clave privada se guarda localmente en el navegador y la pública se

envía al servidor. Soporta el atributo `keytype`, para indicar el algoritmo de seguridad, que puede ser `rsa`, `dsa` o `ec`.

En el caso de formularios HTML5, hasta el momento, no es homogéneo el soporte y la manera en que cada navegador muestra las características. Por esta razón, es recomendable si vamos a utilizar las opciones que nos brinda HTML5 para formularios, trabajar además con algún *framework* de JavaScript y CSS para lograr un resultado uniforme en la mayor cantidad de *browsers* posible.

Una solución que nos puede resultar útil es jQuery UI (http://jqueryui.com/) si trabajamos desarrollando aplicaciones Web para plataformas *desktop*. Por su parte, jQuery Mobile (http://jquerymobile.com/) nos ofrece soluciones cuando se trata de móviles. Sobre esta librería hablaremos más adelante en este libro.

Otras alternativas específicas para formularios son: h5validate (http://ericleads.com/h5validate/), JSColor (https://github.com/jo/JSColor) y WebShims Lib (http://afarkas.github.com/webshim/demos/), entre otras opciones.

Otros elementos que se incorporan al estándar

Además de los que hemos visto anteriormente, en HTML5, se agregan otros elementos que vale la pena repasar, sus etiquetas son:

- `<meter>`: puede utilizarse para mostrar una barra escalar con los valores de un intervalo definido. Puede ser útil, por ejemplo, para mostrar la capacidad de un dispositivo.
- `<progress>`: permite mostrar el progreso de una operación, como por ejemplo, la descarga de un contenido o copia de archivos.
- `<details>`: brinda la posibilidad de definir un detalle que por defecto se mantendrá oculto (excepto en el caso de que se le asigne el atributo `open`). Trabaja con la etiqueta `<summary>`, que siendo un elemento hijo de `<details>`, permite definir el texto visible.
- `<wbr>`: se emplea para definir dónde es posible cortar una palabra cuando no pudiera entrar en una misma línea.

Dentro del listado de etiquetas que se retiran del estándar, a partir de HTML5, encontramos las siguientes: <acronym>, <applet>, <basefont>, <big>, , <font>, <frame>, <frameset>, <noframes>, <strike> y <tt>.

El nuevo rol de JavaScript

En sus primeros años de vida, JavaScript ofrecía algunos efectos y características limitadas. Sin embargo, con la evolución de la Web y la llegada de AJAX, JavaScript tuvo un renacer que redefinió su rol en el mundo de Internet.

HTML5 le da una nueva vida y un papel protagónico a este lenguaje que tiene un interesante resurgimiento en el ámbito del desarrollo Web. Muchas empresas buscan personal calificado que domine JavaScript avanzado y, en muchos casos, resulta complejo encontrar a la persona que se ajuste al perfil. Esta es otra buena razón para sumar conocimientos y experiencia en JavaScript, ya que nos puede facilitar una buena salida laboral, especialmente si sabemos combinarlo con lo que ofrece HTML5.

Si bien JavaScript y las técnicas que definen AJAX son muy amplias y podrían demandar un libro completo, en el apéndice de este libro, encontrarán un breve resumen de estas características para aquellas personas que deseen repasar estos conceptos o tener una visión general de ellos.

JavaScript más allá del cliente

Un cambio importante en el enfoque de JavaScript está dado en que ya no debe considerarse solamente como un lenguaje para trabajar del lado cliente. En la actualidad, JavaScript también ofrece opciones para tecnologías del lado servidor.

Node.js (http://nodejs.org/) y EJScript (http://ejscript.org/) son solo algunos de los ejemplos de lo que se puede hacer hoy en día con JavaScript del lado servidor. Utilizando una arquitectura basada en eventos, los servidores que trabajan con estas tecnologías pueden lograr soluciones eficientes, de baja latencia, funcionando en equipos que no requieren tantos recursos como otras alternativas del mercado.

Estás soluciones son cada vez más utilizadas en producción y apuntan a dar solución a las necesidades del mercado moderno, tanto para "*stream* de contenidos" como para soluciones en tiempo real.

Otras APIs de HTML5

Anteriormente mencionamos que una de las características más destacadas de HTML5 llega de la mano de las APIs de JavaScript que potencian las capacidades actuales del desarrollo Web.

Algunas de estas APIs pueden trabajar en conjunto con las nuevas etiquetas, como por ejemplo la API Media, que se complementa con las ya mencionadas etiquetas `<audio>` y `<video>`, pero también encontraremos otras que trabajan de manera completa desde JavaScript como, por ejemplo, Websockets, Web Workers o Local Storage. A continuación, vamos a conocer otras de las APIs que nos ofrece HTML5, y también sus características principales.

Local Storage y Session Storage

El almacenamiento de información persistente o de sesión en el lado cliente, por mucho años fue resuelto principalmente con el uso de *cookies*. Esta estrategia, algo limitada, pero efectiva en algunos casos puede resultar insuficiente para las necesidades que en la actualidad demanda el desarrollo Web.

HTML5 nos ofrece una solución por medio de **Local Storage** (almacenamiento persistente) y **Session Storage** (almacenamiento de sesión).

Para guardar un valor de sesión lo hacemos con el siguiente *script*:

```
sessionStorage.setItem('nombre_clave','valor');
```

Y lo recuperamos indicando el nombre de la clave:

```
sessionStorage.getItem('nombre_clave');
```

De manera similar, podemos guardar datos persistentes con Local Storage:

```
localStorage.setItem('nombre_clave','valor');
```

Y lo recuperamos con el siguiente código:

```
localStorage.getItem('nombre_clave');
```

Si deseamos remover un ítem almacenamos podemos utilizar:

```
localStorage.removeItem('nombre_de_clave');
```

Como el almacenamiento clave/valor solo soporta cadenas, si necesitamos guardar información en un formato como JSON, deberíamos utilizar los métodos que nos provee JavaScript para transformar en cadena y luego recuperar el formato original. Mediante el método `JSON.stringify()` podremos transformar un objeto JSON en una cadena, que luego podremos guardar como el valor asignado a un ítem en el Storage. Para volver a convertir la cadena a objeto JSON, se puede aplicar el método `JSON.parse()`.

A la hora de recuperar la información guardada, mediante el atributo `lenght` podemos saber la cantidad de elementos (clave/valor) guardados que están asociadas al objeto Storage correspondiente. Esta opción que nos ofrece `lenght` puede ser muy útil para obtener el número total de ítems y recorrerlos. A continuación veremos un ejemplo en el cual vamos a recuperar los valores de las claves guardadas en el Local Storage (asociadas al objeto que corresponde a nuestro dominio). En este caso, primero, crearemos una estructura de control en JavaScript utilizando `for`, recuperaremos la información, y luego mostraremos las claves y los valores recuperados en la consola.

```
for (i=0; i<=localStorage.length-1; i++){
clave = localStorage.key(i);
resultado = localStorage.getItem(clave);
console.log(clave + ' : ' + resultado);
}
```

Si deseamos limpiar todos los elementos (claves/valor) asociados al objeto de Storage con el que estamos trabajando, podemos utilizar el método `clear()` que se aplica como vemos a continuación (no requiere clave/valor):

```
localStorage.clear();
```

Las características relacionadas con LocalStorage tienen alta compatibilidad *crossbrowser* en navegadores de escritorio y también en móviles. Para mayor información sobre el uso de estas características, podemos consultar la documentación que ofrece el W3C ingresando en:

http://www.w3.org/TR/webstorage/.

Web SQL Database

Al momento de definir una base de datos local para el estándar, en primera instancia, se pensó en permitir un modelo relacional que pueda trabajar con el lenguaje SQL. Así nació la idea de Web SQL Database. Sin embargo, el documento de esta especificación (http://www.w3.org/TR/webdatabase/) se discontinuó por parte del W3C en favor de otras opciones.

Si bien esta característica no se estandarizará por parte del W3C, existen varios navegadores que ya la tienen incorporada de manera funcional y la mantienen. Algunos navegadores que la soportan en sus versiones para sistemas de escritorio son: Chrome, Safari y Opera. En el mundo móvil, encontramos esta característica funcionando en: iOS Safari, Android Browser, Chrome para Android, Opera Mobile y BlackBerry Browser.

Los principales métodos para trabajar con esta especificación de acceso a base de datos local son:

- **openDatabase**: permite abrir la base de datos o crearla, en caso de no existir. Como argumento, puede recibir el nombre de la base de datos, el número de versión, el texto que la describe y el tamaño estimado.
- **transaction**: permite crear transacciones con la base de datos.
- **executeSql**: brinda la posibilidad de leer o escribir datos. Se puede utilizar SQL para pasar *queries* con este método.

A continuación, veremos un ejemplo de cómo abrir o crear una base de datos:

```
var base = openDatabase('miDB', '1.0', 'Mi base local', 5 *
1024 * 1024);
```

Como parámetros de la operación anterior, se pasaron el nombre de la base de datos, la versión, una descripción y un tamaño estimado de megabytes.

Una vez creada y abierta la conexión a la base, para crear una tabla, si no existe, podemos usar los comandos que conocemos de SQL. En este caso, usamos `CREATE TABLE`, aplicado en una transacción:

```
tran.executeSql('CREATE TABLE IF NOT EXISTS mitabla (id
unique, nombre)');
```

Para la inserción de datos, contamos con `INSERT`:

```
tran.executeSql('INSERT INTO mitabla (id, nombre) VALUES
(1, "Gael")');
```

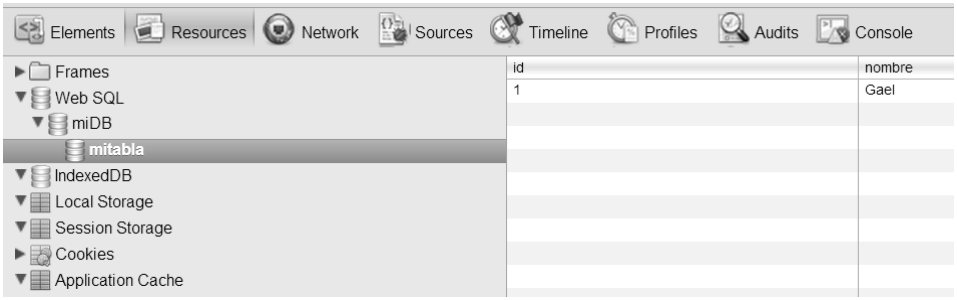

Fig. 3.8. Mediante el inspector de Google Chrome, en la solapa *Resources*, podremos visualizar el los datos guardados de Web SQL en el navegador.

Para trabajar con SQL Database, se recomienda repasar los conceptos vinculados con bases de datos relacionales y con el lenguaje SQL.

IndexedDB

Al descartar el avance sobre Web SQL Database, el W3C decidió enfocarse en otro tipo de base de datos para el almacenamiento en el equipo cliente. Es así como surge la API de IndexedDB. Mediante el uso de índices, podremos lograr un mejor rendimiento y, además, por las características de IndexedDB tendremos una mayor capacidad de almacenamiento que en otras alternativas.

Para abrir una base de datos utilizamos el método `open()`, que recibe como argumentos el nombre y la versión de la base:

```
var request = indexedDB.open("miDB", 1);
```

Esta API puede trabajar de manera asíncrona, ofreciendo una solución de alto rendimiento y permitiendo que las transacciones no bloqueen el hilo principal de la aplicación. En la tabla siguiente, veremos las principales características de IndexedDB.

Objeto	Descripción
IDBFactory	Permite acceder, crear y manipular objetos almacenados en la base de datos.
IDBCursor	Es la interfaz que define un curso, y se puede utilizar para realizar la iteración sobre objetos almacenados en la base de datos e índices.
IDBCursorWithValue	Permite realizar iteraciones sobre objetos almacenados e índices. Brinda la posibilidad de acceder a la propiedad *value*, que retorna el valor de la información en la posición actual del cursor.
IDBDatabase	Contiene los métodos que permiten crear o borrar un *object store* y realizar transacciones.
IDBIndex	Permite acceder a la metadata de un índice y ofrece métodos para contar el número de registros de un *object store* (correspondientes a una clave o rango), obtener la clave de un índice y otros métodos relacionados con cursores.
IDBKeyRange	Brinda la posibilidad de definir un rango para aplicar un filtro dentro de un conjunto de valores.
IDBObjectStore	Es la representación de un *object store* de la base de datos, y cuenta con métodos para agregar o borrar registros, crear o borrar índices, realizar reemplazos y trabajar con cursores.
IDBOpenDBRequest	Se emplea para enviar el requerimiento de apertura de la base de datos.
IDBRequest	Representa el acceso a una petición realizada a la base de datos y a los objetos de la base de datos.
IDBTransaction	Transacción a la base de datos.

Fig. 3.9. Objetos de la especificación de IndexedDB.

Al ser una de las API de HTML5 impulsadas en el último tiempo, IndexedDB aún no tiene soporte en todos los navegadores del mercado. IndexedDBShim (https://github.com/axemclion/IndexedDBShim) es un *polyfill* que permite habilitar el uso de IndexedDB mediante navegadores que soportan Web SQL. Se lo puede ver como un adaptador, que puede funcionar tanto en navegadores *desktop* como móviles con PhoneGap.

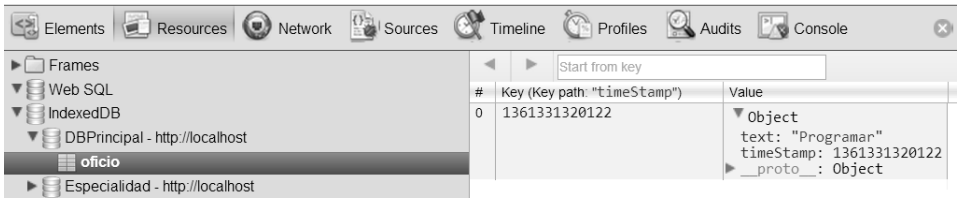

Fig. 3.10. La estructura y la información almacenada de IndexdDB puede ser vista en la solapa *Resources*, en las *Herramientas para desarrolladores* del navegador Google Chrome.

Encontraremos información detallada acerca de las características que introduce IndexedDB, ingresando en el documento del W3C:

http://www.w3.org/TR/IndexedDB/.

Al momento de escribir estas páginas, este documento se encontraba en el estado *Candidate Recommendation*.

Offline Web Application

Construir aplicaciones Web que puedan funcionar *offline* es uno de los grandes retos de esta etapa de Internet. A partir de HTML5, esto se puede lograr mediante la creación de un archivo que actúa como manifiesto, donde se especifican los archivos que pueden trabajar *offline* y aquellos que requieren conexión.

El mecanismo para trabajar con Offline Application Caching API se encuentra detallado en el documento del W3C y, para implementarlo en nuestro proyecto, se destacan las características que vamos a conocer a continuación.

En primer lugar, debemos asegurarnos que el documento Web principal al que usuario accede, contenga la etiqueta `<html>` de la siguiente forma:

```
<html lang="es" manifest="cache.manifest">
```

El código anterior permite indicar el nombre y la ubicación del archivo manifiesto. Este archivo debe ser ofrecido por el servidor en formato MIME type `text/cache-manifest` y la extensión debe ser **.manifest**.

Ahora es posible comenzar a especificar los recursos en el archivo del manifiesto. Para crearlo o editarlo, podemos utilizar cualquier editor de texto o código. Una vez creado el archivo, debemos indicar la versión del manifiesto y escribir, en la sección correspondiente, el nombre de los archivos que deseamos que funcionen *online* y de aquellos que lo harán de modo *offline*. Para comprender mejor cómo especificar las secciones del manifiesto, a continuación, veremos un ejemplo:

```
CACHE MANIFEST
# VERSION 1.0
CACHE:
pag1.html
img/imagen1.jpg

NETWORK:
pag2.html

FALLBACK:
pagina.html
```

El archivo **.manifest** cuenta con tres secciones principales: `CACHE` (archivos que se incluirán en el cache), `NETWORK` (archivos que requieren conexión con el servidor) y `FALLBACK` (alternativa si no está disponible un recurso que requiere conexión).

Para saber el estado del caché, utilizamos la propiedad `status` del objeto `applicationCache`. Los eventos vinculados con este objeto son los siguientes:

Evento	Descripción
onchecking	Es el primer evento que se dispara para verificar actualizaciones y nuevos contenidos.
onerror	Cuando ocurre un error al realizar alguna operación relacionada con el cache.
ondownloading	Ocurre cuando comienzan a descargarse los archivos al cache.
onprogress	Ocurre cuando cada elemento comienza a descargarse al cache.
oncached	Después de que se pudo realizar el cache.
onnoupdate	Cuando el proceso de actualización ha concluido y el archivo **.manifest** no ha cambiado.
onupdateready	La actualización se ha efectuado y el cache está listo para ser utilizado.
onobsolete	Ocurre cuando el manifiesto para un sitio se ha eliminado (error de servidor 404 o 410), el contenido cacheado pasa a ser obsoleto.

Fig. 3.11. Eventos de Application Cache.

Una vez que el contenido nuevo o actualizado se encuentra disponible, y listo para ser utilizado en el caché del navegador del equipo local, deberemos refrescar

los datos en la vista del usuario. Para visualizar estos cambios, se recomienda llamar al método `applicationCache.swapCache()`. Otra alternativa, puede ser recargar la página que comenzar a ver los cambios efectuados.

Los estados de actualización son los siguientes: `UNCACHED` (0), `IDLE` (1), `CHECKING` (2), `DOWNLOADING` (3), `UPDATEREADY` (4) y `OBSOLETE` (5).

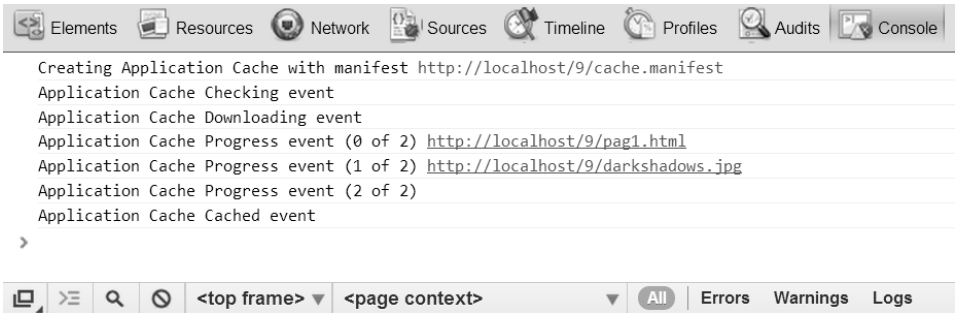

Fig. 3.12. En Google Chrome, accediendo a la solapa *Console* de las *Herramientas para desarrolladores*, es posible observar el progreso del proceso de cacheo para seguir los estados y saber si el procedimiento se ha completado de manera exitosa o con algún error.

Si se está trabajando con un servidor Apache puede ser necesario agregar en la configuración de **.htaccess** la siguiente línea: **AddType text/cache-manifest .appcache**.
Ante cualquier duda sobre qué es y cómo configurar el archivo .htaccess, se puede consultar la documentación de Apache HTTP server en:
http://httpd.apache.org/docs/current/howto/htaccess.html.

File API

La capacidad de interactuar de una manera eficiente con archivos locales sin la intervención del servidor era un tema pendiente para las aplicaciones Web. A partir de HTML5, se definen nuevas interfaces para superar las limitaciones existentes.

Algunos ejemplos de soluciones que puede ofrecer esta API son: vista previa de imágenes antes de subirlas, sin necesidad de pasar por el servidor; ver datos de tamaño y extensión de archivos para filtrarlos de acuerdo con esta información.

Las interfaces que contamos para trabajar son las siguientes:

- **File**: permite leer la información de un archivo (nombre y última fecha de modificación)
- **FileList**: posibilita trabajar con un *array* de archivos
- **Blob**: representa la información binaria de un archivo en formato RAW.

- **FileReader**: provee los métodos para leer un archivo (File o Blob).

El trabajo de la API de archivos de HTML5 debe complementarse con otras tecnologías, como por ejemplo la subida de archivos mediante AJAX u otras técnicas. Si deseamos seleccionar un archivo, es posible hacerlo mediante un campo de formulario como vemos en el siguiente código:

```
<input type="file" name="file" id="archivo">
```

Luego, es posible acceder al valor con JavaScript mediante el siguiente código:

```
var selected_file =
document.getElementById('archivo').files[0];
```

También podremos permitir la selección de varios archivos, si agregamos al `<input>` el atributo `multiple`.

Además, es posible trabajar la interfaz de usuario con la API de Drag & Drop de HTML5, y puede ser recomendable trabajar con hilos (mediante Web Workers) si la aplicación puede demandar muchos recursos en segundo plano.

El tema de compatibilidad es importante para esta API, ya que todavía no goza de una alta compatibilidad en todos los navegadores. El siguiente *script* nos permite verificar estas características, mostrando en consola un mensaje:

```
if (window.File && window.FileReader &&
window.FileList && window.Blob) {
console.log('Funciona el soporte de File API');
}
else {
console.log ('Algunas funcionalidades de File API no se encuentran
disponibles en el navegador.');
}
```

Para conocer más sobre File API podemos leer el documento del W3C: http://www.w3.org/TR/FileAPI/.

Entre sus características, HTML5 incorpora Drag & Drop nativo. Para esto incorpora nuevos atributos para los elementos, y también eventos que permiten manejar estas acciones. La especificación de Drag & Drop se encuentra disponible en la siguiente sección de la documentación de HTML5 del W3C:
http://www.w3.org/TR/html5/editing.html#dnd.

Geolocalización

La capacidad de obtener la ubicación de un dispositivo para ofrecer al usuario la información de geoposicionamiento localizada o ayudarlo a desplazarse en un mapa es una técnica que se encuentra en auge en la actualidad.

Para este fin, el W3C nos ofrece Geolocation API, una alternativa eficaz para poder detectar la ubicación geográfica que se puede obtener de un dispositivo.

Es importante destacar que esta API solamente nos proveerá de datos de ubicación geográfica y no cuenta con mapas asociados. Si necesitamos ubicar las coordenadas en un mapa, deberemos utilizar alguna de las opciones disponibles en Internet, como por ejemplo Google Maps (https://maps.google.com) o Bing Maps (http://www.bing.com/maps/), entre otras opciones.

Para trabajar con las características de geolocalización encontramos tres métodos principales:

- `getCurrentPosition()`: obtiene la posición actual que informa el dispositivo.
- `watchPosition()`: devuelve la posición del dispositivo en lapsos de tiempo, de manera continua.
- `clearWatch()`: detiene y limpia el método `watchPosition()`.

En la mayoría de los desarrollos en los que se trabaja con características de geolocalización, será útil emplear el método `getCurrentPosition()`. Al obtener los datos de posición de un objeto tendremos disponibles las propiedades que veremos en la siguiente tabla:

Propiedades	Descripción
coords.latitude	Latitud (en grados decimales).
coords.longitude	Longitud (en grados decimales).
coords.altitude	Altitud (en metros).
coords.accuracy	Nivel de precisión de las coordenadas que se devuelven (en metros).
coords.altitudeAccuracy	Precisión de la altitud (en metros)
coords.heading	Dirección de navegación o movimiento del dispositivo relativos al norte (en grados, en sentido horario).
coords.speed	Velocidad (en metros por segundo)
Timestamp	Tiempo (cuando la posición es obtenida y el objeto es creado).

Fig. 3.13. Propiedades disponibles al obtener la localización de un dispositivo.

Para detectar si el navegador es compatible con geolocalización, podremos utilizar la siguiente función de JavaScript:

```
function detectaGeo() {
if (!!navigator.geolocation)
{console.log("Funciona Geolocalización");}
else
{console.log("No funciona Geolocalización");}
}
```

Para obtener la posición utilizamos `getCurrentPosition()`:

```
navigator.geolocation.getCurrentPosition
(coor);
```

Luego debemos crear la función `coor()` que será la que obtenga los valores de latitud y longitud del dispositivo:

```
function coor(pos) {
  var lat = pos.coords.latitude;
  var long = pos.coords.longitude;}
```

Las variables `lat` y `long` tendrán los respectivos valores de latitud y longitud obtenidos, que luego podrán ser mostrados en pantalla con su representación numérica o bien pasados a un mapa para señalar las coordenadas.

Encontraremos más información respecto a *Geolocation API Specification* en el documento: http://www.w3.org/TR/geolocation-API/.

Web Workers y Shared Workers

En el uso habitual de AJAX, podremos encontrarnos frente a problemas de concurrencia, si nuestra aplicación requiere de varios procesos trabajando en simultáneo. Si necesitamos crear aplicaciones que requieran el trabajo multihilo, nos será de gran utilidad Web Workers. El enfoque de esta API es permitir que la aplicación pueda correr procesos por separado sin molestar el trabajo de la aplicación principal, evitando por ejemplo que la pantalla se quede "congelada".

Crear un hilo con Web Workers es muy sencillo, simplemente, debemos escribir en JavaScript: `var worker = new Worker("script.js");`.

En lugar de `script.js` debemos indicar el nombre del archive *script* que tiene el proceso que deseamos que corra de manera separada del principal.

Desde el Worker que creamos podemos enviar mensajes empleando el método `postMessage()`. Y los recibimos en la aplicación con `onmessage()`. Para destruir el hilo podemos utilizar `terminate()`.

Al no ser un proceso seguro para el navegador, existen algunas restricciones a la hora de correr un hilo con el Worker, ya que no podrá acceder directamente al DOM ni a los objetos `window` y `document`.

Por lo general, los Workers nos serán útiles en procesos pesados que pueden correr sin problemas en *background*. Algunos ejemplos útiles son: recuperación de datos para ser utilizados posteriormente, consumo de servicios que pueden demorar, filtros o búsquedas en contenidos multimedia o proceso de cálculos complejos.

Los Workers compartidos o Shared Workers nos permiten que diferentes documentos tengan la posibilidad de acceder a la misma instancia. Esto nos provee de una solución para necesidades del desarrollo moderno donde, por ejemplo, nos encontramos con aplicaciones que en diferentes ventanas necesitan acceder a un mismo subproceso que puede estar corriendo en segundo plano.

Lo creamos de una manera similar a lo que hicimos en JavaScript con un Workers: `var sworker = new SharedWorker('script.js');`.

En este caso, los métodos acceden a un objeto denominado `port`. Y tendremos `port.start()`, `port.postMessage()` y para establecer la comunicación.

Es importante destacar que el nivel de compatibilidad *crossbrowser* actual de Web Workers es mayor que el de Shared Workers.

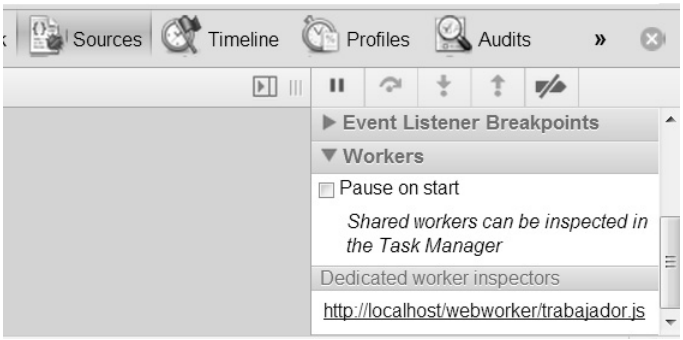

Fig. 3.14. Es posible inspeccionar los Workers desde la solapa Sources en las Herramientas para desarrolladores de Google Chrome.

Para conocer más sobre la especificación de Web Workers podemos consultar el documento del W3C: http://www.w3.org/TR/workers/.

Mediante Web Sockets, HTML5 nos provee de una API con capacidad de conexión bidireccional Full Duplex entre cliente y servidor. Antes de la existencia de Web Sockets, esta característica no estaba disponible directamente en el navegador. La documentación correspondiente a la API de Web Sockets se encuentra en: http://www.w3.org/TR/websockets/.

HTML 5.1 y HTML Living Standard

A pesar de que HTML5 aún no ha llegado a ser estandarizado, ya existen los preliminares de las próximas especificaciones.

HTML 5.1 ya tiene su primer borrador de trabajo, publicado en el 2012, y disponible en: http://www.w3.org/TR/html51/. Entre los editores de esta especificación encontramos personas vinculadas con el W3C, Apple y Microsoft, entre otras empresas.

La idea a la que se apunta es la de tener HTML 5.0 listo en el 2014, y seguir trabajando sobre HTML 5.1 hasta el 2016.

El futuro se encuentra distribuido, por el momento, en dos senderos. Por un lado **HTML.Next** (impulsado por el W3C); y, por otra parte, HTML Living Standard (tendencia respaldada por Ian Hickson en WHATWG).

Los detalles de HTML.Next se encuentran disponibles en: http://www.w3.org/wiki/HTML/next.

Entre la lista de ideas y propuestas para las nuevas versiones de HTML encontramos las siguientes:

- Se ha impulsado como propuesta la inclusión de la etiqueta `<main>`, para que actúe como contenedor del material principal o de valor predominante en la página. Típicamente esto se indica hasta hoy mediante ARIA, utilizando el atributo `role"main"` aplicado a un `<div>` u otro contenedor.
- La etiqueta `<decompress>` permitiría integrar con el navegador un descompresor de archivos ZIP de manera nativa.
- Para integrar geologalización con el marcado HTML se está pensando en crear la etiqueta `<location>`.
- Se está trabajando en el desarrollo de nuevos elementos semánticos para definir títulos y autores (por ejemplo, de libros o notas, entre otras opciones).
- En formularios se plantea la posibilidad de agregar modificaciones para crear `comboBox`.
- La etiqueta `<teaser>` es una propuesta para actuar como *wrapper* de un resumen que pueda tener un enlace que lo amplía. Puede aplicarse a resultados de búsqueda o resumen de noticias de blogs o sitios Web.
- Se plantea la necesidad de incorporar características para trabajar con opciones adaptivas tanto para imágenes como para contenido en *streaming*.

Es importante destacar que estas características se encuentran susceptibles a varias revisiones y aún no hay soporte para ellas, pueden quedar finalmente en el estándar o bien no ser incluidas.

Por su parte, el esfuerzo del WHATWG está enfocado en mantener actualizado HTML de manera continua sin crear o fijar versiones, siempre buscando solucionar errores e incorporar nuevas características viables y útiles para la comunidad que desarrolla con esta tecnología. Quienes no adhieren a esta opción, se basan en la dificultad de seguir un estándar que esté en un flujo permanente de actualizaciones.

Los dos caminos se encuentran abiertos y con el tiempo sabremos cuál logra imponer su rumbo o si se mantendrán ambos por un largo período.

Si deseamos seguir las actualizaciones periódicas de HTML Living Standar, de WHATWG, ingresamos en: http://www.whatwg.org/specs/web-apps/current-work.

CSS: Hojas de estilo en cascada

En este punto del libro, si hemos seguido con atención, ya tenemos claro como se define la estructura semántica mediante el uso de las etiquetas que tenemos disponible en HTML5.

El rol de las hojas de estilo está vinculado con la representación de los contenidos, y a partir de CSS3, también se agregan nuevas capacidades que permiten generar transiciones y también animaciones.

A continuación, repasaremos las funcionalidades que nos ofrece CSS así como su evolución.

La evolución de CSS

Si bien las hojas de estilo tienen décadas de vida, su aplicación en la Web comienza a partir de la década de los noventa. Nos permiten controlar la manera en que se representan los elementos en el medio, y a partir del nivel 3, también crear transiciones y animaciones.

Si repasamos un poco la historia, CSS se incorpora a los estándares que nuclea el W3C desde mediados de los noventa. La primera recomendación del nivel 1 de CSS se publica en 1996. Sus características apuntan principalmente a trabajo con texto, fuentes, colores, dimensiones y alineación. También permite trabajar con imágenes y las opciones básicas de cajas. Aún en este periodo el soporte de CSS por parte de los navegadores era muy rudimentario.

El nivel 2, se publica como recomendación en 1998, ampliando y mejorando varias de las características del nivel anterior, especialmente, para trabajar con cajas y diferentes tipos de medios.

CSS 2.1 (http://www.w3.org/TR/CSS21/) reinó hasta la llegada de CSS3 y provee algunas mejoras y correcciones de lo que teníamos en el nivel 2.

En cuanto a su uso práctico, debemos recordar que para aplicar estilos a una estructura HTML, podemos utilizar CSS de tres maneras:

- Hojas de estilo externas (archivos **.css**), vinculadas en el documento por medio de la etiqueta `<link>`. Ejemplo:

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

- Estilo dentro del documento HTML/XHTML, utilizando la etiqueta `<style>`. Entre la apertura y el cierre de dicha etiqueta podremos escribir reglas CSS que se aplicarán solo a ese documento. Ejemplo:

```
<style>
      p {color: blue; font-size:18px;}
</style>
```

- Agregado de estilo *inline*, es decir, dentro de la etiqueta que se desea afectar, utilizando el atributo `style`. Ejemplo:

```
<p style="color:red;">Texto de prueba</p>
```

CSS3

Los primeros borradores de CSS3 comienzan a discutirse a fines de los noventa, pero su incorporación en los navegadores llegaría casi una década después, aun hoy no todas sus características son soportadas; sin embargo, existen algunos documentos que ya han llegado a ser estandarizados y tienen muy buen soporte en navegadores modernos.

La llegada de CSS3 es una de las mejores noticias que han podido recibir los diseñadores Web, ya que llega en el momento justo para cubrir muchos aspectos que estaban pendientes en la versión anterior de CSS.

Con esta versión de CSS se incorporan nuevos selectores, opciones para trabajar con sombras, contenido adaptable según las características del dispositivo, colores con transparencias, múltiples fondos, imágenes en los bordes, uso de diversas fuentes tipográficas, transiciones y animaciones.

Una de las ventajas de CSS3 es que sus características básicas tienen muy buen soporte en los navegadores de los dispositivos móviles y también en los *browsers* modernos de equipos *desktop*. Es para destacar que CSS3 es modular y no estamos obligados a utilizar todas sus características, sino que podremos emplear las que necesitemos en nuestro proyecto. Existen numerosas soluciones que ofrecen compatibilidad *crossbrowser*, y deben ser evaluadas según las necesidades por cubrir.

El uso de prefijos –webkit, -moz, –o y -ms pueden encontrarse en algunas reglas que fueron introducidas de manera experimental en Chrome, Safari, Firefox, Opera o Internet Explorer, entre otros navegadores. También pueden ser utilizados en propiedades no estándar que se aplican puntualmente a un navegador.

Selectores

Conocer cómo funcionan los selectores en CSS es un tema que nos puede ahorrar muchas horas de trabajo y, a su vez, simplificar el código, evitando tener que recurrir a JavaScript. Dentro de los selectores que contábamos hasta CSS 2.1, encontramos:

- Selector universal (asterisco): `*`.
- Selector de elemento (el nombre del elemento). Ejemplo: `div`.
- Selector de clase (el nombre de la clase con un punto delante). Ejemplo: `.miClase`.
- Selector de ID (el nombre de la ID con un # delante). Ejemplo: `#miID`
- Selector de múltiples elementos (`,`). Ejemplo: `h1,h2,h3,h4,h5,h6`.
- Selector descendente (el elemento que se encuentra dentro de otro, aunque no sea hijo directo). Ejemplo: `p strong`.
- Selector de hijo directo (>). Ejemplo: `div > p`.
- Selector de adyacentes (se utiliza con el símbolo +). Ejemplo: `h1 + p`.
- Selectores de atributos (permiten seleccionar un elemento por su atributo o por el valor de su atributo). Ejemplo (por atributo): `p[class]`. Otro ejemplo (por el valor del atributo): `img[title="Paisaje"]`.
- Pseudoclases (se indican con `:` seguido al elemento). Ejemplos: `:link`, `:visited`, `:active` y `:focus`.
- Pseudoelementos (se indican con `:` seguido al elemento). Ejemplos: `:first-child`, `:first-letter`, `:first-line`, `:after` y `:before`.

Algunos comentarios importantes respecto de los selectores vistos hasta aquí:

- El selector universal (*) afecta a todos los elementos y puede traer problemas en el diseño si se lo utiliza de manera incorrecta. Es menos eficaz, especialmente en móviles, ya que aplica las reglas a cada uno de los elementos posibles.
- El valor de cada ID debe ser única en cada documento, las clases pueden reutilizarse, ya que para eso fueron creadas.

Resets en CSS ¿sí o no? Esta práctica es habitual desde hace tiempo para proyectos pensados en *desktop*, pero si se piensa en opciones para todos los dispositivos o más aún si se enfoca en móviles, es recomendable no utilizar *Resets* genéricos (y mucho menos el selector universal *), sino que es mejor ajustar las propiedades en los elementos que se utilizarán y crear opciones personalizadas.

En CSS3 algunos de los selectores existentes modifican su sintaxis, utilizando a partir de ahora `::` en lugar de `:`. Ellos son: `::first-line`, `::first-letter`, `::before` y `::after`. También aparece un nuevo selector para aplicar a selección de texto: `::selection`.

Para las pseudoclases, CSS3 incorpora `:nth-child(nº)`,`:nth-last-child(nº)`, `:empty`, `first-child`, `:last-child`, `nth-of-type(nº)` y `:nth-last-of-type(nº)`.

Media Queries

Desde CSS2 teníamos la posibilidad de definir el tipo de medio para las reglas de estilos. Por defecto es para todos, pero también podemos apuntar a una impresora, pantalla o dispositivo de mano, entre otras opciones.

Para entender de qué manera funciona esta característica, debemos comprender que el tipo de medio lo informa el navegador, con lo cual dependerá del fabricante definir cuál es el valor de `media`. Por ejemplo, si necesitamos establecer una hoja de estilos específica para impresión, podríamos crear un archivo CSS que sería invocado solamente para dicha finalidad, como veremos a continuación, con la etiqueta `<link>` desde el documento HTML:

```
<link rel="stylesheet" type="text/css"
media="print" href="impresion.css">
```

Así como hemos visto anteriormente que tenemos la posibilidad de indicar el tipo de medio al llamar al archivo CSS, también podremos hacerlo desde la parte de estilos. A continuación, veremos cómo establecer un color a los párrafos desde los estilos CSS, para los dispositivos que se presenten como `screen`:

```
@media screen {
  p { color: blue; }
}
```

Dentro de las posibilidades que nos brinda `media`, los tipos de medios a los que es posible apuntar son: `all`, `aural`, `braille`, `handheld`, `print`, `projection`, `screen`, `tty` y `tv`.

En la actualidad, muchas *tablets* y *smartphones* se presentan con el atributo `media` fijado como `screen`. Por esta razón, no podemos diferenciarlo de una pantalla de un monitor o de una *laptop* con las herramientas con las que contábamos en CSS 2.1. A partir de CSS3, se incorpora Media Queries, una opción que nos permite definir reglas para diferentes medios. A continuación, veremos las principales características que se introducen con Media Queries.

Característica	Descripción
Width	Ancho de la ventana o área disponible para la representación. Soporta *min* y *max*.
height	Alto de la ventana o área disponible para la representación. Soporta min y max.
device-width	Ancho del medio o la pantalla del dispositivo. Soporta *min* y *max*.
device-height	Alto del medio o de la pantalla del dispositivo. Soporta *min* y *max*.
orientation	Orientación del dispositivo. Puede recibir los valores *portrait* o *landscape*.
aspect-ratio	Es el ratio o proporción del área de representación disponible. Soporta *min* y *max*.
device-aspect-ratio	Es el ratio o proporción que ofrece la pantalla del dispositivo. Soporta *min* y *max*.
color	Cantidad de bits de color del dispositivo. Si no es un dispositivo color informará 0. Soporta *min* y *max*.
resolution	Resolución del medio. Soporta *min* y *max*.

Fig. 3.15. Principales características de Media Queries.

Para trabajar con Media Queries se pueden utilizar como operadores el `and`, el `not` y la coma (`,`) para especificar una lista. A continuación, veremos un código que ejemplifica el uso de Media Queries para determinar por el ancho de pantalla si se debe aplicar una hoja de estilos.

```
<link rel="stylesheet" media="screen and (min-width:320px)
and (max-width:480px)" href="estilomovil.css" />
```

El código anterior puede incorporarse en la cabecera de un documento HTML y se aplicará en caso de que se cumplan las reglas. También es posible establecer reglas condicionales de Media Queries directamente en los estilos CSS como vemos en el siguiente código:

```
media (min-width: 640px) and
(orientation: landscape) {
nav {float:left;
         width:100%;}
```

```
}
```

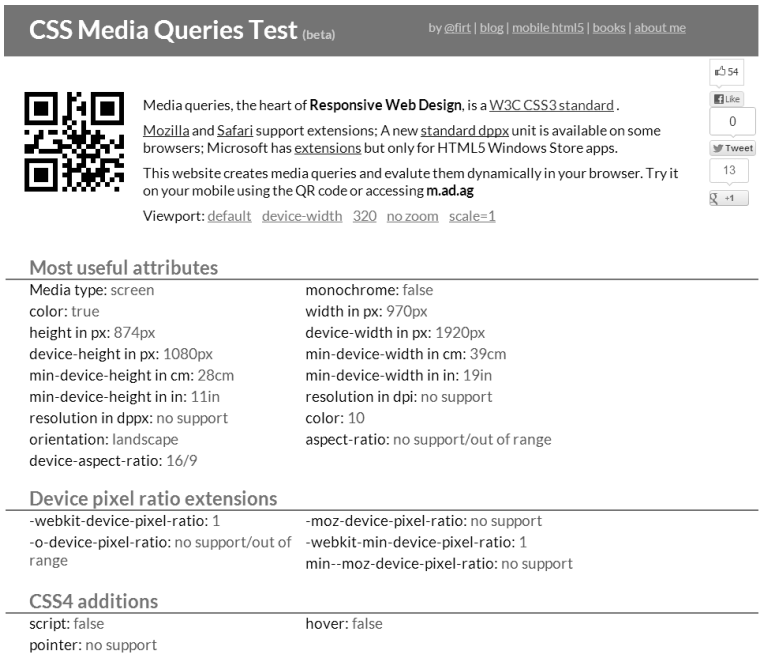

Fig. 3.16. CSS Media Queries Test (http://mediaqueriestest.com/) es una herramienta online que permite evaluar Media Queries en el navegador de manera dinámica.

Vale la pena mencionar la existencia de la características `device-pixel-ratio`, introducida hace un tiempo por varios *smartphones* y *tablets*. Como la densidad de píxeles puede ser diferente en los dispositivos, se creó esta característica para poder diferenciarlos. Para citar un ejemplo práctico, los dispositivos iOS de pantalla retina utilizan informan `-webkit-min-device-pixel-ratio: 2`, mientras que los que no son retina ofrecen el valor `1`, para esta característica.

El ejemplo que sigue a continuación apunta los estilos a un iPad que no cuenta con pantalla retina (iPad 1, iPad 2 o primera generación de iPad Mini) en modo *portrait*:

```
@media only screen
and (min-device-width : 768px)
and (max-device-width : 1024px)
and (-webkit-min-device-pixel-ratio: 1)
```

```
{ /* Estilos  */}
```

Para iPad que ofrecen pantalla retina, el código sería el siguiente para el modo *portrait*:

```
@media only screen
and (min-device-width : 768px)
and (max-device-width : 1024px)
and (orientation : portrait)
and (-webkit-min-device-pixel-ratio: 2)
{ /* Estilos */ }
```

Es importante señalar que las características `-webkit-device-pixel-ratio` también pueden ser utilizadas con navegadores de otros fabricantes que utilicen WebKit como motor, como es el caso de iOS o BlackBerry.

Media Queries es Recomendación del W3C desde junio de 2012 (http://www.w3.org/TR/css3-mediaqueries/), y cuenta con un buen soporte en navegadores modernos, tanto en los *desktop* como en móviles.

El uso de Media Queries abre nuevas alternativas en el mundo del diseño Web. Responsive Web Design es un conjunto de técnicas que permiten que los diseñadores Web creen sitios y aplicaciones que se adapten a las características de la pantalla del usuario. El libro *Responsive Web Design*, de Ethan Marcotte, publicado originalmente en el año 2010, es un referente sobre este tema.

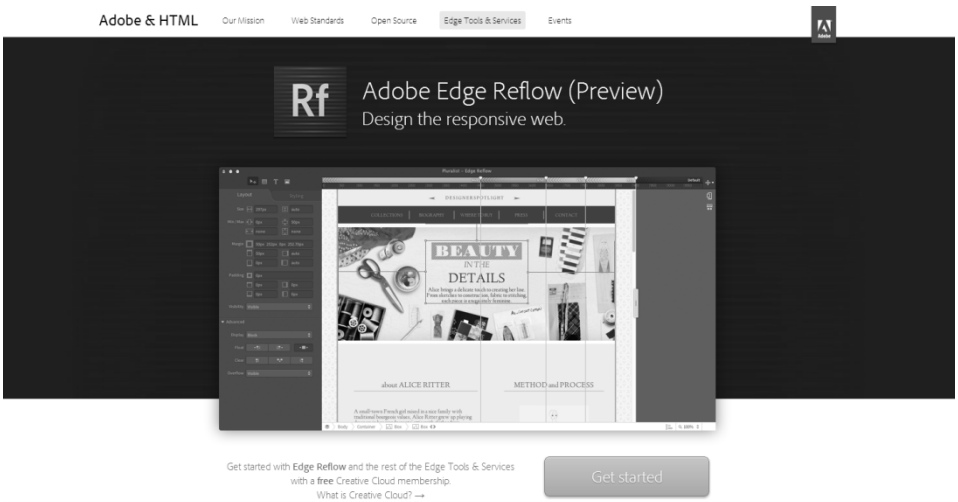

Fig. 3.17. Adobe Edge Reflow (http://html.adobe.com/edge/reflow/) es una herramienta enfocada en crear layouts adaptables a diferentes medios y pantallas.

Cajas Flexibles

Si alguna vez se han enfrentado al duro trabajo de tener que centrar cajas en modo vertical y horizontal y, además, tener la necesidad que se adapten a cambios de pantalla y proporción, seguramente se habrán sentido limitados con lo que les ofrecía hasta ahora CSS.

La realidad es que el modelo de cajas de CSS ha sido muy útil hasta aquí, y lo sigue siendo, pero la diversidad de medios y las exigencias del mercado actual han puesto en evidencia la necesidad de nuevas opciones para trabajar de una manera eficaz con sitios que necesitan adaptarse.

Una de las características avanzadas que se introduce en CSS3 es la posibilidad de trabajar con cajas flexibles, una alternativa que nos provee de soluciones que cambiarán la manera de maquetar sitios que necesitan adaptarse a diferentes medios.

CSS Flexible Box Layout Module nos brinda la posibilidad de trabajar con un modelo de cajas flexible optimizado que permite que los hijos del contenedor se ubiquen, ordenen y centren de una manera mucho más sencilla de la que teníamos hasta el momento.

Para comenzar a comprender cómo funciona este módulo debemos saber que la propiedad `display` ahora también soporta el valor `box`. Para trabajar con esta característica, se incorporan las propiedades `box-orient` (soporta los valores: `horizontal`, `vertical` o `inherit`), `box-pack` (soporta los valores: `start`, `end`, `cener` o `justify`), `box-align` (soporta los valores: `start`,

end, center, baseline o stretch), box-flex (un número entero), box-flex-group (un número entero), box-ordinal-group (un número entero), box-direction (un número entero) y box-lines (soporta los valores: single o multiple).

El soporte de este módulo aún no es total en todos los navegadores y requiere los prefijos –ms, -webkit y –moz, por compatibilidad

En el documento del W3C CSS Flexible Box Layout Module es posible acceder a mayor información sobre este módulo: http://www.w3.org/TR/css3-flexbox/.

Bordes

Como muchos de los lectores sabrán, el redondeado de una caja o de una imagen, hasta aquí, implicaba el uso de una herramienta de retoque para generar el resultado en un formato grafico (JPG, PNG o GIF).

Con la llegada de CSS3, esta característica ahora se puede resolver con gran facilidad desde los estilos. Esto brinda una mejora en el rendimiento y ofrece mayor flexibilidad, especialmente, cuando se trata de contendidos adaptables. La sintaxis básica de border-radius es la siguiente:

```
div{border-radius:20px;}
```

En el ejemplo citado, se aplica un redondeado uniforme de 20px a cada lado de una caja. Pero, también es posible trabajar sobre cada una de las equinas con valores diferentes:

```
div {
border-top-left-radius: 20px;
border-top-right-radius:10px;
border-bottom-right-radius:30px;
border-bottom-left-radius:20px;
}
```

Otra alternativa posible sería la de indicar dos valores para una misma esquina, y de esta forma modificar la forma del redondeado, por ejemplo:

```
border-top-left-radius: 20px 10px;
```

Otra característica nueva que introduce CSS3 es la posibilidad de aplicar imágenes a los bordes. Esto se puede lograr mediante la propiedad border-image que actúa como *shorthand* para establecer los valores de border-image-source (ruta y nombre de la imagen), border-image-slice (el desplazamiento de los bordes internos de la imagen), border-image-width (el ancho del borde de imagen), border-image-outset (la extensión del borde exterior) y border-image-repeat (repetición de la imagen de borde). Veamos un ejemplo a continuación:

```
border-image: url(borde.png) 27 round;
```

El mismo código lo repetimos con los prefijos para Webkit, Mozilla y Opera.

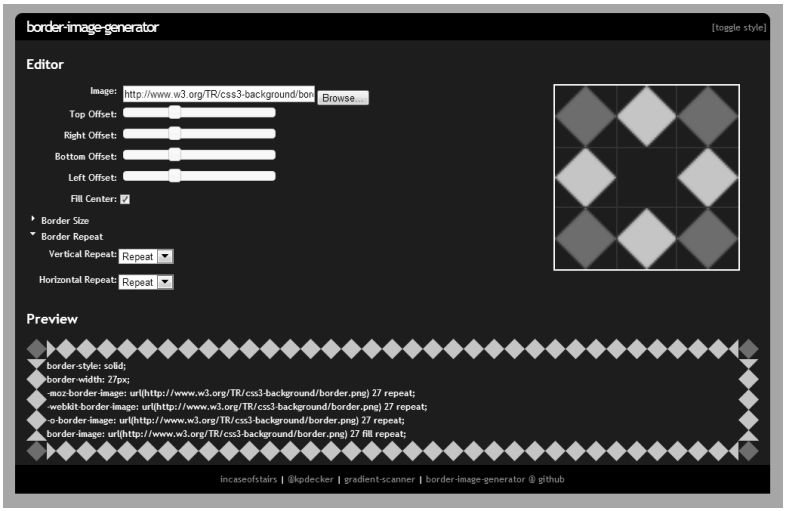

Fig. 3.18. Ingresando en este sitio (http://border-image.com), podremos acceder a la herramienta *Border-image-generator* que permite subir y ajustar una imagen como borde. Como resultado, ofrece el código CSS con todas las propiedades necesarias para incorporarlo en nuestro proyecto Web.

El documento de CSS3 que se refiere al trabajo con bordes se encuentra disponible en: http://www.w3.org/TR/css3-background/.

Sombras

Al igual que en el caso de los bordes redondeados, la posibilidad de aplicar una sombra a un contenidos, estaba, principalmente, resuelta mediante el uso de imágenes. Con CSS3 ahora podremos dar estilos que utilizan uno o varios niveles de sombra a cajas y texto. Esto último es muy importante, ya que el texto sigue siendo accesible y no hay necesidad de ubicarlo en una imagen.

La diferenciación que debemos realizar aquí es entre las sombras al texto (`text-shadow`) y las sombras en la caja (`box-shadow`).

Para aplicar sombra al texto, podremos indicar el desplazamiento de la sombra en el eje x, desplazamiento en el eje y, *blur* (opcional) y color. En el ejemplo a continuación, apreciamos un código aplicando a un párrafo una sombra azul:

```
p {text-shadow: 5px 5px 3px blue;}
```

Si deseamos colocar varios niveles de sombra a un texto, podemos hacerlo, separándolos mediante una coma, tal como podemos ver en el ejemplo que sigue:

```
p {text-shadow: 2px 2px 2px silver,
   5px 5px 3px black;}
```

Para trabajar las sombras de la caja contamos con la propiedad `box-shadow` y los valores que podremos pasarle son: desplazamiento de la sombra en el eje x, desplazamiento en el eje y, *blur* (opcional), *spread* (tamaño de la sombra, opcional), *inset* (si deseamos que sea una sombra interna en la caja, opcional) y el color. A continuación, veremos un ejemplo aplicado:

```
div {box-shadow: 15px 15px 8px black inset;}
```

De la misma manera que para el texto, las cajas también soportan varios niveles de sombra, separados por coma.

Colores y transparencias

Con CSS3 se introducen algunas novedades interesantes en el modulo de color, que ya ha llegado a ser recomendación W3C desde junio de 2011.

Para definir un color en CSS, podemos utilizar la palabra clave de su nombre en inglés (por ejemplo `red` para rojo o `blue` para azul) o bien su valor en hexadecimal para tener un abanico mayor de posibilidades.

A partir de CSS3, se agrega la posibilidad de especificar un color mediante su valor RGB (valores de rojo, verde y azul) o HSL (matiz, saturación y luminancia).

Quizás hasta aquí el lector no encuentre demasiadas ventajas por sobre lo que ya existía dentro de las opciones de color de CSS, pero si pensamos en la posibilidad de utilizar transparencias, los beneficios son muchos.

La propiedad `opacity` nos permitía definir la opacidad de un elemento. El problema de esta propiedad es que se hereda sobre los elementos hijos y, este efecto, no siempre es el deseado. La opacidad puede tener un valor de `0` a `1`, utilizando valores decimales para llegar al valor deseado.

CSS3 incorpora la opción de trabajar sobre el canal alfa como cuarto parámetro, tanto en RGBA (rojo, verde, azul y el canal alfa) como en HSLA (matiz, saturación, luminosidad y canal alfa).

Para RGBA, las tres primeras posiciones pueden recibir un valor desde `0` hasta `255` y el valor de alfa puede ser de `0` a `1` (con valores decimales). A continuación, veremos un ejemplo aplicado al color de un título:

```
h1 {color: rgba(100,150,280,0.7);}
```

En el caso de HSLA, el primer valor es numérico (de `0` a `360`), mientras que en los dos siguientes son porcentuales (`0%` a `100%`). El valor del cuarto parámetro, al

igual que con RGBA, es un número entre `0` y `1` (con valores decimales). Veamos un ejemplo aplicado como fondo de una caja:

```
div { background-color:
     hsla(150, 40%, 35%, 0.5);}
```

Para conocer más detalles sobre CSS Color Module Level 3, podemos ingresar en el sitio del W3C y leer el documento: http://www.w3.org/TR/css3-color/.

Fondos

Para el trabajo con fondos tenemos dos novedades muy interesantes en CSS3. La primera tiene que ver con la posibilidad de ajustar el tamaño del fondo y la segunda nos posibilita especificar múltiples imágenes para el fondo de una caja.

La propiedad `background-size` puede recibir los valores de ancho y alto (x e y) para ajustar la imagen de fondo. Estos valores pueden estar expresados en una medida de longitud o bien en un valor porcentual. También pueden aplicarse las palabras reservadas `cover` (escala la imagen para cubrir todo el fondo, aunque parte de la imagen no sea visible) o `contain` (ajusta la imagen para que pueda verse completa en el fondo).

Para trabajar con `background-size` debe estar especificada la imagen de fondo con `background-image`. Opcionalmente se puede establecer también si la imagen se repetirá o no, empleando la propiedad `background-repeat`.

La opción de múltiples imágenes de fondo llega para permitirnos en una misma caja el poder colocar más de un fondo. Esto se logra, separando los valores por coma, tanto el del nombre y ruta de la imagen como el de la posición, repetición y, si lo deseamos, el tamaño. Veamos un ejemplo a continuación, donde ubicamos dos imágenes, establecemos su posición, además de evitar que se repitan:

```
div {
width: 200px;
height: 200px;
background-image: url(personaje.png), url(bosque.png);
background-position: center top, left top;
background-repeat: no-repeat;
}
```

Como podemos ver en el código, en el caso de `background-repeat`, si el valor se aplica a todas las imágenes podemos especificarlo una sola vez (en el ejemplo, que no repita las imágenes).

Fuentes tipográficas

Durante varios lustros, el uso de tipografias en la Web estuvo limitado a lo que se conoce como *Web Safe Fonts* o *fuentes seguras* para utilizar en la Web, es decir, las que habitualmente tienen los usuarios instaladas en sus sistemas.

Mediante la característica @font-face ahora es posible incorporar fuentes que el usuario no tiene instaladas en sus sistema. Para esto, debemos subir las fuentes a un sitio de Internet (en los formatos apropiados) y crear la regla para cada una, empleando @font-face.

Es importante incorporar la fuente en cada uno de los formatos que requieran los navegadores compatibles: True Type Fonts (TTF), EOT fonts (EOT), WOFF Fonts (WOFF) y SVG Fonts (SVG).

En móviles, las fuentes embebidas se pueden utilizar en Safari para iOS 3.2 o superior, Android 2.2 o superior BlackBerry 7.0 o superior.

Veamos a continuación un ejemplo de declaración de @font-face:

```
@font-face
{
font-family: mi_fuente;
src: url('mifuente.ttf'),
      url('mifuente.woff'),
      url('mifuente.svg'),
url('mifuente.eot');
}
```

Luego para utilizarla, podemos aplicarla como vemos en el siguiente ejemplo sobre un párrafo, pero es igual para cualquier otro elemento:

```
p{
font-family: mi_fuente, Arial, Verdana, Sans-serif;
}
```

Si necesitamos convertir fuentes para obtener todos los formatos necesarios, podemos emplear la herramienta que nos brinda el sitio Web Font Squirrel (http://www.fontsquirrel.com/fontface/generator).

Al subir fuente a Internet para utilizarlas en un sitio, es importante tener en claro el tema licencia, ya que no todas las fuentes son libres para este uso.

Si buscamos una alternativa sencilla de aplicar, podemos probar las opciones que nos brinda Google Web Fonts (http://www.google.com/webfonts). Con más de 600 tipos disponibles, este servicio nos provee de una API para utilizar fuentes en

proyectos Web y también para impresión. El sitio nos permite elegir las fuentes que deseemos utilizar y nos provee del código necesario para integrarlo en nuestro proyecto, además permite descargar una colección.

En la sección denominada **About Google Web Fonts** :

(http://www.google.com/webfonts#AboutPlace:about), podemos leer todos los detalles de estas fuentes que se ofrecen como *Open Source Fonts*.

Múltiples columnas

Multi-column layout es un nuevo módulo de CSS3 que permite especificar que un contenido se distribuya en una o más columnas, indicando también el espacio y separaciones entre ellas.

Las principales propiedades que podemos emplear para esta característica son:

- `column-count`: permite especificar la cantidad de columnas en las cuales se distribuirá el contenido. El valor aplicable debe ser numérico.
- `column-gap`: se emplea para especificar el espacio de separación entre las columnas. Puede recibir un valor en una unidad de longitud (por ejemplo, en píxeles).
- `column-rule`: permite crear una línea de separación entre las columnas. Se puede emplear como un *shorthand* o trabajar con las propiedades de manera separada: `column-rule-width` (ancho de la línea), `column-rule-style` (estilo de la línea) y `column-rule-color` (color de la línea).

Es recomendable utilizar, además de la opción estándar, las alternativas con prefijos `-webkit` y `-moz` para lograr mayor compatibilidad.

Encontraremos más información acerca de CSS Multi-column Layout Module en http://www.w3.org/TR/css3-multicol/.

Transformaciones

CSS3 introduce la capacidad de realizar transformaciones a los elementos mediante estilos. Este tipo de funcionalidad, anteriormente, estaba limitada al uso de JavaScript o mediante *filters* en el caso de los navegadores de la familia de Internet Explorer.

A partir del nivel 3 de CSS, ahora es posible rotar, trasladar escalar, realizar perspectivas y aplicar deformaciones a los elementos, en los ejes: x, y, z.

La propiedad `transform` actúa como *shorthand* para establecer todas las transformaciones que deseemos a un elemento. Por compatibilidad *browser* se puede trabajar también con los prefijos `-ms` (para IE9), `-moz`, `-webkit` y `-o`.

Por medio de esta propiedad, CSS3 permite realizar transformaciones 2D y 3D. Para lograr esto podemos utilizar las características detalladas en la tabla que sigue a continuación, como valor para esta propiedad `transform`.

Transformación	Descripción de la función
rotate(ángulo)	Recibe como valor el ángulo de rotación.
rotate3d(x,y,z,ángulo)	Permite rotar en tres dimensiones. Para lograrlo recibe los valores de x, y, z y el ángulo. También es posible trabajar con cada uno por separado: rotateX(ángulo), rotateY(ángulo), rotateZ(ángulo).
scale(x,y)	Se aplica para escalar un elemento en los ejes x e y.
scale3d(x,y,z)	Se puede utilizar para escalar un elemento utilizando tres dimensiones. Puede trabajar también con los valores de manera separada: scaleX(x), scaleY(y), scaleZ(z).
translate(x,y)	Se utiliza para realizar la translación de un elemento en los ejes X e Y.
translate3d(x,y,z)	Permite realizar una translación en los tres ejes. Es posible realizar también esta operación con los tres ejes por separado usando: translateX(x), translateY(y), translateZ(z).
perspective(n)	Se puede emplear para realizar una perspectiva 3D.
skew(ángulo x,ángulo y)	Se utiliza para torcer una elemento en dos dimensiones (recibe el valor de los ángulos para x e y).
matrix(n,n,n,n,n,n)	Puede recibir 6 valores que definen una matriz para realizar una deformación en dos dimensiones.
Matrix3d(n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n)	Para el caso de matrices en tres dimensiones, se pueden pasar 16 valores separados por coma.

Fig. 3.19. Transformaciones que se pueden realizar utilizando CSS3.

Para comprender cómo funcionan estas características, veremos un código de ejemplo donde se le aplica una transformación a una caja. Combinando varias de las características aprendidas en este apartado, lograremos que cambie su forma. En primer lugar, en el cuerpo del documento tendremos una caja, como la siguiente:

```
<div>Ejemplo de transición con CSS3</div>
```

A la caja le aplicaremos las opciones de transformación de CSS3:

```
div
{
background:black;
color:white;
width:700px;
height:400px;
font-size:96px;
text-align:center;
transform: rotate(20deg) scale(0.723) skew(-38deg)
translate(182px);
-webkit-transform: rotate(20deg) scale(0.723) skew(-38deg)
translate(182px);
-moz-transform: rotate(20deg) scale(0.723) skew(-38deg)
translate(182px);
-o-transform: rotate(20deg) scale(0.723) skew(-38deg)
translate(182px);
-ms-transform: rotate(20deg) scale(0.723) skew(-38deg)
translate(182px);
}
```

Fig. 3.20. Resultado del ejemplo visto en el navegador.

Transiciones

Hasta la llegada de CSS3, no era posible realizar transiciones entre cambios de estado o acciones del usuario, salvo mediante el uso de JavaScript. La incorporación de esta característica desde las hojas de estilo, es un avance importante para brindar nuevas herramientas a los diseñadores y, a su vez, una opción que nos permite ganar en *performance*, al obviar el paso de JavaScript.

La propiedad `transition` actúa como un *shorthand* para establecer las características que adoptará la transición que deseamos realizar. También podremos trabajar con cada una de las características de la transición mediante las propiedades: `transition-property` (propiedad afectada a la transición, pueden indicarse varias o `all` para todas), `transition-duration` (duración de la transición, en segundos), `transition-timing-function` (velocidad o atenuación del efecto en la transición) y `transition-delay` (demora en el tiempo en que comienza la transición, en segundos).

Por razones de compatibilidad es recomendable utilizar los modificadores con el prefijo `-webkit`, `-moz` y `-o`. A continuación, veremos el ejemplo de estilo para que un enlace cambie de color al pasar el puntero del *mouse* por encima y, a su vez, haga este efecto con una transición de tres segundos.

```
a
{
color: blue;
transition:color 3s;
-moz-transition:color 3s;
-webkit-transition:color 3s;
-o-transition:color 3s;
}

a:hover
{
color: red;
}
```

Para conocer más sobre las posibilidades que ofrece CSS3 respecto a transiciones, es posible consultar el documento CSS Transitions, publicado por el W3C: http://www.w3.org/TR/css3-transitions/.

Animaciones

Llegamos aquí a uno de los aspectos más sorprendentes y revolucionarios que introduce CSS3. Si con la incorporación de transiciones podemos lograr efectos de gran calidad visual, las propiedades vinculadas con animaciones nos permiten resolver de manera nativa algunas situaciones que, anteriormente, solo eran posible mediante aplicaciones de terceros como, por ejemplo, Flash o Silverlight.

Con CSS3 se introduce la posibilidad de trabajar con cuadros clave (*keyframes*) que nos brindan la posibilidad de crear animaciones directamente desde las propiedades de los estilos, trabajando sobre el tiempo de duración y

transformaciones, entre otras opciones disponibles. A continuación, veremos cómo animar un cambio en el color de fondo de una caja:

```
@keyframes animacion
{
from {background: blue;}
to {background: green;}
}
```

Para mayor compatibilidad deberíamos utilizar los prefijos para el resto de los navegadores, de la siguiente forma:

```
@keyframes animacion
{
from {background: blue;}
to {background: green;}
}
@-webkit-keyframes animacion
{
from {background: blue;}
to {background: green;}
}
@-moz-keyframes animacion
{
from {background: blue;}
to {background: green;}
}
@-o-keyframes animacion
{
from {background: blue;}
to {background: green;}
}
```

La regla `@keyframes` cuenta con un nombre, en este caso `animacion`, el mismo que se debe definir en el elemento afectado, como veremos en el ejemplo a continuación:

```
div
```

```
{
width:400px;
height:400px;
background:blue;
animation:animacion 2s;
-moz-animation: animacion 2s;
-webkit-animation:animacion 2s;
 -o-animation:animacion 2s;
 }
```

A este ejemplo le aplicamos el nombre que definimos (`animacion`) y el tiempo en segundos (`2s`). Como podemos observar en el código, por razones de compatibilidad, se recomienda trabajar con prefijos para la propiedad `animation`.

La propiedad `animation` actúa como un s*horthand*, pero también contamos con varias propiedades para manejar de manera individual animaciones en CSS3.

Propiedad	Descripción
animation-name	Nombre de la animación
animation-delay	Tiempo de demora hasta que comience la animación. Se mide en segundos y, por defecto, su valor es cero.
animation-duration	Duración de la animación (en segundos).
animation-timing-function	Permite definir el tipo de animación cómo evolucionará en el tiempo de duración (por defecto su valor es *ease*)
animation-iteration-count	Permite establecer las veces que se repetirá la animación. Acepta un valor numérico o *infinite* (para reproducción continua). De manera predeterminada su valor es 1.
animation-direction	Brinda la posibilidad de que se indique si la animación se reproducirá de modo reverso. Por defecto no lo hace, y su valor es normal, pero es posible asignar el valor *alternate*.
animation-play-state	De manera predeterminada, las animaciones comienzan a reproducirse cuando cargan; su valor por defecto es *running*. Si deseamos que la animación esté pausada, se puede asignar a esta propiedad el valor *paused*.

Fig. 3.21. Propiedades de animación en CSS3.

Siguiendo con el ejemplo que comenzamos a animar en el código anterior, también podríamos trabajar con *keyframes* que cambiarán el estado del elemento en momentos prefijados. Entonces, sobre el mismo elemento que trabajamos

anteriormente, podríamos comenzar a marcar cuadros clave, en los cuales se modificará su trayectoria, si lo que deseamos es moverlo. Veamos el ejemplo a continuación sobre el mismo `div`, con el que trabajamos anteriormente:

```
@keyframes animacion
{     0%   {margin-left:0px; margin-top:0px;}
25%  {margin-left:200px; margin-top:0px;}
50%  {margin-left:200px; margin-top:200px;}
75%  {margin-left:0px; margin-top:200px;}
100% {margin-left:0px; margin-top:0px;}
}
@-moz-keyframes animacion
{     0%   {margin-left:0px; margin-top:0px;}
25%  {margin-left:200px; margin-top:0px;}
50%  {margin-left:200px; margin-top:200px;}
75%  {margin-left:0px; margin-top:200px;}
100% {margin-left:0px; margin-top:0px;}
}
@-webkit-keyframes animacion
{     0%   {margin-left:0px; margin-top:0px;}
25%  {margin-left:200px; margin-top:0px;}
50%  {margin-left:200px; margin-top:200px;}
75%  {margin-left:0px; margin-top:200px;}
100% {margin-left:0px; margin-top:0px;}
}
```

Las animaciones de CSS3 tienen muy buena compatibilidad con los navegadores móviles modernos, especialmente con los que utilizan WebKit (Android 2.2 o superior; iOS 3.2 o superior; y BlackBerry 7 o superior). En navegadores *desktop* encontraremos compatibilidad en Mozilla Firefox, Google Chrome, Apple Safari y Opera. Internet Explorer ofrece compatibilidad con esta característica a partir de la versión 10.

El documento CSS Animations (http://www.w3.org/TR/css3-animations/), publicado por el W3C contiene más información sobre cómo trabajar con animaciones desde las reglas de CSS.

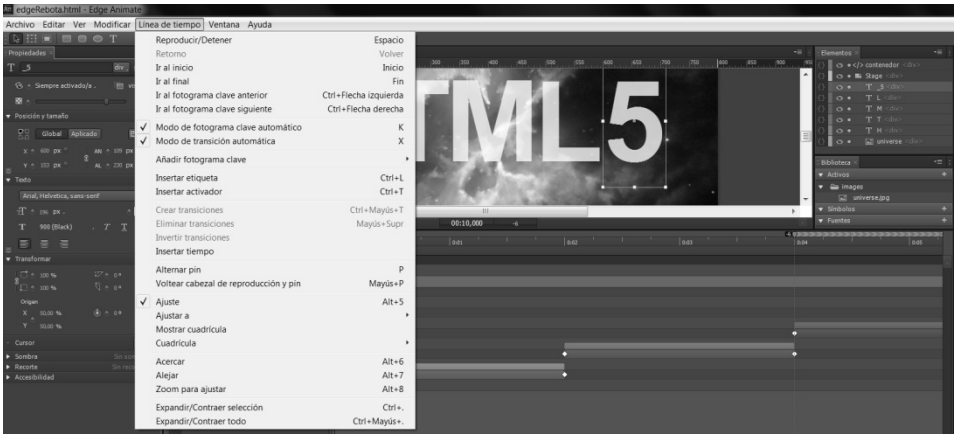

Fig. 3.22. Adobe Edge Animate (http://html.adobe.com/edge/animate/) es una herramienta que permite generar animaciones que combinan código HTML5, CSS3 y JavaScript.

Extensiones CSS de Webkit

Los navegadores basados en Webkit, tanto en ediciones para desktop como para móviles, pueden ofrecer algunas características que permiten aprovechar ventajas específicas. Algunos de ellos, como el caso de Safari, cuentan con soporte para extensiones de CSS que no forman parte del estándar que impulsa el W3C, pero que le ofrecen a los desarrolladores funcionalidades muy interesantes. Un ejemplo de esto, es la propiedad `-webkit-aspect-ratio`, característica que ya hemos mencionado cuando estudiamos las opciones que pueden trabajar en conjunto con Media Queries para permitirnos crear un diseño que se adapta el medio.

Safari para iOS cuenta con la propiedad `-webkit-box-reflect` para brindar una solución nativa a la necesidad de incorporar un reflejo en la caja. También es posible en este navegador incorporar una máscara para las imágenes, mediante la propiedad `-webkit-mask-image`. Otra característica que se introdujo a partir de la versión 5 de iOS es `-webkit-overflow-scrolling`, que soporta el valor `touch`, y ofrece una solución desde CSS a algunos problemas con el *scroll*.

Para el texto encontramos `-webkit-text-stroke`, que permite definir un trazo para el texto y `-webkit-text-fill-color`, para establecer el color de relleno del texto dentro del trazo.

Para campos de formulario, encontramos `-webkit-text-security`. Esta propiedad nos permite especificar la forma que tomarán los caracteres ocultos cuando se trata de un campo `<input>` del tipo `password`. Los valores que se le pueden especificar a este atributo son: `circle`, `disc`, `square` o `none`.

Safari 3.0 para iOS introduce `-webkit-user-select` para permitir establecer lo que el usuario puede seleccionar dentro de un elemento. Los valores que acepta son: `auto` (permite seleccionar contenido dentro del elemento), `text` (permite seleccionar texto dentro del elemento) o `none` (no permite realizar ninguna selección dentro del elemento).

Otra propiedad útiles `-webkit-appearance`. Usando esta característica, podremos habilitar la opción que permite modificar determinados elementos, que de manera predeterminada no es posible desde CSS (por ejemplo, algunos controles de formulario).

Al utilizar extensiones de Webkit, siempre debemos tener en cuenta que pueden no ser compatibles con todos los navegadores que se basen en este tipo de motor, por lo cual deberíamos verificar la documentación actualizada y realizar pruebas de compatibilidad para las plataformas que deseemos desarrollar.

CSS nivel 4

El tiempo es veloz y no se detiene en un estándar. Mientras algunos documentos del nivel 3 de CSS eran estandarizados, el nivel 4 de las hojas de estilos ya vio la luz. Aunque por el momento no hay gran soporte para estas características, nos ofrece una muy interesante ventana para observar lo que viene en este campo.

En septiembre de 2011, los selectores del nivel 3 de CSS llegaron al estado de Recomendación. Ese mismo año, comenzó la ruta del siguiente nivel para selectores. En el momento de escribir este libro, Selectors Level 4 se encontraba en *working draft*. Podemos seguir su evolución, ingresando en http://www.w3.org/TR/selectors4/.

Algo interesante ocurrirá con las imágenes. El borrador de CSS Image Values and Replaced Content Module Level 4 (http://dev.w3.org/csswg/css4-images/), incorpora la posibilidad de establecer opciones de reemplazo (ideal para diseños adaptables). Cabe señalar que al tiempo de escribir estas líneas, esta característica aún no se encontraba lista para implementar.

Eventos táctiles

La evolución de los dispositivos móviles ha tenido un importante desarrollo en los últimos años y las interfaces táctiles han ganado mucho terreno en este ámbito. Más allá de este dato, es importante recordar que no todos los dispositivos utilizan tecnologías táctiles, ya que, como hemos visto anteriormente, aun hoy existen móviles basados en teclado o tecnología de cursor.

En este apartado vamos a enfocarnos en los dispositivos que responden a eventos táctiles. En este sentido, debemos tener en cuenta que si bien los

dispositivos de interfaz táctil pueden interpretar la mayoría de los eventos del *mouse*, existen eventos táctiles pensados específicamente para interfaces de este tipo. Para comenzar vamos a conocer los principales *touch events*:

Evento	Descripción
touchstart	Ocurre cuando el usuario comienza a tocar con su dedo sobre el elemento.
touchmove	Ocurre cuando el usuario arrastra su dedo con el elemento.
touchend	Ocurre cuando el usuario termina la acción de mover su dedo el elemento.
touchenter	Se dispara en el momento en que el punto de contacto se mueve sobre el área interactiva del elemento.
touchleave	En este caso, ocurre cuando el punto de contacto es movido fuera de la zona interactiva del elemento en cuestión.
touchcancel	Ocurre cuando la acción es abortada (por ejemplo, por un suceso del dispositivo, como una llamada en un *smartphone*).

Fig. 3.23. Eventos táctiles.

Cada punto de contacto contiene tres listas que contienen objetos e información del toque:

- **touches**: contiene la lista de los dedos que se encuentran actualmente en sobre la pantalla.
- **targetTouches**: representa la lista de dedos que se encuentran sobre el elemento.
- **changedTouches**: cuenta con la lista de dedos vinculados con el evento actual.

Cada lista cuenta con un identificador para cada dedo utilizado en la acción (*identifier*), el elemento que es destino de la acción y datos de las coordenadas de pantalla (*target*), el lugar de la pantalla donde está ocurriendo la acción (*screen coordinates*), y el elipse que define la forma de cada dedo que está interactuando (*radius coordinates*, *rotationAngle*).

El W3C está trabajando en la estandarización de la documentación relacionada con eventos táctiles. Esta información se encuentra disponible en el documento: http://www.w3.org/TR/touch-events/.

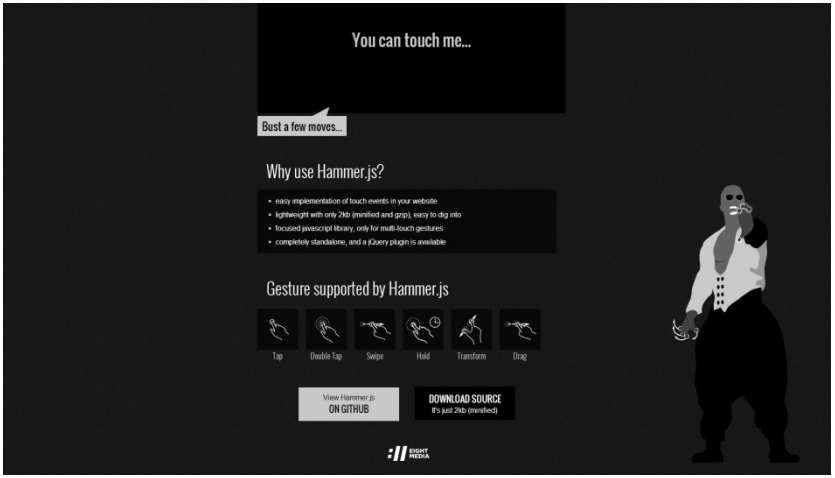

Fig. 3.24. Una librería de JavaScript que permite trabajar con gestos multitáctiles es Harmmer.js. Es liviana, fácil de integrar y puede funcionar de manera independiente o como *plugin* junto con jQuery.

Trabajo con los sensores del móvil

Los *smartphones* y *tablets* modernas, en la mayoría de los casos, cuentan con sensores que le permiten al dispositivo acceder a información sobre localización, movimiento, aceleración y rotación que pueda experimentar.

Estas características pueden ser utilizadas de diversas maneras, tanto en aplicaciones como en juegos, logrando en la actualidad resultados realmente sorprendentes.

Desde el punto de vista de la programación, en primer lugar, es importante arrancar por conocer la naturaleza de los cada uno de los sensores que puede estar disponible en un móvil, y a partir de este conocimiento, comenzar a imaginar e idear qué tipo de soluciones se pueden ofrecer, aprovechando estas características.

Al trabajar con Media Queries hemos visto que es posible saber la orientación del dispositivo y aplicar reglas de estilo sobre la base de esta información. En algunos casos, puede resultar útil detectar este cambio a raíz de eventos y actuar mediante JavaScript. Para esto, contamos con nuevos eventos del DOM, que nos permiten acceder a información de la orientación física del dispositivo. Esta especificación se encuentra en el documento del W3C titulado DeviceOrientation Event Specification, que se encuentra disponible en la dirección: http://www.w3.org/TR/orientation-event/.

Mediante el evento `deviceorientation`, accedemos a la información de la orientación que llega a través del navegador. En primer lugar, podremos detectar si esta característica es soportada, empleando:

`window.DeviceOrientationEvent`:

```
if (window.DeviceOrientationEvent) {
 console.log("DeviceOrientation soportado.");
}
```

Una vez que detectamos si el navegador es compatible con esta característica, podemos incorporar un *listener* en nuestro código para manejar el evento, como vemos en el código a continuación:

```
window.addEventListener('deviceorientation', devOrient,
false);
```

El evento DeviceOrientation retorna tres datos importantes: `alpha` (la rotación alrededor del eje Z), `beta` (la rotación en torno del eje X), y `gamma` (la rotación en torno al eje Y). A continuación, veremos un ejemplo de cómo obtenerlos mediante una función de JavaScript que llamaremos `devOrient`:

```
function devOrient(event) {
 var alpha = event.alpha;
 var beta = event.beta;
 var gamma = event.gamma;
 }
```

En el código anterior, asignamos a variables los valores de `alpha`, `beta` y `gamma`, para luego mostrarlos en pantalla o realizar alguna acción en base a ellos. En caso de no estar soportada la característica, devuelven el valor `null`.

Si lo que necesitamos hacer es correr un código cuando el dispositivo cambia de orientación, podremos utilizar `onorientationchange`, como vemos en el siguiente *script*:

```
window.onorientationchange = function() {
      if (orientation == 0){
//Código par el dispositivo en 0º
}
      else if ((orientation == 90) ||
(orientation == -90)){
//Código par el dispositivo en 90º o -90º
      }
```

```
      else if (orientation == 180){
//Código par el dispositivo en 180º
      }

      }
```

Mediante `devicemotion` podemos acceder a la aceleración, información que nos será provista en coordenadas cartesianas de los ejes x, y y z. Para verificar si se encuentra soportado utilizamos el siguiente *script*:

```
if (window.DeviceMotionEvent) {
// El dispositivo soporta Device Motion
}
```

A continuación, procedemos por crear el *listener*:

```
window.addEventListener('devicemotion',
aceleracion, false);
```

Al crear la función `aceleracion()` podremos obtener, mediante la propiedad `acceleration`, la información de cada uno de los ejes:

```
function aceleracion (event) {
  var acX = event.acceleration.x;
  var acY = event.acceleration.y;
  var acY = event.acceleration.z;
}
```

Si deseamos obtener dicha información, pero considerando los efectos de la fuerza de gravedad, podremos escribir:

```
function aceleracon (event) {
  var acGravX = event.accelerationIncludingGravity.x;
```

```
  var acGravY = event.accelerationIncludingGravity.y;

  var acGravY = event.accelerationIncludingGravity.z;

}
```

El W3C se encuentra trabajando sobre la API "Screen Orientation" (http://www.w3.org/TR/screen-orientation/), cuyo objetivo es ofrecer una interfaz para aplicaciones Web que necesiten saber el estado y los cambios en la orientación de la pantalla, además de fijar o bloquear la orientación de la pantalla en un estado.

En el próximo capítulo, nos enfocaremos en analizar las librerías y el desarrollo que nos ayudará en nuestro desarrollo.

Software y librerías para el desarrollo

4

Elección del software para el desarrollo

En el capítulo anterior, conocimos las ventajas que introduce HTML5 en combinación con CSS3 y JavaScript. Estas características marcan el pulso de una nueva revolución en el mundo del desarrollo Web.

Como hemos aprendido hasta aquí, esta revolución va mucho más allá del mundo Web, permitiéndonos crear una nueva generación de aplicaciones que pueden funcionar en diversos "ecosistemas".

En este capítulo, vamos a conocer el software que nos permitirá escribir el código de nuestra aplicación, y también analizaremos las librerías que nos ayudarán en nuestro proyecto. Además, veremos qué alternativas existen para realizar las tareas de *testing*, *debugging* y *profiling*. Ahora, comenzaremos por recorrer las mejores opciones disponibles para escribir y editar código.

Adobe Dreamweaver

Dreamweaver es uno de los editores WYSIWYG más conocidos para el desarrollo y diseño de proyectos Web. En diciembre de 1997, Macromedia lanzó la versión inicial de este producto para Mac. En marzo del año siguiente, se publicó la primera versión compatible con sistemas Windows.

Los productos de Macromedia son adquiridos posteriormente por Adobe, y comienza una nueva etapa en su comercialización. A partir del 2007, se publican las primeras versiones de Dreamweaver bajo el sello de Adobe.

Macromedia fue una empresa de desarrollo de software para diseño, edición de gráficos y multimedia. Nacida en 1992, y con sede en San Francisco (California, Estados Unidos), logró gran fama mundial por prestigiosos productos como Director, Flash, Dreamweaver y Fireworks, entre otros. Adobe Systems adquirió la empresa en el 2005 para integrar los principales productos de Macromedia en sus Suites.

Dreamweaver ofrece soporte para HTML5 desde la versión CS5.5. Cabe mencionar que también existen algunos *packs* de actualización para versiones anteriores, que permiten habilitar el marcado de etiquetas HTML5 en la vista de código. El listado de actualizaciones de este producto se encuentra disponible en la siguiente dirección URL de Adobe:

http://www.adobe.com/support/dreamweaver/downloads_updaters.html.

A partir de la versión CS6, se introducen numerosas mejoras para trabajar con HTML5, CSS3 y también características para facilitar el desarrollo destinado a plataformas móviles. Entre las mejoras e incorporaciones de esta versión se destacan:

- Uso de Fluid Grid Layout con CSS3.
- Paneles que facilitan la creación de transiciones con CSS3.
- Inserción rápida e inteligente de multimedia con HTML5 (audio y video).
- Compatibilidad con animaciones de Edge Animate.
- Rendimiento mejorado de la transferencia vía FTP.
- Actualización del panel de vista previa multipantalla.
- Compatibilidad mejorar y actualizada con jQuery Mobile y con PhoneGap.

Dreamweaver CC (lanzado en el 2013) suma el soporte para Edge Web Fonts, transiciones CSS3 e inserción más ágil de elementos HTML5, entre otras características

Para instalar Dreamweaver CC se requiere un equipo con sistema Windows 7 o Windows 8 o una Mac OS X v10.7 o superior. Para conocer más información sobre los requisitos de este producto, se puede ingresar en:

http://www.adobe.com/es/products/dreamweaver/tech-specs.html.

Sublime Text

Sublime Text es el nombre de un editor de código multipropósito que ha logrado, en el último tiempo, una muy buena adopción en el mundo del desarrollo Web.

Permite elegir la combinación de colores y personalizarla, cuenta con marcado de etiquetas y sintaxis para lenguajes Web con opciones de resaltado configurables. Ofrece selección múltiple, y también brinda un mini mapa de navegación que le permite saber siempre al desarrollador en qué lugar del código está ubicado. Otras

ventajas son el *Multi Layout*, las búsquedas dinámicas, y una cantidad considerable de atajos que facilitan el trabajo de los programadores.

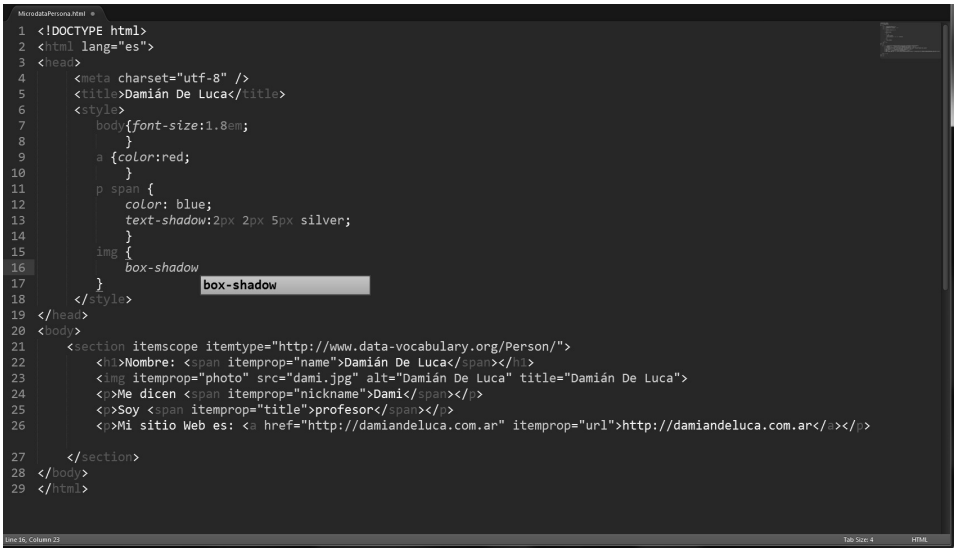

Fig. 4.1. Submile Text es una herramienta que ofrece marcado en color y ayuda en pantalla para escribir código HTML5 y CSS3.

Esta aplicación cuenta además con muchas opciones de personalización y puede ser expandido por medio de *plugins*.

Esta herramienta se destaca por ser multiplataforma, ya que cuenta con versiones para Windows, Linux y Mac OSX. Se puede descargar una versión de prueba, ingresando en el sitio Web http://www.sublimetext.com/.

Eclipse

Eclipse es un potente entorno de desarrollo integrado (IDE) que permite trabajar con diversos lenguajes de programación. Esta herramienta es muy popular, especialmente, para los desarrolladores JAVA. Esto queda demostrado en las estadísticas: dentro de los usuarios que eligen esta IDE, más del noventa por ciento programa en JAVA.

Si bien es una alternativa válida para escribir código empleando lenguajes de etiquetas, en nuestro proyecto aprovecharemos la versatilidad de Eclipse para poder empaquetar el contenido para generar el archivo compatible con dispositivos Android. Esto lo lograremos incluyendo algunos *plugins*.

Eclipse se distribuye bajo Licencia Pública Eclipse (EPL). Es código abierto y se conocen numerosas versiones adaptadas para diferentes necesidades. Se puede obtener ingresando en el sitio http://www.eclipse.org/.

Xcode

Para todos aquellos que desarrollan en el mundo de Mac OSX, Xcode es un aliado muy importante. Permite escribir, y también compilar código creado en lenguajes como Java, C, C++, Objetive-C y Objetive-C++.

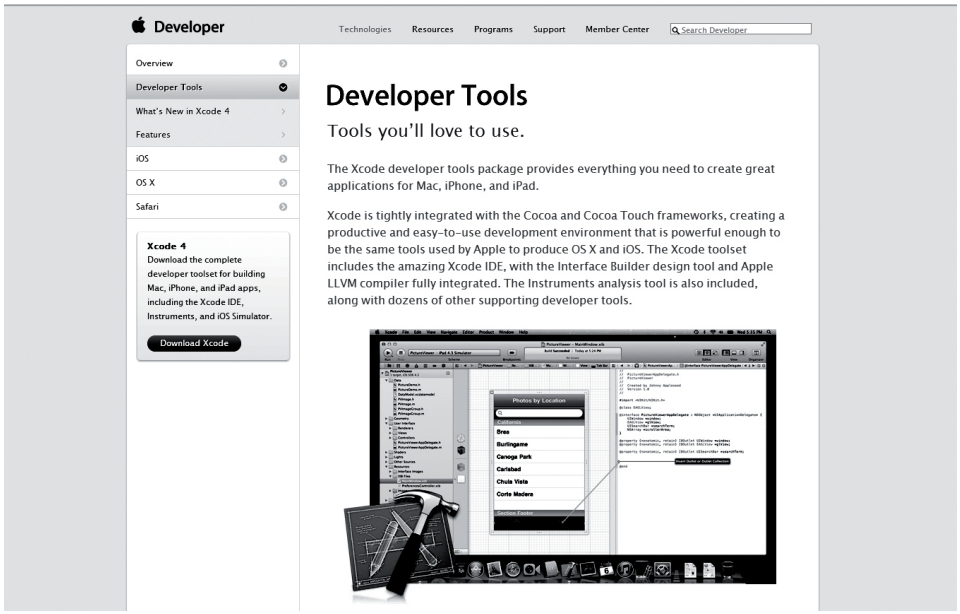

Fig. 4.2. Es posible encontrar información sobre Xcode en la sección *Developers Tools* de Apple: https://developer.apple.com/technologies/tools/.

Si bien en este libro no nos introduciremos en el mundo del desarrollo con código nativo para iOS, Xcode nos facilitará el trabajo al tomar nuestros proyectos creados con HTML5 y empaquetarlos con PhoneGap.

Vale la pena recordar que Xcode no cuenta con versiones para otras plataformas, como por ejemplo Microsoft Windows o Linux.

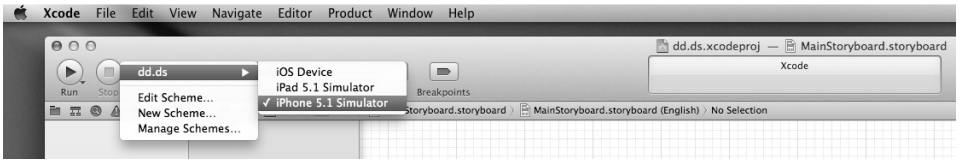

Fig. 4.3. Desde Xcode es posible acceder a los simuladores de iPhone y también de iPad.

Otras opciones

A pesar de que puedan existir algunas limitaciones de acuerdo con las plataformas elegidas, en la actualidad existe una gran variedad de opciones para que los desarrolladores puedan escoger las herramientas que utilizarán en sus proyectos.

Notepad++ es un editor de código gratuito (licencia GPL) que funciona en sistemas Windows. Por sus características y configuración es un programa simple de usar. Este software se presenta como una alternativa muy interesante para programadores que necesitan soporte para diversos lenguajes de programación. Notepad++ está disponible para su descarga, en el sitio Web http://notepad-plus-plus.org/.

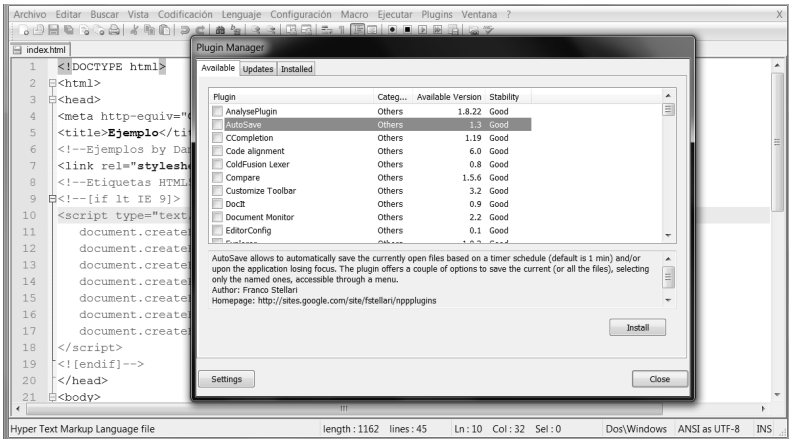

Fig. 4.4. Una de las opciones más valoradas de Notepad++ es la capacidad de extender sus características mediante la inclusión de *plugins*.

Entre las opciones multiplataforma se encuentra la alternativa que ofrece Bluefish. Esta herramienta se puede descargar desde el sitio http://bluefish.openoffice.nl/ y su licencia es GNU GPL.

PhoneGap - Apache Cordova

La historia cuenta que PhoneGap nació como un proyecto de una empresa llamada Nitobi. El objetivo del producto apuntaba a brindarle a desarrolladores con conocimientos de HTML5, CSS3, JavaScript y otras técnicas Web, la posibilidad de empaquetar sus creaciones para que luego pudieran correr de manera nativa en diversas plataformas móviles.

La idea tuvo una muy buena repercusión en el mundo del desarrollo, ya que con un mismo código fuente, y sin necesidad de aprender los lenguajes nativos de programación para cada plataforma, los desarrolladores podrían tener rápidamente un producto para distribuir, por ejemplo, en tiendas de aplicaciones móviles.

Estas aplicaciones, en muchos casos, son denominadas **híbridas**, ya que no se compilan en código nativo de cada entorno, sino que se ofrecen empaquetadas, aprovechando de esta forma el código de origen.

Luego del éxito obtenido por PhoneGap, y de que una importante comunidad de desarrolladores haya elegido este producto para sus proyectos, Adobe Systems decide adquirir Mitobi para incorporar a PhoneGap dentro de su cartera de productos e integrarlo en sus servicios.

Adobe acordó con Nitobi mantener una versión libre del producto, y es así como se donó el código base de PhoneGap a la Fundación Apache. En principio, el nombre que se eligió para ese proyecto fue Apache Callback, y posteriormente, se optó por Apache Cordova. De esta forma, PhoneGap queda como una distribución de Apache Cordova.

Adobe incorporó funcionalidades de PhoneGap en sus productos, y también en servicios que se pueden utilizar desde la nube, como el caso de PhoneGap Build.

PhoneGap Build

Si deseamos realizar el empaquetado de modo independiente de la plataforma de desarrollo, existe una alternativa que funciona en línea y que no requiere la instalación de aplicaciones en nuestro equipo: PhoneGap Build.

Para comenzar a utilizar esta herramienta, podremos ingresar en el sitio https://build.phonegap.com/ y acceder a la opción **Register**. Desde allí se podrá optar o bien por el plan gratuito (permite una aplicación privada) o por una alternativa paga (hasta 25 aplicaciones privadas). Una vez elegida la opción, el sistema nos mostrará los pasos para crear una nueva cuenta.

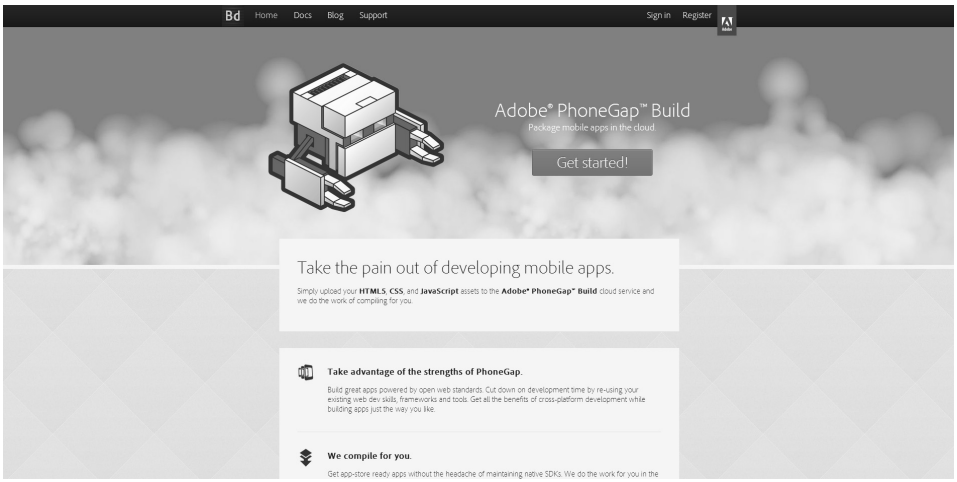

Fig. 4.5. Adobe PhoneGap Build permite acceder a un servicio en la nube para empaquetar aplicaciones sin necesidad de instalar plugins adicionales en el equipo. Se requiere registración para comenzar a utilizarlo.

Una vez dentro de PhoneGap Build, podremos administrar nuestros proyectos y realizar el empaquetado para las diversas plataformas móviles que deseemos. Para transferir los archivos de nuestro proyecto, tendremos la posibilidad de subir un archivo comprimido (.zip) o conectar a un repositorio de Github.

Fig. 4.6. Las opciones de transferencia de los archivos del proyecto son simples y permiten elegir entre las pestañas *open-source* o *private*.

Cada uno de los proyectos que utilicemos está visible y disponible en nuestro panel de administración de PhoneGap Build. Desde allí podremos ver: identificador de la aplicación (*App ID*), versión de la aplicación, versión de PhoneGap que se ha utilizado, y la fecha que indica la última vez que se realizó el Build. Esta ventana puede configurarse en idioma inglés o francés.

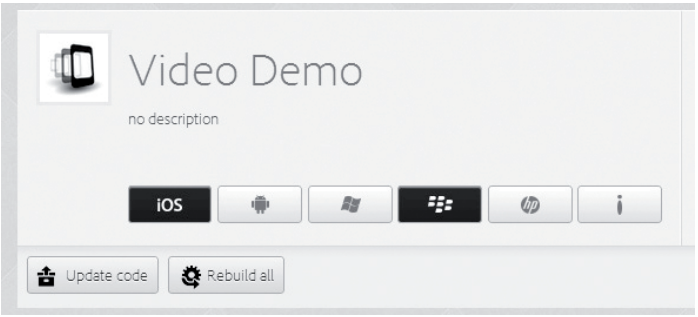

Fig. 4.7. PhoneGap Build ofrece un botón para realizar de manera rápida y sencilla una actualización del código y otro para realizar un *Rebuild*. Algunas plataformas móviles pueden requerir firmas o credenciales para permitir el proceso. Se mostrará en color rojo en caso de error o cuando no se hayan subido los archivos correspondientes.

PhoneGap recibe actualizaciones de manera frecuente, y siempre hay una versión en estado de desarrollo. Por estas razones, resulta muy importante estar al tanto de todas las novedades en cuanto a sus características y procedimientos.

Para más información sobre cómo configurar y preparar nuestros proyectos para ser empaquetados en línea con PhoneGap Build, podemos ingresar en la dirección Web: https://build.phonegap.com/docs/preparing-your-app.

Utilizar PhoneGap desde Adobe Dreamweaver

Desde la versión CS6 de Adobe Dreamweaver se puede utilizar la opción de acceder a las características de PhoneGap ingresando a través del menú Sitio. Desde allí se puede acceder a la opción Servicio PhoneGap Build.

Cuando accedemos a esta ventana, en primer lugar, hay que ir a la opción de configuración e indicar las rutas de acceso a los SDK utilizados. Vale la pena destacar que la opción de iOS solo estará disponible en equipos con Mac OSX, no así en sistemas operativos basados en Windows.

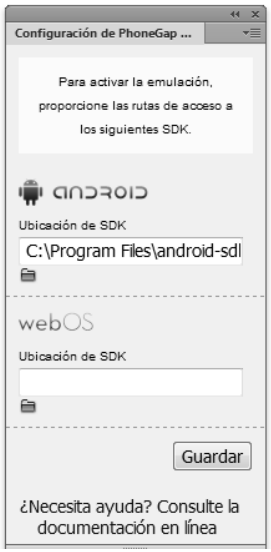

Fig. 4.8. Si existen dudas para configurar las rutas de los SDK disponibles, Adobe ofrece ayuda en línea y documentación.

Una vez configuradas las rutas a los SDK ya es posible utilizar PhoneGap Build desde Dreamweaver. También desde el menú **Sitio**, accedemos a **Servicio PhoneGap Build** y, luego, nuevamente hay que elegir la opción denominada **Servicio PhoneGap Build**.

Fig. 4.9. Al acceder a PhoneGap Build desde Dreamweaver, debemos ingresar la dirección de correo y la contraseña del servicio. Luego podremos seguir los pasos para la publicación.

Descargar PhoneGap para utilizar desde otros entornos

Si optamos por realizar el empaquetado desde otros entornos de desarrollo, la dirección Web que nos permite obtener el paquete de PhoneGap para trabajar desde nuestra computadora es: http://phonegap.com/download. Allí encontraremos disponibles las últimas versiones publicadas de PhoneGap.

El paquete comprimido que se obtiene de la sección de descarga nos ofrece documentación, ejemplos, y los archivos necesarios para crear aplicaciones compatibles con dispositivos que utilicen sistemas Android, iOS, BlakBerry, Bada, WebOS, Windows Phone 7 o Windows 8, entre otras plataformas.

PhoneGap puede utilizarse desde línea de comando o se puede agregar para trabajar desde algunos entornos de desarrollo. Desde la versión 3 de PhoneGap (mediados de 2013) es posible realizar la instalación desde la línea de comando empleando el soporte que ofrece Node.js (http://nodejs.org/), siguiendo el detalle que se explica en: http://phonegap.com/install/.

A continuación, veremos cómo integrar PhoneGap con Eclipse y Xcode; y más adelante en el libro (en el capítulo 6), aprenderemos cómo empaquetar nuestro proyecto una vez que se encuentre listo.

Incorporar PhoneGap en Eclipse

Eclipse es un IDE que se destaca por su flexibilidad y capacidad para extenderse mediante agregados. Entre las alternativas de desarrollo que brinda este software, como ya hemos mencionado, se destaca la posibilidad de crear aplicaciones para

Android. Esto puede lograrse directamente trabajando desde código nativo o bien con el uso de PhoneGap, posibilidad alternativa que nos permitirá empaquetar el contenido de un proyecto desarrollado con lenguajes Web (HTML, CSS y JavaScript, entre otros).

Toda la información necesaria para realizar la instalación de PhoneGap en Eclipse se encuentra en la sección de documentación de Apache Cordova, disponible en http://docs.phonegap.com. Las guías para cada plataforma se encuentran publicadas en la sección **Getting Started Guides**.

La instalación tiene algunas variantes, dependiendo de la versión de PhoneGap que se utilice en el desarrollo. Por este motivo, es importante verificar que la guía que estamos siguiendo corresponde a la versión de PhoneGap que vamos a utilizar.

Para comenzar, ante todo, es fundamental contar con Eclipse 3.4 (se recomienda tener la versión 3.6.2 o superior). Este software se puede obtener ingresando en la dirección web: http://www.eclipse.org/downloads/.

A continuación, debemos descargar el SDK de Android y asegurarnos que tenemos todo lo necesario para instalarlo en nuestro equipo.

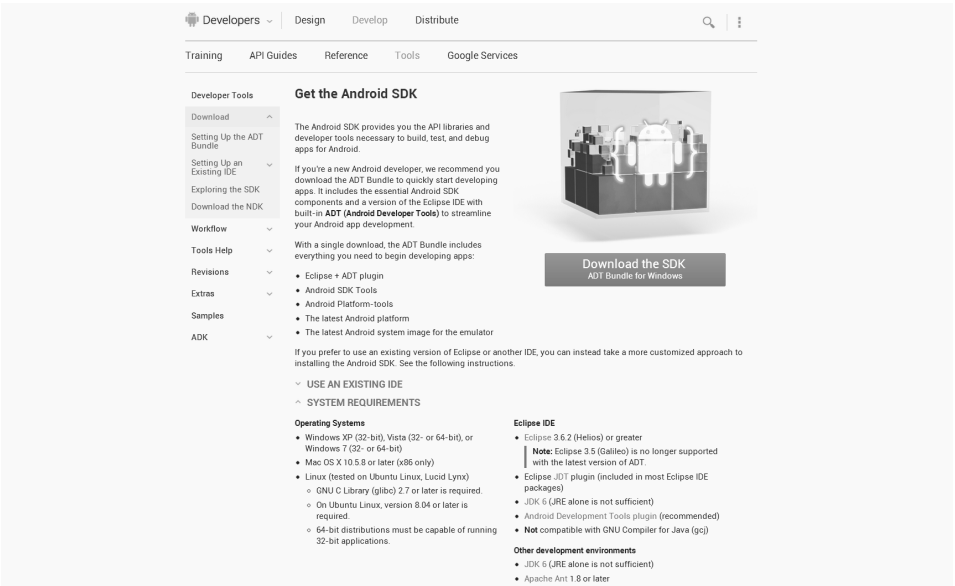

Fig. 4.10. Android SDK cuenta con versiones para Windows, Mac OSX y Linux. Requiere una versión moderna de Eclipse (se recomienda 3.6.2 o superior) y JDK 6 o superior. Los enlaces de descarga de todo el software necesario se encuentran en el sitio Web: http://developer.android.com/sdk/index.html.

Una vez completados los pasos detallados anteriormente, será el momento de descargar el *Android Development Tools* (ADT), que se encuentra disponible en:

http://developer.android.com/tools/sdk/eclipse-adt.html#installing. Este *plugin* para Eclipse nos permite integrar el entorno para desarrollar y crear aplicaciones Android. La instalación y configuración del *plugin* para Eclipse se detalla en el documento http://developer.android.com/sdk/installing/installing-adt.html.

Luego será el momento de extraer la carpeta `lib/android` del archivo comprimido, descargado de PhoneGap Cordova (http://phonegap.com/download). Para efectuar este proceso, se recomienda seguir los pasos indicados en la guía **Getting Started with Android**, que se ubica en **Getting Started Guides** , en la documentación suministrada para Phonegap (http://docs.phonegap.com). En ese documento, es importante realizar los pasos necesarios para la configuración de las rutas de variables de entorno en Microsoft Windows (**Setup your PATH environment variable on Windows**) así como seguir la guía para configurar un nuevo proyecto (**Setup New Project**).

Una vez efectuados estos procesos, ya estaremos listos para utilizar las opciones integradas de PhoneGap dentro de Eclipse. Más adelante en el libro, veremos cómo proceder para configurar nuestro proyecto y empaquetarlo con PhoneGap.

Al tiempo de escribir este libro se lanzaron las versiones preview release de Android Studio, un entorno de desarrollo enfocado en Android. Este entorno se encuentra disponible para Windows, Linux y Mac OSX y se puede obtener ingresando en: http://developer.android.com/sdk/installing/studio.html.

Incorporar PhoneGap en Xcode

Xcode es una herramienta que posibilita realizar desarrollos nativos para iOS y que, además, nos permite una fácil integración para generar proyectos con PhoneGap.

Los requisitos para la instalación de este paquete son: un equipo con Mac OSX 10.7 (o superior), Xcode 4.5 (o superior) y las herramientas de línea de comando de Xcode (Xcode Command Line Tools).

Dentro del archivo comprimido descargado del sitio de PhoneGap (http://phonegap.com/download), se puede extraer la carpeta destinada a iOS (`lib/ios`).

Las instrucciones para instalar este paquete se encuentran en la documentación de Apache Cordova (http://docs.phonegap.com/), en la sección **Getting Started Guides**. En la siguiente pantalla, hay que elegir **Getting Started with iOS**. Allí hay que seguir los pasos para instalar CordovaLib (**Install CordovaLib**) y, luego, crear un nuevo proyecto (**Create a New Project**). En ese documento podremos ver cómo configurar el simulador y el dispositivo para llevar a cabo nuestras pruebas.

Incorporar PhoneGap en otros entornos

En los apartados anteriores nos detuvimos en la configuración de PhoneGap para desarrollos basados en Android (utilizando Eclipse), y para iOS (trabajando con Xcode).

Otras opciones de desarrollo y guías disponibles en la sección **Getting Started Guides** de PhoneGap (http://docs.phonegap.com/) son:

- **Getting Started with Windows Phone:** es necesario tener un sistema operativo Microsoft Windows Vista (SP2) o superior y el SDK de Windows Phone (http://www.microsoft.com/en-us/download/windowsphone.aspx). Con Visual Studio Express for Windows Phone es posible manejar los proyectos y trabajar con plantillas predefinidas para simplificar nuestra labor.
- **Getting Started with Windows 8**: es necesario contar con Microsoft Windows 8, y con Visual Studio 2012 Professional (o superior) o Visual Studio 2012 Express para Windows 8.
- **Getting Started with Bada**: es necesario disponer de un sistema Microsoft Windows y el SDK de Bada (http://developer.bada.com/).
- **Getting Started with Tizen**: se puede desarrollar para esta plataforma desde Microsoft Windows o Ubuntu Linux, utilizando Eclipse. Los proyectos pueden iniciarse desde plantillas, y podremos visualizarlos previamente en el simulador o en el dispositivo.
- **Getting Started with WebOS**: el desarrollo para esta plataforma puede realizarse desde Windows, Linux o Mac OSX. Es necesario tener instalado VirtualBox (https://www.virtualbox.org/) y el WebOS SDK (https://developer.palm.com/). Si utilizamos Windows debemos asegurarnos de instalar cygwin SDK. En Mac OSX se necesita contar con Xcode y Command Line Tools parar XCode.
- **Getting Started with Blackberry**: es posible trabajar con Microsoft Windows XP (o superior) o Mac OSX 10.6.4 (o superior). En Windows se requiere Oracle JDK (http://www.oracle.com/technetwork/java/javase/downloads/index.html#jdk) y Apache Ant (http://ant.apache.org/bindownload.cgi). En Mac OSX es importante verificar que esté instalado Java en el sistema. Hay que instalar los SDK, según la plataforma para la cual vamos a desarrollar (BlackBerry 10, BlackBerry PlayBook, BlackBerry OS 7 o anteriores). Además de la opción que nos brinda PhoneGap, también es posible desarrollar para BlackBerry, utilizando HTML5 WebWorks de una manera independiente, como veremos en el próximo apartado.

Para cada una de estas opciones de desarrollo, debemos descargar el archivo comprimido de PhoneGap que incluye las librerías necesarias para cada plataforma (http://phonegap.com/download). También deberemos tener las firmas digitales y los certificados para los sistemas móviles que lo requieran.

BlackBerry WebWorks

Research in Motion (RIM) ha decidido apostar fuerte a una nueva generación de dispositivos, y en conjunto con las novedades de hardware y software ha renovado las posibilidades con las que cuentan los desarrolladores. Con **Cascades** (https://developer.blackberry.com/cascades/) los programadores encuentran una nueva manera de crear aplicaciones nativas. Por otra parte, **WebWorks** y **bbUI** son opciones muy interesantes para aquellos desarrolladores que vienen del mundo Web y desean crear apps para BlackBerry, empleando las herramientas que les resultan naturales para el desarrollo de Web apps.

Mediante WebWorks se introduce la posibilidad de empaquetar desarrollos realizados en lenguajes Web, aprovechando especialmente las ventajas que incorpora HTML5. De esta manera, podremos crear aplicaciones que pueden instalarse como nativas, y así distribuirse en las tiendas. Con esta tecnología podremos tener empaquetado todo nuestro proyecto en un archivo para dispositivos BlackBerry y también acceder a recursos que se encuentren en un servidor Web.

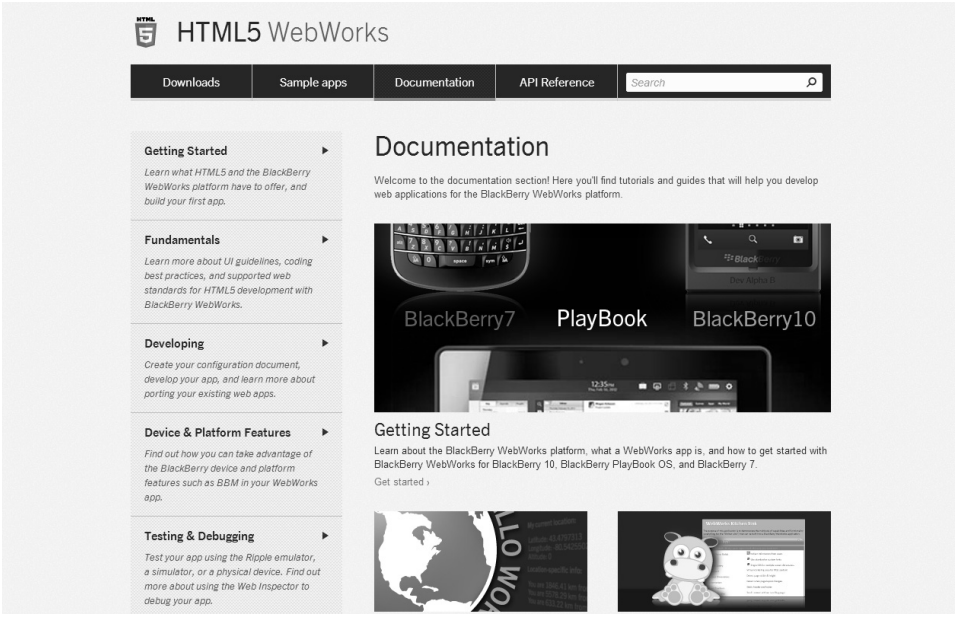

Fig. 4.11. BlackBerry ofrece información, documentación y varias guías sobre el uso de WebWorks en https://developer.blackberry.com/html5/documentation.

Con WebWorks es posible empaquetar aplicaciones que funcionarán en dispositivos BlackBerry 7, BlackBerry 10 y también PlayBook. El SDK para Windows y Mac OSX está disponible en: https://developer.blackberry.com/html5/download. A través de

ese enlace también es posible acceder a las opciones de descarga de Ripple Emulator (para Google Chrome) y BlackBerry 10 Dev Simulator.

La comunidad también ha realizado adaptaciones para poder utilizar WebWorks desde sistemas Linux, ejecutando archivos JavaScript con node.js (http://nodejs.org/) corriendo en el equipo local del desarrollador.

Por su parte, bbUI es un *framework* de JavaScript que nos facilitará la creación de aplicaciones con un "*look* nativo" de BlackBerry.

bbUI se puede obtener de manera gratuita en:

https://github.com/blackberry/bbUI.js/.

Si descargamos el archivo comprimido, que contiene todas las características, encontraremos ejemplos que emplean código HTML, CSS, JavaScript e incorporan imágenes para completar la posibilidad de componer la apariencia de nuestra aplicación. Es importante destacar que bbUI es compatible, y nos permite integrar características de etiquetas y API de HTML5.

Hay que subrayar que no es imprescindible la utilización de bbUI para crear aplicaciones BlackBerry que se empaqueten con WebWorks, ya que pueden utilizarse otras librerías, incluso trabajar directamente con HTML, CSS y JavaScript.

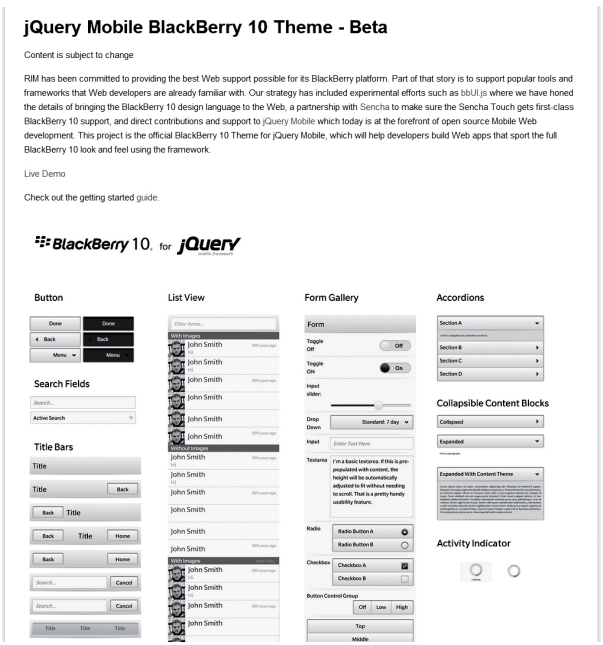

Fig. 4.12. Quienes tienen experiencia con jQuery Mobile pueden trabajar con esta librería y agregar un tema llamado jQuery Mobile BlackBerry 10 Theme. Este tema se encuentra disponible en:

https://github.com/blackberry/jQueryMobile-BB10-Theme.

Como ya hemos adelantado, en el capítulo 6, aprenderemos cómo empaquetar nuestra aplicación; y en el capítulo 7, veremos todo aquello que se relaciona con la distribución en las tiendas *online*.

Librerías para el desarrollo

Cuando nos enfrentamos a un desarrollo, hay varios factores que inciden en nuestro planteo. Uno de ellos es el tiempo del que disponemos para tener listo el trabajo. Aprovechar las ventajas que nos ofrecen diversas librerías nos puede ayudar a acelerar estos tiempos. Para ser claros en este concepto: no es necesario crear la rueda cada vez. Existen muchas librerías que nos ayudarán a lograrlo con características ya definidas.

Lo importante a evaluar en el caso de incorporar librerías en nuestro desarrollo es que sean las adecuadas para que no produzcan una disminución del rendimiento de nuestra aplicación. Cuando se incorporan varias librerías diferentes, debemos tener en cuenta que también pueden surgir incompatibilidades. Esto quiere decir, en palabras sencillas, que no podemos descansar todo nuestro desarrollo sobre librerías que, a simple vista, pueden parecer la solución mágica de todos nuestros problemas. O al menos nuestros problemas de desarrollo.

Por eso conviene estudiar en profundidad, por un lado, los beneficios y, por otra parte, los contra que nos puede provocar utilizar una u otra librería. En conclusión: deberemos tratar de elegir la librería que sea más eficaz y adecuada para nuestro proyecto.

jQuery

jQuery es una de las librerías de JavaScripts más conocidas y difundidas en el mundo del desarrollo Web. Fue creada por John Resign, y lanzada en el 2006. Hasta la fecha cuenta con numerosas actualizaciones y una gran comunidad que ha desarrollado tutoriales, documentación y *plugins* para potenciar su uso. Es una librería que se distribuye bajo licencia MIT y GPL, ya que es software libre y de código abierto.

Podemos descargar de manera gratuita jQuery desde su sitio Web: http://jquery.com/. Allí encontramos una versión para desarrollo (*Development*) y otra opción lista para incluir en nuestros proyectos y comenzar a utilizarla (*Production*). La versión de desarrollo es ideal para aquellos que cuentan con experiencia y desean crear *plugins* o ampliar las funcionalidades de esta librería. La versión *Production* es mucho más liviana y cuenta con todas las funcionalidades que necesitamos para nuestro proyecto. Por este motivo, será en este caso la elegida para avanzar en nuestro desarrollo.

Comenzar a utilizar jQuery

Este libro no pretende ser una guía extensa de uso de jQuery, existen otros manuales que cumplen muy bien con esa misión. Pero en este apartado veremos las nociones que necesitamos conocer para comenzar a utilizar esta librería en nuestro proyecto.

El primer paso es descargar jQuery y ubicarlo en la carpeta de nuestro proyecto. Todas las versiones de jQuery se encuentran disponibles en: http://code.jquery.com/. Esta librería también puede ser utilizada desde un CDN (Content delivery network): http://jquery.com/download/.

Luego deberemos ubicar en la cabecera, de cada documento que contenga código que utilice jQuery, el vínculo a la librería de la siguiente forma:

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

En el código anterior, el nombre del archivo y su ruta pueden variar dependiendo de la ubicación y de la versión utilizada.

Una vez que jQuery está colocado en nuestro código, podremos comenzar a utilizarlo cuando el documento esté listo, para esto podremos utilizar el siguiente código:

```
$(document).ready(function(){
   // Colocar el código aquí
 });
```

En el código podemos ver que para invocar a jQuery utilizamos el símbolo `$`. En lugar del comentario, ubicado en la segunda línea, deberemos incorporar el código de lo que deseamos programar utilizando jQuery.
De manera básica podremos decir que la mecánica de jQuery funciona de la siguiente manera: `$(elemento).evento(función/parametro);`

En el lugar del `elemento` podremos indicar cualquier elemento de nuestro documento HTML, mediante el selector (por ejemplo: id, clase o nombre del elemento). También podremos indicar el propio documento (`document`) como hemos visto anteriormente. Encontramos la lista de selectores para jQuery en el siguiente enlace: http://api.jquery.com/category/selectors/.

Al referirnos al `evento`, deberemos indicar el método que se relaciona, por lo general, con una acción del usuario (por ejemplo: un clic con el mouse o cuando se hace *scroll*) o algo que ocurre en el tiempo de ejecución del navegador (por ejemplo: cuando carga el documento o cuando se dispara un error). La lista de eventos disponibles para jQuery se encuentra en: http://api.jquery.com/category/Events/. Más adelante, en el libro, aprenderemos como adjuntar eventos táctiles para interactuar con nuestra aplicación en dispositivos táctiles y/o multitáctiles.

Aquí tenemos un ejemplo en el cual se imprimirá un mensaje, en la consola del navegador, cuando el usuario realice un clic sobre un elemento cuya id es `boton`.

```
$("#boton").click(function() {
console.log("Mensaje");
});
```

Los usos de jQuery pueden ser muy variados y su potencia es muy amplia, algunas de las principales ventajas que brinda jQuery para un desarrollo son:

- Manipulación del árbol del DOM (http://api.jquery.com/category/manipulation/).
- Intercambio de datos mediante AJAX (http://api.jquery.com/category/ajax/).
- Creación de efectos y animaciones (http://api.jquery.com/category/effects/).
- Obtener dimensiones de elementos (http://api.jquery.com/category/dimensions/).
- Trabajo con formularios (http://api.jquery.com/category/forms/).
- Compatibilidad crossbrowser (http://docs.jquery.com/Browser_Compatibility).

Para aquellas personas que deseen conocer la documentación completa de jQuery, es posible ingresar en la dirección http://docs.jquery.com/.

Quienes necesiten ampliar sus conocimientos acerca de todo lo que permite realizar AJAX, y el uso de jQuery, pueden buscar el libro titulado *AJAX - Web 2.0 Con jQuery Para Profesionales*, escrito por Maximiliano Firtman y distribuido por esta misma casa editorial.

Existe una gran comunidad alrededor de jQuery; y por este motivo, es posible encontrar una gran variedad de librerías y *plugins* que extienden y facilitan en gran medida el uso de este popular *framework*.

Detección del dispositivo

Como ya hemos podido comprobar en el capítulo 2, mediante el uso de las técnicas relacionadas con Responsive Web Design, podemos adaptar un contenido al medio. En el capítulo 3, hemos analizado lo que ofrece Media Queries para lograr con CSS que una estructura HTML se ajuste a las dimensiones del dispositivo. Sin embargo, si vamos a diferenciar documentos, según las características del dispositivo, deberemos optar por otras alternativas.

Existen diferentes técnicas y tecnologías que nos permiten conocer datos sobre el navegador del usuario. Entre las características que puede informar, encontramos el tipo de navegador, su motor, su versión y también el sistema operativo desde el cual está corriendo.

Algunos lenguajes y tecnologías que nos permiten realizar la detección son: JavaScript, PHP, ASP, ASP.NET, Pyton y JSP. También podremos encontrar opciones empleando otras plataformas y tecnologías; pero, en nuestro caso, nos enfocaremos en la detección del lado cliente mediante JavaScript.

Si trabajamos con jQuery podríamos detectar el tamaño de la pantalla y redireccionar a diferentes lugares de la siguiente manera:

```
$(document).ready(function() {
var anchoviewport = $(window).width();
if (anchoviewport <= 480){
window.location = "http://www.damiandeluca.com.ar/movil";
}
else{
window.location = "http://www.damiandeluca.com.ar/";}
});
```

El código, en primer lugar, mide el ancho del *viewport* del navegador, y lo coloca en una variable. Luego, realiza una evaluación, y si el dato es menor o igual a 480 (medida en píxeles) envía a una versión móvil del sitio; en caso contrario, lo deriva a la dirección habitual del sitio (utilizando `window.location`).
Con jQuery, también es posible detectar el alto de un elemento con el método `height()`.

Un camino más efectivo, y utilizado para detección de móviles, es el que nos permite leer y analizar el **User Agent** que informa el navegador. Esto puede lograrse, pasando por el método `test()` el valor de `navigator.userAgent`.

Veamos un ejemplo en el cual jQuery puede detectar los dispositivos móviles Android, BlackBerry y los desarrollados por Apple:

```
if( /Android|iPhone|iPad|iPod|BlackBerry/i.test
(navigator.userAgent) ) {
 // Colocar aquí la acción a realizar;
}
```

Para ser aún más eficaces en la detección, deberíamos establecer un mayor número de navegadores de sistemas móviles.
En http://detectmobilebrowsers.com/ encontraremos un conjunto de opciones Open Source para detección de móviles. Al acceder nos indicará si el navegador desde el que estamos accediendo es de un móvil, y nos mostrará el User Agent que informa. También nos brindará la posibilidad de descargar *scripts* de detección de

móviles para Apache, ASP, ASP.NET, ColdFusion, C#, IIS, JSP, JavaScript, jQuery, Lasso nginx, node.js PHP, Perl, Python y Rails.

Detección de características compatibles

Cuando pensamos en la compatibilidad que ofrecen los navegadores, debemos saber que no encontraremos, al menos por el momento, un navegador que ofrezca el cien por ciento de compatibilidad con HTML5, CSS3 y las API asociadas a estas tecnologías. Por esta razón, uno de los puntos clave a la hora de aplicar nuevas tecnologías Web en nuestros desarrollos, será la detección de la compatibilidad del lado cliente.

En general esto lo lograremos desde código escrito en JavaScript, pero también existen muy buenos *frameworks* que nos ayudarán en esta tarea. Modernizr es una librería que nos hará mucho más liviano nuestro trabajo, logrando su objetivo de una manera eficaz.

Trabajar con Modernizr para detectar compatibilidad

Parar arrancar con Modernizr, encontraremos todo lo que necesitamos en el sitio Web http://modernizr.com/. Esta librería puede ser descargada con opciones personalizadas. Se puede elegir la versión de desarrollo (*Development)* o construir una versión de producción que se ajuste solo a las características que necesitamos detectar en cada caso (*Production*). Este último camino es más liviano y compacto, ideal para incluir en nuestro proyecto.

En la sección **Documentation** (http://modernizr.com/docs/) hay información para comenzar a comprender la naturaleza de Modernizr, y también opciones y ejemplos de uso. Dentro de ese mismo documento, en la sección que lleva el título **Features detected by Modernizr**, encontraremos qué características pueden ser evaluadas utilizando Mondernizr, y cómo debemos escribir el código para lograrlo. Al evaluar una característica, recibiremos como resultado **True** si es compatible con el navegador que está corriendo el código; en caso contrario, **False** para los no compatibles.

A continuación, estudiaremos algunos ejemplos que nos ilustrarán de qué manera se puede trabajar con Modernizr para verificar la compatibilidad con una característica HTML5. En este caso, verificaremos la compatibilidad con Local Storage, pero podríamos realizarla con cualquier otra característica.

En primer lugar, desde el sitio Web http://modernizr.com/ hay que ingresar en la sección de descarga (**Download**), e indicar la(s) característica(s) que vamos a evaluar; luego, pulsamos el botón **Generate** para que se genere el código personalizado, según lo indicado en esta pantalla. Para descargarlo hay que utilizar el botón **Download**.

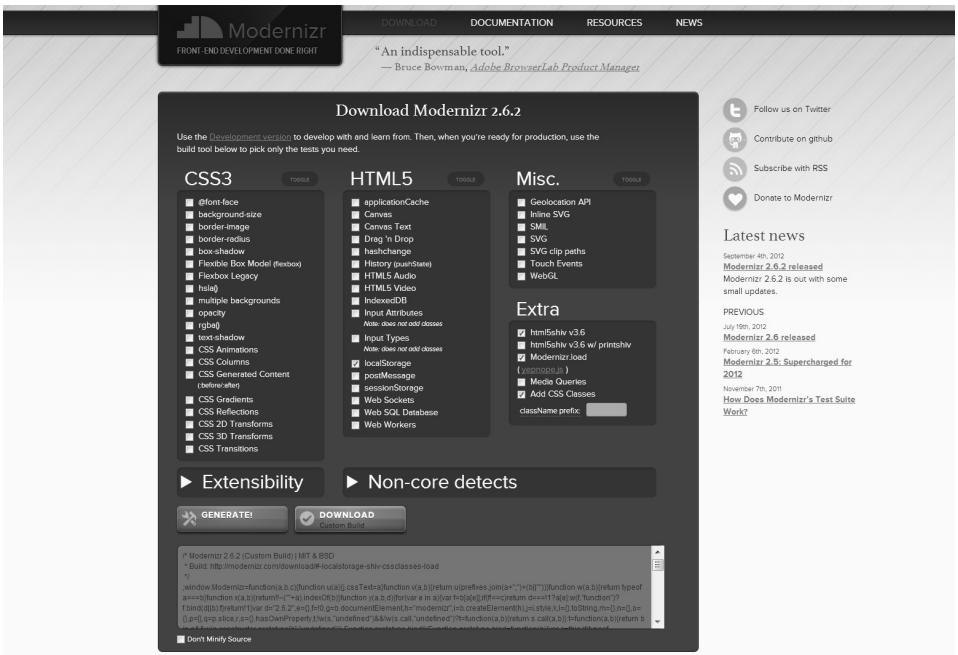

Fig. 4.13. Modernizr permite seleccionar múltiples características para evaluar, pero es recomendable solo elegir las que se utilizarán en el proyecto que se está desarrollando.

El siguiente paso será colocar en la carpeta correspondiente a nuestro proyecto el archivo de Modernizr que se ha generado. Lo haremos así:

```
<script language="javascript"
src="modernizr.base.js"></script>
```

El código que nos permitirá realizar la detección de la característica es el siguiente:

```
if (Modernizr.localstorage){
   console.log("Funciona HTML5 LocalStorage");
      }
else{
   console.log("No funciona HTML5 LocalStorage");
            }
```

Este ejemplo genera un mensaje en la consola indicando si está presente la característica *Local Storage* en el navegador.

Emuladores y simuladores

Cuando hablamos de emuladores y simuladores, ante todo debemos saber que no son exactamente lo mismo. Si bien con ambos podremos poner a prueba a nuestros proyectos, existen diferencias que debemos conocer.

Los emuladores permiten ejecutar la aplicación dentro del sistema operativo para el cual fue creada, y a su vez, emular las características de hardware de la arquitectura original. Por otra parte, los simuladores son aplicaciones que tienen como objetivo brindar a los usuarios y desarrolladores, la posibilidad de probar una aplicación reproduciendo un entorno similar al original.

Para el mundo de los móviles, estas herramientas serán aliadas muy importantes para examinar nuestros desarrollos. Si bien es fundamental contar con dispositivos físicos para realizar pruebas y disponer de aplicaciones que nos permitan efectuar un test, también constituyen un mecanismo decisivo a la hora de avanzar en nuestro desarrollo y detectar problemas. Pueden ser útiles para probar los pequeños cambios que vamos introduciendo, y también una alternativa interesante para testear cuando no tenemos el dispositivo físico en nuestro poder. Aunque en algún momento del desarrollo será imprescindible realizar pruebas en *smartphones* y *tablets* de verdad.

A continuación nos centraremos en los principales emuladores y simuladores que podemos encontrar para realizar un test desde nuestra computadora:

- iOS Simulator para Mac OSX que incluye soporte para iPad, iPhone e iPod Touch (https://developer.apple.com/devcenter/ios/index.action#downloads).
- Los Emuladores de sistemas Android se incluyen junto con el SDK, y se pueden descargar diversas versiones (Level API) una vez instalado (http://developer.android.com/sdk/index.html).
- El emulador de Windows Phone se incluye junto con los SDK de cada versión (http://dev.windowsphone.com/en-us/downloadsdk).
- BlackBerry Simulators para PlayBook, BlackBerry 10 y otros *smartphones* de BlackBerry (http://us.blackberry.com/sites/developers/resources/simulators.html).
- Opera Mobile Emulator (http://www.opera.com/developer/tools/mobile/).

Finalmente, *Mobile Emulators & Simulators: The Ultimate Guide* es un documento, creado por Maximiliano Firtman, que ofrece una amplia guía de recursos para desarrolladores. Se encuentra disponible en:

http://www.mobilexweb.com/emulators.

Herramientas para debugging y profiling

Para muchos el mundo sería otro si los proyectos funcionaran siempre en la primera prueba, sin embargo, todos sabemos que las etapas de *testing* son necesarias siempre.

Anteriormente, en este mismo capítulo, hablamos sobre simuladores y emuladores. Ahora analizaremos las técnicas que nos ayudarán a encontrar errores y depurar el rendimiento de nuestra aplicación.

Cuando nos referimos al *debugging*, nos referimos a un proceso que nos brinda la posibilidad de localizar y corregir los errores que se encuentren en este procedimiento.

Por otra parte, el *profiling* apunta a la mejora en el rendimiento de la aplicación, que se alcanzará mediante los diversos análisis que se le pueden efectuar a la aplicación mientras se está ejecutando. Esto permite tener una visión sobre cómo está funcionando la aplicación y qué aspectos pueden mejorarse para optimizar el rendimiento. Las herramientas que permiten realizar los test de rendimiento se denominan con el término inglés *profilers*.

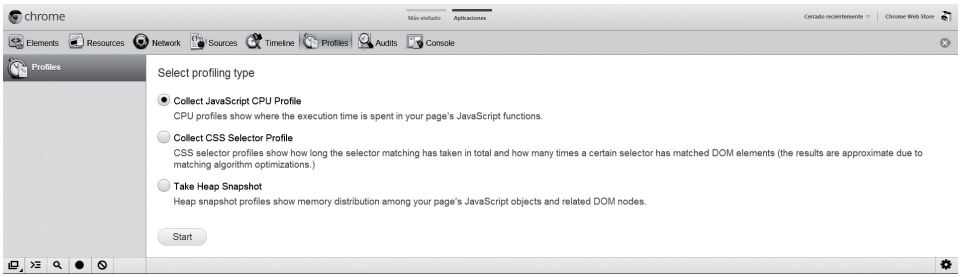

Fig. 4.14. Google Chrome, en su versión desktop, permite realizar profiling desde la opción ubicada en la caja de herramienta para desarrolladores.

Los desarrolladores que utilizan Mac OSX pueden aprovechar las ventajas que brinda iWebInspector. Esta aplicación gratuita permite realizar *debugging* e inspeccionar código que esté corriendo en el navegador del iOS Simulator, en sistemas Mac OSX 10.6 o 10.7. Requiere Xcode 4.2 con SDK iOS 5.0 o Xcode 4.3 con SDK iOS 5.1. El sitio Web donde se la puede descargar es: http://www.iwebinspector.com/.

A partir del lanzamiento de iOS 6 y Safari 6 por parte de Apple, surgen varias novedades muy importantes para los desarrolladores en este campo, ya que se incorpora de manera oficial el **Remote Web Inspector**, un Inspector Remoto Web.

Con este inspector podremos realizar la tarea de *debugging* de una forma integrada, que nos permitirá trabajar tanto con el simulador como con un dispositivo real que tenga iOS 6 o superior.

Para acceder a estas funcionalidades desde Safari 6, hay que ingresar a las opciones avanzadas del navegador, en el dispositivo, y habilitar el inspector, que está deshabilitado por defecto. Desde el dispositivo con iOS 6, se debe ingresar en **Ajustes**, luego pulsar sobre **Safari,** y acceder a la opción **Avanzado**. Allí encontraremos la función que nos permitirá habilitar en Inspector Web, y nos ofrecerá una breve descripción de cómo usarlo (nos indicará que debemos conectar el dispositivo a la computadora mediante el cable).

Con esta herramienta podremos hacer *debugging* remoto desde la versión de Safari desktop, con la que podremos analizar las vistas de la ventana del navegador del dispositivo o del simulador, y también de aplicaciones que utilizan el Web View de iOS, como es el caso de PhoneGap, entre otras opciones.

Dentro de las funciones de este *debugging* remoto se destaca la posibilidad de ver y manipular código en vivo, encontrar errores, realizar pruebas con *breackpoints* de JavaScript, y acceder a información de almacenamiento local.

Otra alternativa es la que ofrece **Adobe Edge Inspect**, anteriormente llamado Adobe Shadow. Este inspector nos permite navegar y sincronizar los resultados mediante una conexión Wi-Fi; de esta manera, podremos realizar una inspección remota de sitios y aplicaciones desde nuestro navegador. Con estas herramientas también será posible hacer cambios para verlos en vivo. Además podremos tener capturas de pantallas y grabarlas en nuestro equipo.

Para comenzar a utilizar Adobe Edge Inspect (parte de lo que ofrece Edge Tools & Services) es necesario estar registrado con una cuenta *Creative Cloud*. Este paso puede realizarse ingresando en: http://html.adobe.com/edge/inspect/. Se puede utilizar en sistemas Windows 7, Windows 8 o Mac OSX 10.6 o superior. Para realizar el proceso de inspección, el equipo debe tener instalado Google Chrome 14 o superior (se recomienda que la versión utilizada sea 21 o superior).

Una vez registrados, podemos acceder a descargar Adobe Edge Inspect en la siguiente dirección Web: https://creative.adobe.com/inspect.

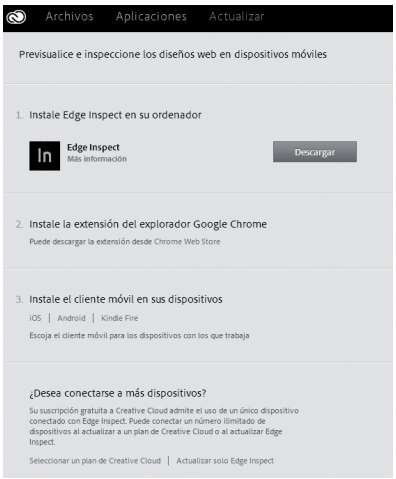

Fig. 4.15. Cuando se accede a la descarga de Edge Inspect, también se ofrecen las alternativas para acceder a la extensión para Google Chrome, y las aplicaciones para dispositivos móviles, entre otras opciones.

Una vez descargado, en primer lugar, hay que ejecutar el instalador de Edge Inspect para instalar la aplicación en nuestro equipo. Para comenzar a utilizarlo deberemos autenticarnos con nuestro correo electrónico y contraseña registrada para los servicios de Adobe.

El paso siguiente será instalar la extensión para Google Chrome, disponbile en Chrome Web Store. El *link* para acceder se encuentra disponible debajo de la opción de descarga de Edge Inspect.

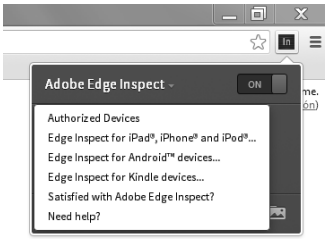

Fig. 4.16. Una vez instalada la extensión es posible habilitar, deshabilitar o configurar las opciones desde la esquina superior derecha de Google Chrome, donde se suelen incorporar las extensiones del navegador.

Luego de estos pasos se puede proceder a instalar las aplicaciones en los dispositivos. Desde la página de descarga de Adobe Edge Inspect se puede acceder a los enlaces de las tiendas para descargar las versiones compatibles con iOS (4.3 o superior), Android (2.1 o superior) y Kindle Fire.

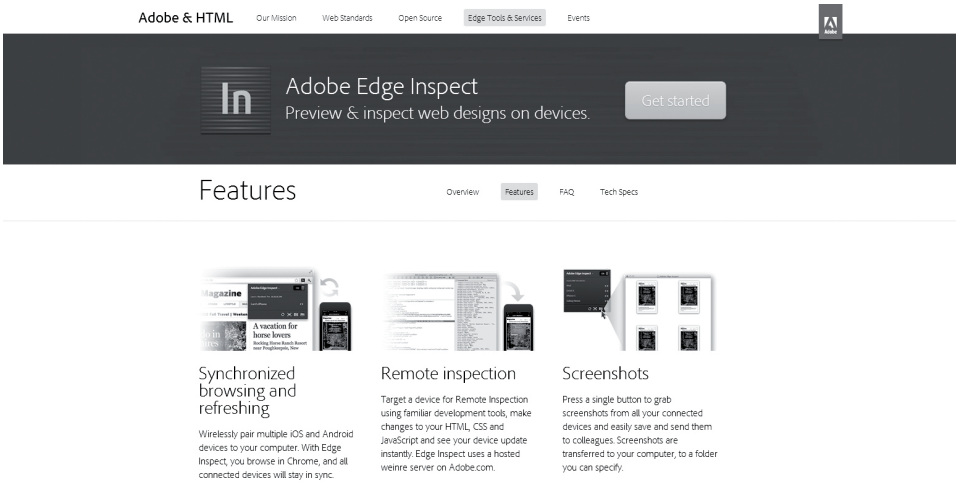

Fig. 4.17. Para aquellos que deseen saber más sobre las opciones que brinda esta herramienta, las características principales de Adobe Edge Inspect pueden leerse en: http://html.adobe.com/edge/inspect/features.html.

Maquetado y programación de la aplicación

5

De la planificación al código

En el segundo capítulo del libro, atravesamos las etapas de planeamiento de nuestra Web App. Primero, comenzamos a cambiar nuestra manera de pensar, para comprender mejor lo que necesita el mundo de las aplicaciones móviles. Luego, aprendimos la importancia que tiene en la planificación la arquitectura de la información, y cómo llevar adelante este factor a favor de nuestra aplicación.

Vistos los conceptos de usabilidad y accesibilidad para el mundo móvil, emprendimos el camino de crear el diagrama de flujo de la aplicación y armamos los *mockups* que componen las pantallas de la Web App.

En los siguientes capítulos, conocimos los lenguajes, tecnologías y herramientas que nos abren la posibilidad de llevar a la realidad nuestro proyecto. En este capítulo, ha llegado la hora de poner manos a la obra sobre el código, aplicando lo aprendido e incorporando algunos condimentos nuevos que permitirán que nuestra Web pueda tomar cuerpo y forma.

Para llevar a la práctica nuestro proyecto, primero vamos a analizar cómo optimizar las imágenes y qué técnicas podemos emplear para lograr un mejor rendimiento. Además, vamos a utilizar una librería que nos facilitará el desarrollo y nos permitirá simplificar la escritura de código para las pantallas de nuestra aplicación. Con este *framework* también vamos a aplicar efectos visuales de transiciones y podremos simplificar el trabajo con eventos, entre otros aspectos importantes.

Preparación y optimización de imágenes

Las imágenes juegan un papel muy importante en nuestra interfaz gráfica, tanto para enriquecer el aspecto visual de nuestra aplicación como para guiar y ayudar al usuario a comprender su funcionamiento y acciones que puede realizar.

En cuanto a la funcionalidad, escoger las imágenes adecuadas colaborará en gran medida en el éxito de la aplicación. En la parte estética, se recomienda elegir o crear imágenes que formen un conjunto homogéneo y respeten un estilo.

Formatos de imágenes y compresión

Existe una gran variedad de formatos de imágenes, pero no todos son ideales para el desarrollo de aplicaciones Web en general o para aplicaciones móviles en particular. Por esta razón es importante repasar qué ventajas nos ofrece cada formato y cuál es el más conveniente en cada caso:

JPEG: permite llegar a 24 bits de color. Ofrece formatos de alta compresión (con pérdida). No soporta transparencias. Es muy utilizado para fotografías en la Web.

GIF: puede trabajar con paletas de 2 a 256 colores. Generalmente no se usa para fotografías, solo para dibujos de líneas blanco y negro o color. Soporta transparencias.

PNG: puede trabajar con paletas de 8 bits o 24 bits de color y su algoritmo de compresión es sin pérdidas. Soporta transparencias. Todas estas características lo destacan como un formato muy versátil para la Web.

WEBP: es un formato de imágenes nuevo, impulsado por Google. Este formato gráfico se basa en compresión con pérdida. Puede trabajar con imágenes de 24 bits de color con una muy buena compresión. Por el momento, su soporte nativo en navegadores aún no es demasiado alto.

Para el proyecto que estamos llevando adelante, necesitaremos utilizar imágenes e íconos, ya que son opciones que ayudan al usuario en el reconocimiento de la aplicación "a primera vista". Existen muchos recursos en Internet que nos ofrecen íconos de muy buena calidad. Para el proyecto utilizaremos los que provee jQuery Mobile y otros que se pueden obtener en sitios que brindan íconos o bancos de imágenes. Una buena opción es realizar una búsqueda en http://www.iconfinder.com/, ya que pone a nuestra disposición más de 200 mil íconos en casi 1200 *sets* de imágenes. Al elegir los recursos es muy importante verificar el tipo de licencia con el que se comparten y si pueden ser utilizados en proyectos personales, comerciales u otro tipo y en qué casos corresponde citar al autor o incluir un enlace a éste.

Sprites

El uso de *sprites* para definir una especie de "mapa de imágenes" es una técnica muy útil, heredada del mundo de los videojuegos, que goza de un éxito renovado en el mundo de las aplicaciones Web y Móviles.

Los primeros pasos en el mundo de los *sprites* fueron dados entre las décadas de los setenta y de los ochenta para hacer más eficaz el uso de gráficos en las consolas de videojuegos que comenzaban a sorprender al mundo en aquellos años.

En el ámbito del diseño Web se comenzó a utilizar esta técnica para trabajar de manera más eficaz en sitios y/o aplicaciones que contaban con gran número de imágenes. Agrupar los gráficos en una sola imagen, formando un *sprite*, permite reducir la cantidad de peticiones del servidor y, a su vez, tener precargados estos contenidos, evitando, por ejemplo, parpadeos no deseados en transiciones y animaciones. En los móviles también puede ser útil si se toman imágenes provenientes de un servidor ya que reduce de manera importante las peticiones HTTP.

En el caso de las aplicaciones que funcionan completamente *offline* o instaladas en el dispositivo, el uso de *sprites* puede resultar útil para organizar los recursos gráficos del proyecto. En este caso, vale la pena aclarar que si los archivos se encuentran en el dispositivo local no hay una ventaja tan significativa en cuanto a *performance*, porque no existen las peticiones al servidor. Sin embargo, y más allá de no obtener una diferencia notable, para el dispositivo puede ser más liviano el trabajo de leer una sola imagen que tener que buscar y cargar muchas.

Si se trabaja con aplicaciones complejas (por ejemplo, juegos) al emplear texturas o animaciones que requieren mucha memoria, también encontramos una ventaja en los *sprites*. Aquí el ahorro lo podemos lograr al limitar la cantidad de llamadas necesarias a las características de dibujo del sistema.

Si bien existen algunas herramientas específicas para su creación, los *sprites* pueden ser armados con cualquier editor de imágenes que nos permita realizar la composición del mapa de bits con todo el contenido gráfico necesario.

Para todos aquellos que deseen crear y editar imágenes para sus proyectos, Gimp es una opción gratuita (Licencia GPL) y multiplataforma (Windows, Linux, Unix y Mac OSX, entre otras). Se encuentra disponible en una gran variedad de idiomas (incluido el español), y es una alternativa que nos ayudará a trabajar con las imágenes de nuestros desarrollos. Se puede descargar desde el sitio Web: http://www.gimp.org/.

Comenzar con jQuery Mobile

Uno de los puntos clave en nuestros desarrollos es la elección de las librerías que utilizaremos para llevarlo a cabo. Si bien en algunos casos, por razones de *performance*, requerimientos, diseño y/o simplicidad puede resultar mucho más efectivo crear la aplicación sin emplear librerías adicionales, en otras oportunidades nos ayudarán a ahorrar horas de trabajo.

Por las características de este proyecto y por sus fines didácticos, nos apoyaremos en la flexibilidad y ventajas que nos provee jQuery Mobile para el desarrollo de aplicaciones para móviles con soporte *cross-platform*.

jQuery Mobile es un *framework* que trabaja con jQuery y se basa en el lema "Write less, do more" (escribe menos, haz más), ofreciendo una alternativa no intrusiva para el desarrollo de interfaces móviles. Se destaca por su fácil implementación, soporte de características de accesibilidad, muy buenas alternativas de personalización y la posibilidad de implementarlo mediante técnicas de mejora progresiva. Con esta librería, aplicando los conceptos que aprendimos en el capítulo anterior, vamos a ser capaces de armar nuestra aplicación y dotarla de funcionalidades, reduciendo la cantidad de pasos para llegar a nuestro objetivo final.

jQuery Mobile divide su grado de compatibilidad en tres niveles: A (*full*), B (*full minus* Ajax), C (*basic* HTML). El nivel A será el que mayor cantidad de recursos de la librería permite aprovechar, y el C solo es compatible para trabajar con las características básicas de HTML. En el documento http://jquerymobile.com/gbs/ hay más detalles sobre este aspecto y las plataformas que se agrupan en cada una de estas categorías.

Descargar jQuery Mobile e integrarlo en nuestro proyecto

jQuery Mobile ofrece una buena solución para *Web Designers*, ya que podrán hacer mucho, escribiendo poco código. Pero también puede resultar una opción interesante para desarrolladores, ya que no deberán meterse en el duro camino de diseñar la interfaz gráfica de la aplicación.

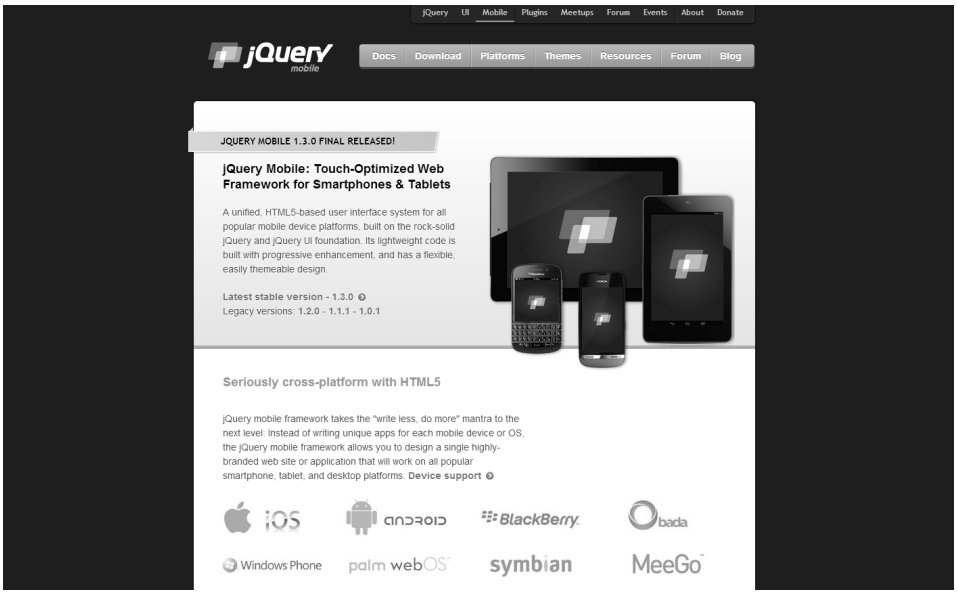

Fig. 5.1. jQuery Mobile se puede descargar en su última versión estable (*Latest stable version*) o en sus versiones previas disponibles (*Legacy versions*) ingresando en http://jquerymobile.com/.

Al descargar el archivo comprimido del sitio de jQuery Mobile, veremos que contiene todos los archivos del *framework* y también ejemplos. Dentro del paquete, encontraremos los archivos JavaScript, CSS y las imágenes necesarias para incorporarlas en nuestro proyecto. Debemos recordar también que para que todo funcione, la librería jQuery debe estar presente en el proyecto y debe ser llamada desde el documento HTML, antes de agregar el *script* principal de jQuery Mobile.

Para trabajar de manera básica, podremos optar por copiar a nuestro proyecto los archivos "minificados" que contienen todas las funcionalidades de esta librería, pero sin espacios innecesarios y comentarios, para reducir su peso.

Para comenzar, los archivos que necesitamos incorporar en nuestro proyecto son los siguientes: **jquery.mobile-x.x.x.min.css** y **jquery.mobile-x.x.x.min.js** (la porción del nombre **x.x.x** identifica la versión). También es recomendable copiar a nuestro proyecto la carpeta **images** y su contenido, donde encontraremos las animaciones y sprites que utiliza jQuery Mobile de manera predeterminada.

En el nivel inicial del archivo comprimido (el **.zip** que descargamos), encontraremos también algunos archivos CSS y JavaScript para estructura y personalización, que en principio no utilizaremos en este proyecto, pero que pueden ser útiles en otros desarrollos. Dentro del mismo archivo comprimido, si recorremos la carpeta **demos**, podremos ver ejemplos de uso de esta librería. Si creamos o descargamos un

theme personalizado, también deberemos agregar los archivos necesarios a nuestro proyecto para luego utilizarlos.

Algunos archivos, como por ejemplo la librería jQuery o jQuery Mobile (disponibles en http://code.jquery.com/), pueden estar hospedados en nuestro servidor o en un CDN; sin embargo, si luego empaquetaremos la aplicación, debemos asegurarnos de que todos los archivos estén en las carpetas del proyecto, para no crear dependencias innecesarias a recursos que se encuentren en Internet.

Una vez que tenemos los archivos necesarios, podremos comenzar a trabajar sobre el código de nuestro proyecto; primero, definiendo las etiquetas que introduciremos en el encabezado.

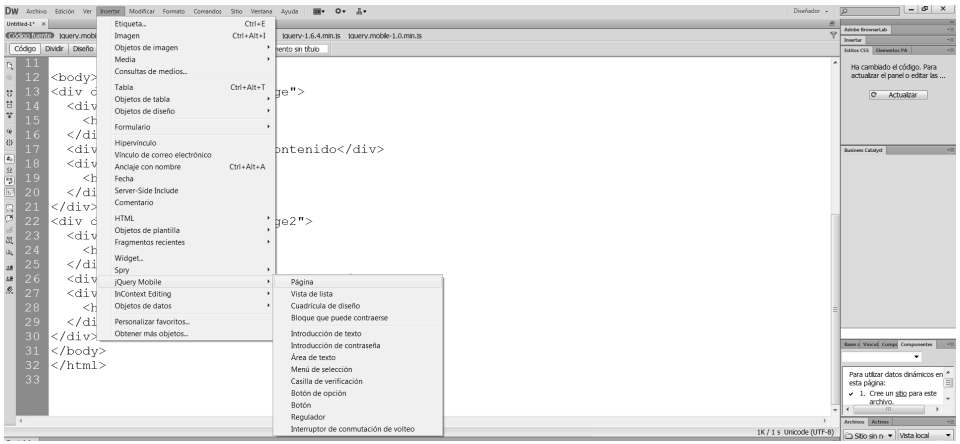

Fig. 5.2. Adobe Dreamweaver (CSS5.5, CSS6 o CC) ofrece una fácil integración con jQuery Mobile desde las opciones de menú, y también permite la previsualización desde la vista en vivo con que cuenta el programa (Live View). Para nuestro proyecto podremos optar por esta aplicación o podremos trabajar desde otro editor de código.

Las primeras versiones de jQuery Mobile fueron publicadas en 2010 (1.0a1 y 1.0a2). Todas las novedades sobre este *framework* se pueden seguir desde el blog: http://jquerymobile.com/blog/.

El encabezado del documento

Para comenzara a crear el documento HTML de nuestra Web App, primero vamos a escribir el `DOCTYPE`. Como trabajaremos con HTML5, el tipo de documento quedará definido de la siguiente manera:

```
<!DOCTYPE html>
```

Ahora vamos a trabajar sobre el <head>, que como hemos visto es una etiqueta opcional en HTML, pero para agregar claridad a este ejemplo la utilizaremos e incorporaremos el código dentro de ella (reemplazar **x.x.x** por la versión correspondiente):

```
<head>
<title>Mi primera Web App</title>
<meta charset="UTF-8" />
<link rel="stylesheet"
href="css/jquery.mobile-x.x.x.min.css" />
<script src=
"js/jquery-x.x.x.min.js"></script>
<script src=
"js/jquery.mobile-x.x.x.min.js"></script>
</head>
```

El título lo podremos personalizar, según el nombre que le demos a la aplicación. La metaetiqueta `charset` es recomendable incluirla con el valor `UTF-8` para compatibilidad con el juego de caracteres de nuestro idioma. El nombre de los archivos que especificamos dentro de las etiquetas <link> y <script>, puede variar según la versión. La ruta de la carpeta dependerá de la ubicación que le destinemos a los archivos en el proyecto.
Como hemos visto en el capítulo 3, también en la sección del encabezado, podremos escribir y configurar los valores para la metaetiqueta `viewport`. Con esta opción, especificamos las acciones posibles para el zoom y, además, como "encajará" la aplicación en la vista. Esto resuelve algunos problemas que tienen algunos visualizadores al mostrar el contenido de manera correcta y ajustada a las características de la pantalla. Recordamos el código para fijar el nivel de *zoom* inicial e impedir que el usuario escale la Web App en la vista de la aplicación:

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

Más adelante en el libro, veremos que contamos con la opción de incorporar otras etiquetas según el enfoque de nuestro proyecto. Para esta etapa, serán suficientes estas líneas de código que hemos especificado en nuestro encabezado.

Custom data y los roles en jQuery Mobile

Con HTML5 se introduce el concepto de **Custom data** o, para decirlo en español, la posibilidad de incorporar datos personalizados en los atributos. Esta característica antes estaba resuelta con algunas librerías o de una manera no estándar, pero

ahora forma parte de la renovación que impulsa HTML5. Esto permite, de una manera no intrusiva y sencilla, colocar atributos que luego pueden ser trabajados desde JavaScript y CSS para incorporar funcionalidades y detalles de apariencia.

jQuery Mobile aprovecha la potencia de esta característica para permitir el armado de un esquema de páginas y, a su vez, personalizar nuestra aplicación de una manera muy simple. Para trabajar con este *frameworks* encontramos atributos que nos permiten definir la estructura de cada página, personalizar el tema y seleccionar las transiciones, entre otras muchas opciones disponibles. Esta información la incorporamos como atributos dentro del código. Por ejemplo, para definir una página en jQuery Mobile lo hacemos de la siguiente forma:

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

En la tabla que veremos a continuación, encontraremos el detalle de los principales "roles" disponibles en jQuery Mobile.

data-role	Descripción
page	Se utiliza para definir una página que jQuery Mobile puede mostrar de manera independiente, como si fuera una pantalla individual de la aplicación.
header	Encabezado en la página.
content	Permite establece la sección de contenido principal de una página.
footer	Pie de la página.
navbar	Barra de navegación de la aplicación
listview	Se emplea para contener múltiples elementos de lista y así generar una vista de ellas. Puede trabajar con un buscador y aplicar filtros a los elementos de la lista (*data-filter*).
button	Se utiliza para definir un elemento como un botón.
controlgroup	Permite agrupar botones.
collapsible	Es útil para crear contenido que puede colapsarse. Al presionar sobre el elemento que cuenta con este atributo, muestra u oculta un contenido. Puede trabajar en conjunto con *data-collapsed* para definir si por defecto el contenidos se expande (o no).

collapsible-set	Permite definir un conjunto de elementos que se pueden colapsar. Se pueden lograr efectos similares al cásico acordeón.
dialog	Permite crear ventanas de diálogo para la aplicación.

Fig. 5.3. Valores para el atributo *data-role* en jQuery Mobile.

Es importante señalar que el concepto de página de jQuery Mobile no requiere que el usuario deba hacer alguna acción adicional para ocultarlas o mostrarlas, ya que de eso se encargará la librería de manera autónoma. Simplemente al seguir el *link* que corresponda, como veremos más adelante, se podrán mostrar páginas que formen parte del mismo documento HTML o incorporar contenido externo.

Otro aspecto a tener en cuenta es que si bien Custon Data permite la creación de atributos personalizados, al utilizar jQuery Mobile debemos emplear los *data-role* que están definidos en el *framework*, para usar sus características. Si creamos Custom Data con valores que no existen, no tendrán efecto, excepto que los programemos por nuestra cuenta o desarrollemos algún tipo de *plugin*.

Estructura de una aplicación con jQuery Mobile

En este apartado veremos cómo comenzar a crear la estructura de la Web App que comenzamos a planificar en el capítulo 2 de este libro.

En primer lugar, analizaremos cómo crear la estructura HTML de las pantallas que, en este caso, serán las páginas de jQuery Mobile. Con fines prácticos para este proyecto, estas páginas formarán parte de un único documento HTML.

Creación de la estructura de las páginas

Como hemos visto, en jQuery Mobile podemos crear las pantallas de nuestra aplicación mediante el concepto de páginas, que pueden estar albergadas en un mismo documento HTML o en diferentes; en este caso, optaremos por la primera alternativa. Dentro de las páginas (`data-role="page"`), podremos definir un lugar para el encabezado (`data-role="header"`), otro para lo que será el contenido principal (`data-role="content"`) y el pie (`data-role="footer"`). Esto establece una estructura elemental que luego podremos personalizar, según los requisitos de la aplicación. Para comenzar a visualizar estos conceptos en el código, vamos crear una página, con su estructura básica:

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

Como vemos en el código anterior, definimos una página con una cabecera, una sección principal de contenido y un pie. No es obligatoria armar siempre esa estructura, ya que por ejemplo, en algunos casos puede variar según la distribución que tienen las pantallas de la aplicación que se esté desarrollando.
Con jQuery Mobile es posible trabajar con las etiquetas HTML/XHTML, como el caso de `<div>`, pero también podremos emplear las semánticas que introduce HTML5, como `<header>`, `<section>` o `<footer>`. Cualquiera sea el camino que escojamos, debemos recordar que es fundamental especificar el `data-role` correspondiente. Esto es fundamental para que jQuery Mobile aplique los estilos necesarios, ya que de lo contrario no se llevará a cabo ninguna acción sobre el contenido.

Un aspecto que puede resultar interesante en algunas aplicaciones es establecer la cabecera y/o el pie como persistentes. Esto se puede lograr incluyendo el atributo `ui-state-persist`. Si además deseamos que el elemento persista entre las transiciones, podemos utilizar el atributo `data-id`, cuyo valor debería ser igual entre las páginas que deseamos mantenerlo fijo en el momento del pase de página.

Enlaces

En nuestro proyecto, la navegación principal de la aplicación está definida en la parte inferior de la pantalla. El esquema que usaremos en jQuery Mobile para llevar a la práctica esta característica será mediante *links* internos a diferentes páginas ubicadas en el mismo documento.

Este concepto de documento multipágina, lo llevamos a cabo especificando una id para cada página. Es decir, cada contenedor `data-role="page"`, deberá tener una *id*. Los *links* que referencien a esa página deben referirse a ese identificador. Por ejemplo, para ofrecer un enlace que direccionará a una página cuyo *id* es `pagina2` deberemos escribir el siguiente código HTML:

```
<a href="#pagina2">Página 2</a>
```

Cuando el usuario pulse el botón o el enlace correspondiente, transición mediante, se esconderá la página actual y se mostrará la referenciada en el enlace.

Si deseamos ubicar las páginas de nuestra aplicación jQuery Mobile en diferentes documentos, podremos hacerlo escribiendo un código como el siguiente:

```
<a href="pagina2.html">Link a otra página</a>
```

Para que funcione como una página en nuestra Web App, **pagina2.html** debe encontrarse en el mismo dominio, contener la estructura de una página de jQuery Mobile y tener incluidos los archivos de jQuery y jQuery Mobile (CSS y JavaScript) en la cabecera. La página externa que se cargue por este método se agregará al árbol del DOM de la estructura principal.

Si deseamos ganar velocidad en la carga de algunos *links* hacia página externas que pertenecen a la aplicación, podemos usar una precarga, utilizando una técnica conocida como **prefetch**. Esto se implementa colocando el atributo `data-prefetch`. No es recomendable abusar de esta técnica, ya que no sería bueno para el rendimiento aplicarlo a todas las páginas que componen la aplicación. Un ejemplo de uso es el que se detalla en el código a continuación:

```
<a href="pagina2.html" data-prefetch>Link a otra página</a>
```

Para crear un enlace hacia una página externa que no forma parte de nuestra aplicación, podremos hacerlo utilizando `data-rel="external"`, como vemos en el siguiente código:

```
<a href=" http://css3html5.com.ar/"
data-rel="external">CSS3 y HTML5</a>
```

Es importante destacar que también se abrirá como página externa un documento HTML del mismo dominio si al ancla tiene asignado el atributo `target="_blank`.

Botones

Para lograr que un elemento tenga el aspecto de botón, jQuery Mobile pone a nuestra disposición el atributo `data-role="button"`, que podremos aplicar, por ejemplo, a un enlace.

Podremos establecer los íconos a mostrar por medio del atributo `data-icon`. Esto permite utilizar imágenes predefinidas en la librería para definir íconos típicos de aplicaciones móviles. Los valores posibles son: `arrow-l` (flecha izquierda), `arrow-r` (flecha derecha), `arrow-u` (flecha hacia arriba), `arrow-d` (flecha hacia abajo), `delete` (borrar, x), `plus` (signo más, +), `minus` (signo menos, -), `check` (símbolo de check), `gear` (ícono de engranaje, para configuración), `refresh` (refrescar), `forward` (acción para ir hacia adelante), `back` (acción para ir hacia atrás), `grid` (grilla) `star` (estrella), `alert` (alerta), `info` (información) `home` (inicio), `search` (ícono de búsqueda).

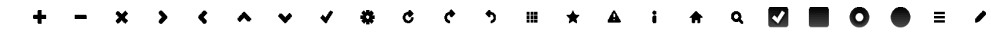

Fig. 5.4. Dentro de los recursos que incluye jQuery Mobile se encuentra una carpeta con imágenes que componen *sprites* utilizados para crear los botones.

Los botones tienen efectos aplicados de manera predeterminada, pero podemos modificar esto. Para deshabilitar sombras, colocamos `data-shadow="false"` y para deshabilitar bordes redondeados, podemos incluir `data-corners="false"`.

Para definir que los botones estén en línea, no necesitamos recurrir a CSS de manera directa, sino que podremos aplicar el atributo `data-inline="true"`, al elemento que tenga aplicado el atributo `data-role="button"`.

Los botones pueden agruparse con una caja. Por ejemplo, a una etiqueta `<div>` se le puede asignar el valor: `data-role="controlgroup"`. De esta manera, el grupo de botones se mostrará como un conjunto unido, sin espacios entre ellos, pero manteniendo la división en cada botón. Dentro de cada grupo de botones, los elementos estarán definidos como `<a>`, y tendrán `data-role="button"`. Si deseamos que se vean en línea agregamos también `data-inline="true"`.

Las páginas de jQuery Mobile soportan la opción de incorporar un botón para regresar atrás ("back button"). Para definirlo dentro de un contenedor que tenga asignado data-role="page", podemos incluir el atributo `dataadd-back-btn="true"`. Para asignarle un texto a este botón, usamos `data-back-btn-text="Anterior"` (personalizamos el mensaje que está entre comillas); y si deseamos modificar el *theme*, aplicamos el atributo `data-back-btn-theme="c"` (especificamos la letra del *theme* entre comillas).

Si deseamos especificar sobre un botón de nuestra interfaz que funciona como una opción para regresar hacia atrás en la navegación, podremos agregarle el atributo `data-rel="back"` y `data-icon="back"`.

Listas

Las listas son elementos muy útiles en la creación de aplicaciones móviles. En jQuery Mobile para definir una lista, podremos aplicar el atributo `data-role="listview"` al `<ul>` que contiene los elementos de lista. De esta manera, podremos crear un código como el del siguiente ejemplo:

```
<ul data-role="listview">
      <li><a href="#item1">Item 1</a></li>
      <li><a href="#item2">Item 2</a></li>
      <li><a href="#item3">Item 3</a></li>
</ul>
```

La diferencia que obtendremos será esencialmente visual y de usabilidad. La lista se mostrará de la manera en que estamos acostumbrados a verla en las interfaces móviles. Los elementos tendrán el ancho completo de la pantalla y el alto adecuado para ser utilizados en una interfaz táctil. Luego, podremos personalizar la combinación de color y otras características de representación, cambiando o modificando el *theme*.
Si deseamos agregar otras características, jQuery Mobile ofrece numerosas opciones para las listas: anidar elementos, numerar ítems, agregar imágenes e íconos, incorporar separador, y hasta colocar un buscador que actúa como filtro, entre otras opciones.

Las listas numeradas las podemos crear utilizando las listas ordenadas de HTML `<ol>` junto con `data-role="listview"`:

```
<ol data-role="listview">
```

Para incorporar un separador, jQuery Mobile nos permite agregar a una etiqueta `<li>` el atributo `data-role="list-divider"`.

Si deseamos incorporar un buscador que actúe como filtro en la lista, deberemos agregar al `<ul>` el `atributo data-filter="true"`, como vemos en el código que sigue a continuación:

```
<ul data-role="listview" data-filter="true">
```

Si necesitamos agregarle un texto temporal, podemos aplicarle al código anterior el atributo `data-filter-placeholder`.

La opción de generar elementos de lista que se colapsen, se puede lograr especificando el atributo `data-role="collapsible"` a un contendor. Podemos crear un conjunto de elementos que se colapsan empleando sobre otro contenedor general el atributo `data-role="collapsible-set"`.

Las listas en jQuery Mobile también pueden utilizar lo que se conoce como *Bubble* (burbujas), y mostrar por ejemplo la cantidad de elementos que contienen dentro, como vemos en el siguiente ejemplo:

```
<li><a href="#p1">Item 1</a>
<span class="ui-li-count">10</span>
</li>
```

Para definir el *theme* de la burbuja, debemos recurrir al atributo `data-count-theme` que irá aplicado al `<ul>` correspondiente a la lista.

Toolbars

Cuando trabajamos con jQuery Mobile podemos establecer dos tipos de *Toolbars*: el encabezado (*header bar*) y el pie (*footer bar*).

Para fijar las *toolbars*, podemos agregarles el atributo `data-position="fixed"`.

Por ejemplo, para crear la *header bar* de nuestra aplicación utilizamos el siguiente código:

```
<header data-role="header"  data-position="fixed" data-id
="cabecera">
 <a href="#principal" data-icon="home"
data-role="button" title="Inicio" data-iconpos="notext" class="ui-
btn-active">Inicio</a>
 <h1>Logo</h1>
 <a href="#configuracion" data-icon="gear" data-role="button"
title="Configuración" data-iconpos="notext">Configuración</a>
 </header>
```

Fig. 5.5. Resultado del código del encabezado de la Web App con los estilos e íconos de jQuery Mobile aplicados.

Si deseamos que se ajusten cuando se utiliza algún control en pantalla completa podremos incluir `data-fullscreen="true"`.

Barras de navegación

Para crear las barras de navegación de la aplicación jQuery Mobile nos ofrece la posibilidad de usar `data-role="navbar"`. Dentro de la caja, luego podremos incluir listas con `<ul>` y elementos hijos `<li>`. Esta opción puede ser útil para crear la barra de navegación que incluimos en la parte inferior de nuestra aplicación.

Cuando creamos las barras de navegación podremos configurar también la posición de los íconos de los elementos que contienen con `data-iconpos`, que puede recibir el valor `left`, `right`, `top` o `bottom`. Por ejemplo, para posicionar los íconos en la parte superior de los botones de la barra lo establecemos de la siguiente forma:

```
<nav data-role="navbar" data-iconpos="top">
```

Para mostrar un elemento de la barra de navegación como activo, utilizamos `class="ui-btn-active"`. Es recomendable agregarle también al elemento la clase `ui-state-persist` para que retorne el estado activo al recargar el DOM. El marcado del elemento quedaría de la siguiente forma:

```
<li><a href="pagina2.html" class="ui-btn-active ui-state-
persist">Página 2</a>
```

Para nuestro proyecto, el pie conforma la barra de navegación principal de la aplicación y el código para conformarlo queda compuesto de la siguiente forma (deberíamos repetirlo en cada una de las pantallas de la aplicación):

```
<nav data-role="footer" data-position="fixed"  data-id ="pie">
 <div data-role="navbar">
  <ul>
      <li><a href="#eventos">Eventos</a>
      <li><a href="#noticias">Noticias</a>
      <li><a href="#notas">Notas</a>
      <li><a href="#redes">Redes</a>
  </ul>
 </div>
</nav>
```

Fig. 5.6. Representación de la barra de navegación inferior con los estilos de jQuery Mobile.

Diálogos y popups

Dentro de las características que permiten utilizar con facilidad jQuery Mobile, encontramos las ventanas de diálogos y *popups*.

Las ventanas de diálogo están definidas en jQuery Mobile también como parte del documento y se podrían ver como instancias de páginas, pero con algunas características diferentes. Para definir que una página es una ventana de diálogo, lo hacemos de la siguiente forma:

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

Si nos resulta conveniente para nuestro proyecto, también es posible hacer que la ventana de diálogo se encuentre en otro documento HTML.

Para invocar un diálogo desde un enlace, utilizamos `data-rel="dialog"` y es recomendable especificar una transición que proporcione la sensación de que se abre dicha ventana como un *pop-up*, como vemos en el código a continuación:

```
<a href="#dialogo" data-rel="dialog"
data-transition="pop">Acerca de…</a>
```

Para personalizar la opción de cierre de la ventana, podremos crear un ancla o un botón que le permita al usuario realizar este procedimiento. Desde el código HTML debemos llamar a la página que lo invocó y agregar `data-rel="back"`.

Si deseamos generar el cierre desde JavaScript, podremos usar el método `close()` que nos provee jQuery Mobile de la siguiente forma:

```
$('.ui-dialog').dialog('close')
```

Otra opción que puede resultar útil en nuestro proyecto es crear un *popup*. A diferencia de un diálogo, el *popup* no se presenta necesariamente como una ventana.

Un *popup* puede mostrar simplemente un *tooltip*, una lista de opciones, un formulario o hasta una ventana de diálogo. Por ejemplo, si necesitamos crear un *popup* con los vínculos para las redes sociales, podremos escribir un código como el siguiente:

```
<div data-role="popup" id="pop">
 <ul data-role="listview">
  <li data-role="divider">Redes sociales
  <li><a href="http://facebook.com">
       Facebook</a>
  <li><a href="http://twitter.com">
       Twitter</a>
  <li><a href="https://plus.google.com">
       Google Plus</a>
 </ul>
</div>
```

Para crear el *link* que llame a un *popup*, escribimos en el documento HTML un código como el siguiente:

```
<a href="#pop" data-rel="popup">Compartir</a>
```

Mientras que las páginas generan entradas en el historial, los diálogos y *popups* de jQuery Mobile no lo hacen. Esto es importante a la hora de manejar esta característica, en especial, cuando se desea retornar a la página anterior desde el código.

Transiciones

Una característica que destaca de las aplicaciones para móviles es el desplazamiento suave y elegante entre pantallas y ventanas. Por esta razón, las transiciones juegan un papel muy importante en el desarrollo de jQuery Mobile y en la programación que estamos preparando para nuestra Web App.

Las transiciones se establecen mediante `data-transition` y sus valores posibles son: `slide`, `slideup`, `slidedown`, `fade`, `pop` y `flip`. Por la complejidad de algunas de estas transiciones, la eficacia y fluidez con la que se pueda apreciar cada uno de estos efectos, dependerá del motor de *renderizado* del navegador o visualizador de turno en cada sistema operativo.

Si necesitamos que el efecto de transición se efectúe de manera inversa, podremos incluir el atributo `data-direction="reverse"`.

Configuración del Theme

Los temas de jQuery Mobile (*themes* en inglés) nos permiten establecer y personalizar el aspecto de nuestra Web App, por medio de la configuración de los elementos de la interfaz de usuario, para darle el aspecto de una aplicación móvil.

Para configurar el *theme*, contamos con `data-theme`, que puede recibir el valor de una letra que identifica al tema. Por defecto, contamos con el tema **a** hasta el **e**, pero podemos agregar nuevos temas identificados con las siguientes letras del abecedario.

Dentro de las opciones de cada tema, también tendremos la posibilidad de trabajar con **swatches** (literalmente "muestras" en español) que nos permiten definir combinaciones de colores para el tema.

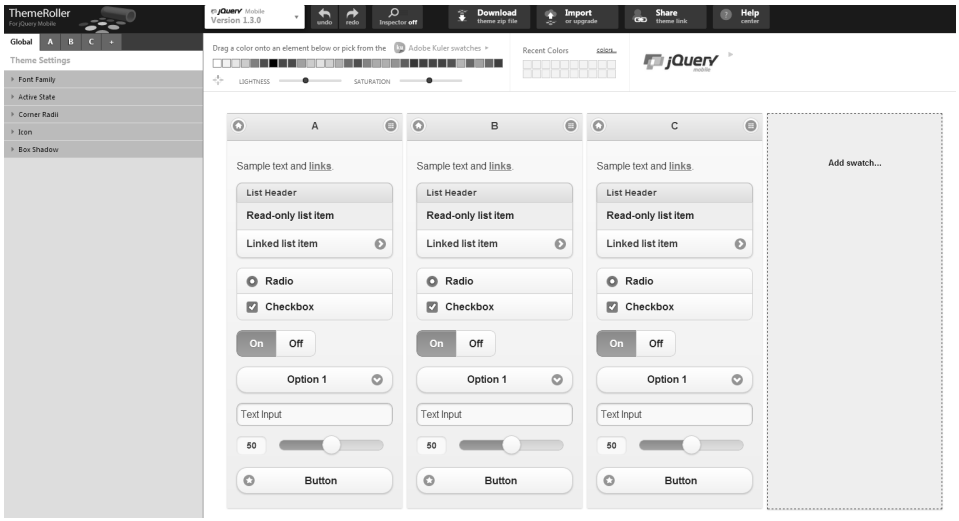

Fig. 5.7. Mediante ThemeRoller (http://jquerymobile.com/themeroller/) es posible crear temas para jQuery Mobile de una manera visual, sin tocar código.

Las características del *theme* se pueden agregar a cada uno de los elementos de nuestra aplicación, utilizando la propiedad `data-theme`, a la que le podremos asignar la letra del *theme* que deseemos. Por ejemplo, veamos de qué manera se le aplica un *theme* al encabezado de una página de jQuery Mobile:

```
<header data-role="header" data-position="fixed" data-id
="cabecera" data-theme="c">
```

Esta característica nos permite utilizar siempre el mismo *theme* para toda la aplicación o personalizar cada una de sus partes a nuestro gusto. Si no definimos un *theme*, como nos podremos imaginar, jQuery Mobile aplica uno predeterminado para cada componente.

Si deseamos modificar estas características por defecto, podremos incluir un *script* después de cargar jQuery y antes de cargar el *script* de jQuery Mobile. Para lograr esto, debemos recurrir al evento `mobileinit`, que se dispara cuando jQuery Mobile se inicia, y nos posibilita establecer los valores de las opciones de los elementos por defecto. A continuación, veremos un ejemplo que especifica nuevos valores por defecto para la página y la navegación, personalizando desde el inicio algunos de los elementos que los componen:

```
<script>
      $(document).bind("mobileinit", function () {
       // Navegación
```

```
      $.mobile.page.prototype.options.backBtnText
= "Regresar";
      $.mobile.page.prototype.options.addBackBtn
= true;
      $.mobile.page.prototype.options.backBtnTheme
= "c";
      // Pagina
      $.mobile.page.prototype.options.headerTheme
= "c";
      $.mobile.page.prototype.options.contentTheme
= "d";
      $.mobile.page.prototype.options.footerTheme
= "d";
            });
</script>
```

Formularios

Mediante HTML podemos crear y configurar controles de formulario. Gracias a jQuery Mobile lograremos la apariencia de una aplicación móvil para estos controles. Si bien son varias las opciones que nos ofrece jQuery Mobile para formularios, es importante tener en cuenta que, hasta el momento, este *framework* no cuenta con la opción de personalizar todos los nuevos elementos de entrada de formulario que incorpora HTML5. Por esta razón, en algunos casos como para fecha o color, se mostrará el control de entrada con su apariencia nativa en el navegador del dispositivo.

A continuación, recorreremos algunas de las principales características que jQuery Mobile pone a nuestra disposición para personalizar elementos de formulario.

Una alternativa de personalización que puede resultar muy útil para algunos proyectos es la posibilidad de mostrar los controles como "mini". Esto se puede lograr incorporando el atributo `data-mini="true"` al control (por ejemplo a `<input>`).

Un elemento que solemos encontrar en aplicaciones móviles es el *slider* que nos permite activar o desactivar una característica. Este elemento es muy fácil de configurar en jQuery Mobile, ya que lo podemos lograr indicando en la etiqueta `<select>` el atributo `data-role="slider"`. Veamos un ejemplo aplicado, a continuación:

```
<label for="notas">Habilitar notas:</label>
<select name="notas" data-role="slider">
      <option value="no">No</option>
      <option value="si">Si</option>
</select>
```

Para los radio *button*, la ventaja es que podremos agruparlos con un *fieldset* mediante `<fieldset data-role="controlgroup">`. De esta manera, podemos crear un código como el siguiente:

```
<fieldset data-role="controlgroup">
 <legend>Elige una opción:</legend>
  <label for="op1">Opción 1</label>
  <input type="radio" name="opcion" value="op1"
   checked="checked" />
  <label for="op2">Opción 2</label>
  <input type="radio" name="opcion" value="op2"/>
  <label for="op2">Opción 3</label>
  <input type="radio" name="opcion" value="op3"/>
</fieldset>
```

El uso de `data-role="controlgroup">` también es aplicable a *checkbox.*
Por defecto, la información del formulario, jQuery Mobile la puede enviar mediante AJAX, pero podremos programar otras opciones si así lo deseamos. Para deshabilitar el comportamiento de AJAX por defecto en el envío, podemos utilizar el atributo `data-ajax="false"`.

Si no deseamos que jQuery Mobile le aplique su propio estilo a un formulario y se vea con su representación por defecto en el navegador o visualizador del dispositivo, podremos utilizar el atributo `data-role="none"`, que se puede aplicar al formulario o al control que deseemos.

La estructura de nuestra Web App

Con todo el material que hemos visto hasta aquí en el capítulo, ya tenemos suficientes elementos para poder escribir la estructura de nuestra aplicación, utilizando las etiquetas HTML y las características que nos provee jQuery Mobile.

En la sección **El encabezado del documento**, vimos el código necesario para escribir la sección `<head>`, que conformará el documento HTML de la aplicación. Allí lo que deberemos modificar es la versión y ruta de los archivos de jQuery y jQuery Mobile.

También dentro de este capítulo, en la sección que lleva el título **Estructura de una aplicación con jQuery Mobile**, están detalladas las características fundamentales de esta librería, y también cómo se integran con un proyecto.

Si repasamos lo visto en el capítulo 2 del libro, cuando creamos las estructuras de "alambre" de nuestra aplicación y definimos las pantallas, encontraremos que tenemos que crear una pantalla principal, una pantalla para las noticias y otra para las notas. Esta última tendrá subpantallas para visualizar y crear notas. También encontramos otras opciones como las pantallas de configuración y acceso a redes sociales. A continuación, veremos en detalle cada una de ellas.

Página principal de la aplicación

La pantalla principal de nuestra aplicación contiene en la parte superior el encabezado que utilizaremos en todas las pantallas de la aplicación. Su creación fue detallada anteriormente, en la sección de **Toolbars** de jQuery Mobile, en este mismo capítulo. Contiene el ícono para regresar siempre a esta pantalla, el logo de la aplicación y el ícono de configuración. En este apartado, veremos cómo se integra con la página.

En la parte central de esta pantalla, tendremos en la sección de **Noticias**, la destacada o última noticia que se mostrará y permitirá acceder a la sección. A continuación, aparece la sección de **Eventos** que (en este caso) muestra el próximo evento a realizarse, y al pulsarlo lleva al usuario a la pantalla de eventos. Debajo, encontramos las novedades de redes sociales. En este caso se muestra una simulación con tuits de la red social Twitter.

La parte inferior de la pantalla está definida por el bloque de navegación de la aplicación, que se crea como hemos visto en el apartado **Barras de navegación**, de este capítulo.

Con estos elementos definidos, tenemos todo lo necesario para escribir la pantalla principal de la Web App. A continuación, vamos a ver de qué forma queda conformado el código completo de esta página:

```
<div data-role="page" id="principal">
 <header data-role="header" data-position="fixed"
 data-id ="cabecera">
  <a href="#principal" data-icon="home"
  data-role="button" title="Inicio"
  data-iconpos="notext" class="ui-btn-
```

```
  active">Inicio</a>
  <h1>Logo</h1>
  <a href="#configuracion" data-icon="gear"
  data-role="button" title="Configuración"
  data-iconpos="notext">Configuración</a>
 </header>
 <div data-role="content">
  <ul data-role="listview">
   <li data-role="list-divider">Noticias</li>
   <li><a href="#noticias"><img src="radio.png"
   width="64" height="64"/>
   <p>Lorem ipsum dolor sit amet, consectetuer
   adipiscing elit. Aenean commodo ligula eget do
   lor. Aenean massa. Cum sociis natoque
   penatibus et magnis dis parturient montes,
   nascetur ridiculus mus.</p></a></li>
  </ul>
   <ul data-role="listview">
    <li data-role="list-divider">Eventos</li>
    <li><a href="#eventos"><img src="mundo.png"
    width="64" height="64"/>
      <p>Lorem ipsum dolor sit amet, consectetuer
     adipiscing elit. Aenean commodo ligula eget
     dolor. Aenean massa. Cum sociis natoque pe
     natibus et magnis dis parturient montes,
     nascetur ridiculus mus.</p></a></li>
    </ul>
<!-- Colocar aquí Widget o codigo para mostrar contenido de redes
sociales -->
    <ul data-role="listview">
     <li data-role="list-divider">
      Redes sociales</li>
     <li><a href="#"><img src="twitter.png"
     width="64" height="64"/>
     <p>Lorem ipsum dolor sit amet, consectetuer
     adipiscing elit. Aenean commodo ligula eget
     dolor.</p></a></li>
     <li><a href="#"><img src="twitter.png"
```

```
     width="64" height="64"/>
     <p>Lorem ipsum dolor sit amet, consectetuer
     adipiscing elit. Aenean commodo ligula eget
     dolor.</p></a></li>
     <li><a href="#"><img src="twitter.png"
     width="64" height="64"/>
     <p>Lorem ipsum dolor sit amet, consectetuer
     adipiscing elit. Aenean commodo ligula eget
     dolor.</p></a></li>
    </ul>
  </div>
  <nav data-role="footer" data-position="fixed"
  data-id ="pie">
   <div data-role="navbar">
    <ul>
     <li><a href="#eventos">Eventos</a>
     <li><a href="#noticias">Noticias</a>
     <li><a href="#notas">Notas</a>
     <li><a href="#redes">Redes</a>
    </ul>
      </div>
 </nav>
</div>
```

Para obtener el código que nos permite integrar características de redes sociales en nuestra Web App, debemos recurrir a las opciones que ofrecen para desarrolladores. Twitter permite embeber tuits y *timelines* (https://dev.twitter.com/), mientras que Facebook cuenta con *plugins* sociales (http://developers.facebook.com/docs/plugins/) y un SDK para JavaScript (http://developers.facebook.com/docs/reference/javascript/), entre otras opciones.

Fig. 5.10. Pantalla principal de la aplicación vista en un móvil.

El texto aplicado en las secciones de **Noticias**, **Eventos** y **Redes sociales** se ha creado con una herramienta de simulación de texto (una muy buena opción es la que ofrece http://www.blindtextgenerator.com/es). Al llevar el proyecto a la práctica, el texto real puede cargarse directamente en el HTML u obtenerse de otras fuentes, como pueden ser bases de datos o almacenamiento local (tal como vimos en el capítulo anterior) o mediante otras técnicas que permitan conseguir datos desde servidores o recursos que estén en *online*.

Tanto la incorporación de la cabecera como la barra de navegación en la parte inferior de la aplicación se repetirán en las siguientes páginas. En cada caso, se deberá identificar el ícono de la sección activa especificando la clase `ui-btn-active` y agregando también la clase `ui-state-persist`, cuando se trata de un *navbar*.

En este caso, y en todas las pantallas de la aplicación, deberemos asegurarnos de que las imágenes se encuentren en las rutas correspondientes y con los nombres tal cual están escritos en el código.

Página *Noticias*

En nuestro modelo, la pantalla de **Noticias** está limitada a una cantidad y utiliza un modelo de paginado, pero también podríamos optar por incluir las noticias que deseemos y permitir que el usuario se desplace. Más adelante, al final del capítulo, veremos opciones de *plugins* que nos permitirán hacer el paginado y manejar todo lo referente al *scroll*, tanto de esta pantalla como para el resto de la aplicación.

A continuación, nos vamos a centrar en el código de la página de noticias:

```
<div data-role="page" id="noticias">
  <header data-role="header"
  data-position="fixed" data-id ="cabecera">
   <a href="#principal" data-icon="home" data-
   role="button" title="Inicio"
   data-iconpos="notext">Inicio</a>
   <h1>Logo</h1>
   <a href="#configuracion" data-icon="gear"
   data-role="button" title="Configuración"
   data-iconpos="notext">Configuración</a>
  </header>
  <div data-role="content">
   <ul data-role="listview">
    <li data-role="list-divider">Noticias</li>
    <li><a href="#"><img src="radio.png"
    width="64" height="64"/>
    <p>Lorem ipsum dolor sit amet, consectetuer
    adipiscing elit…</p></a></li>
    <li><a href="#"><img src="radio.png"
    width="64" height="64"/>
    <p>Lorem ipsum dolor sit amet, consectetuer
    adipiscing elit…</p></a></li>
    <li><a href="#"><img src="radio.png"
    width="64" height="64"/>
    <p>Lorem ipsum dolor sit amet, consectetuer
    adipiscing elit…</p></a></li>
   </ul>
  </div>
 <nav data-role="footer" data-position="fixed"
 data-id ="pie">
```

```
  <div data-role="navbar">
   <ul>
    <li><a href="#eventos">Eventos</a>
    <li><a href="#noticias" class="ui-btn-active
    ui-state-persist">Noticias</a>
    <li><a href="#notas">Notas</a>
    <li><a href="#redes">Redes</a>
   </ul>
  </div>
 </nav>
</div>
```

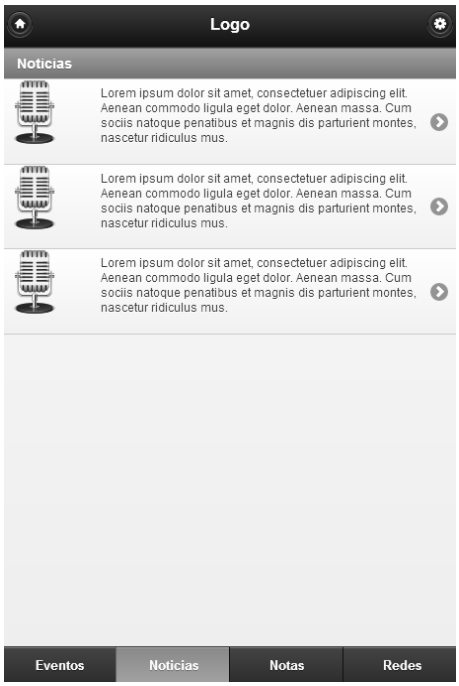

Fig. 5.11. Pantalla de la sección *Noticias* de la aplicación vista en un móvil.

Vale la pena recordar que el código que estamos escribiendo se encuentra en un mismo documento HTML, al que podremos llamar **index.html**. Cada elemento contenedor (el `<div>` en este caso) que está identificado como `data-role="page"` conforma el bloque de la página. jQuery Mobile mostrará solamente el primero. Al pulsar sobre un enlace o llamar otra página, se mostrará el que

corresponda y se ocultará el resto. Todos estos bloques que conforman páginas de jQuery Mobile pueden encontrarse uno debajo del otro en el mismo documento.

Página *Eventos*

La sección **Eventos** en nuestra aplicación tiene una estructura muy similar a la que definimos en la página de **Noticias**, la diferencia entre ambas es, principalmente, el contenido de cada una de ellas. A continuación, veremos el código para la página de eventos de nuestra Web App:

```
<div data-role="page" id="eventos">
 <header data-role="header" data-position="fixed"
 data-id ="cabecera">
 <a href="#principal" data-icon="home"
 data-role="button" title="Inicio"
 data-iconpos="notext">Inicio</a>
  <h1>Logo</h1>
  <a href="#configuracion" data-icon="gear"
  data-role="button" title="Configuración"
  data-iconpos="notext">Configuración</a>
 </header>
 <div data-role="content">
   <ul  data-role="listview">
    <li data-role="list-divider">Eventos</li>
    <li><a href="#"><img src="mundo.png"
    width="64" height="64"/>
    <p>Lorem ipsum dolor...</p></a></li>
    <li><a href="#"><img src="mundo.png"
    width="64" height="64"/>
    <p>Lorem ipsum dolor...</p></a></li>
    <li><a href="#"><img src="mundo.png"
    width="64" height="64"/>
    <p>Lorem ipsum dolor...</p></a></li>
   </ul>
 </div>
 <nav data-role="footer" data-position="fixed"  data-id ="pie">
  <div data-role="navbar">
   <ul>
```

```
    <li><a href="#eventos" class="ui-btn-active ui-state-
persist">Eventos</a>
    <li><a href="#noticias">Noticias</a>
    <li><a href="#notas">Notas</a>
    <li><a href="#redes">Redes</a>
   </ul>
  </div> </nav> </div>
```

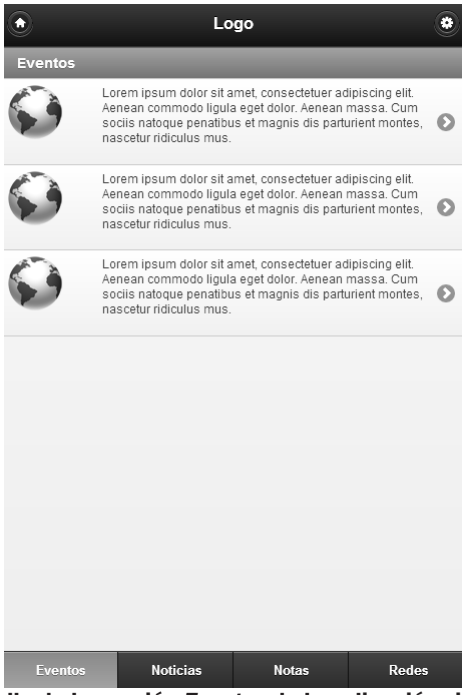

Fig. 5.12. Pantalla de la sección *Eventos* de la aplicación vista en un móvil.

Páginas de *Notas*

En nuestra aplicación, la sección **Notas** le permite al usuario crear y leer notas personales. Cuando el usuario accede, desde el botón **Notas**, de manera predeterminada llega a la pantalla que le permite leer las notas guardadas. Cada nota tiene una sección, un texto y un enlace. Desde allí se puede acceder también a la opción de crear notas. El código de esta página es el que observamos a continuación:

```
<div data-role="page" id="notas">
 <header data-role="header"
  data-position="fixed" data-id ="cabecera">
  <a href="#principal" data-icon="home"
  data-role="button" title="Inicio"
  data-iconpos="notext">Inicio</a>
   <h1>Logo</h1>
   <a href="#configuracion" data-icon="gear"
   data-role="button" title="Configuración"
   data-iconpos="notext">Configuración</a>
   <div data-role="navbar">
     <ul>
      <li><a href="#notas" class="ui-btn-active
      ui-state-persist">Leer notas</a>
      <li><a href="#crearnota">Crear una nota</a>
    </ul>
   </div>
 </header>
 <div data-role="content">
   <ul data-role="listview">
     <li>
      <h3>Categoría 1</h3>
      <p>Lorem ipsum dolor sit amet…</p>
      <p><a href="htttp://www.sitio1.com">
      htttp://www.sitio1.com</a></p></li>
     <li>
      <h3>Categoría 2</h3>
      <p>Lorem ipsum dolor…</p>
      <p><a href="htttp://www.sitio2.com">
      htttp://www.sitio2.com</a></p></li>
   </ul>
</div>
 <nav data-role="footer" data-position="fixed"
 data-id ="pie">
  <div data-role="navbar">
   <ul>
    <li><a href="#eventos">Eventos</a>
    <li><a href="#noticias">Noticias</a>
```

```
    <li><a href="#notas" class="ui-btn-active
     ui-state-persist">Notas</a>
    <li><a href="#redes">Redes</a>
   </ul>
  </div>
 </nav>
</div>
```

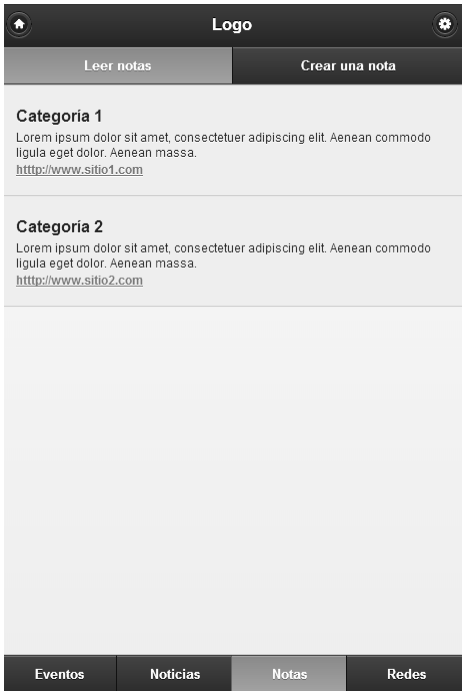

Fig. 5.13. Pantalla *Notas*, situada en la opción Leer notas.

Cuando el usuario pulsa la opción **Crear una nota** (arriba, en la parte superior de la ventana) se pasa a una nueva pantalla. Allí, el usuario puede indicar una categoría (que cuenta con un botón que permite acceder a la pantalla de agregar o eliminar categorías), escribir la nota personal, especificar una URL y le ofrece un botón para guardar la nota. El código para definir la estructura de la ventana de ceración de notas es el que vemos a continuación:

```
<div data-role="page" id="crearnota">
 <header data-role="header" data-position="fixed"
```

```
data-id ="cabecera">
  <a href="#principal" data-icon="home"
  data-role="button" title="Inicio"
  data-iconpos="notext">Inicio</a>
  <h1>Logo</h1>
  <a href="#configuracion" data-icon="gear"
  data-role="button" title="Configuración"
  data-iconpos="notext">Configuración</a>
  <div data-role="navbar">
  <ul>
    <li><a href="#notas">Leer Notas</a>
    <li><a href="#crearnota"
    class="ui-btn-active ui-state-persist">
    Crear una nota</a>
   </ul>
  </div>
 </header>
<div data-role="content">
<form id="formnota" data-ajax="false"
 method="get">
  <label>Categoría</label>
   <fieldset class="ui-grid-a">
   <div class="ui-block-a">
    <select name="categoria">
     <option value="opcion1">Categoría 1
     </option>
     <option value="opcion2">Categoría 2
     </option>
     <option value="opcion2">Categoría 3
     </option>
    </select>
   </div>
   <div class="ui-block-b"><a href="#categoria"
   data-role="button">Agregar /
   Elminar</a></div>
   </fieldset>
   <label for="nota">Nota</label>
   <textarea id="nota" name="nota">
   </textarea>
```

```
    <label for="url">URL</label>
    <input type="url" id="url" name="url"/>
    <div class="ui-block-b"><input type="submit"
     value="Guardar nota" id="guardarnota"/>
    </div>
   </form>
  </div>
 <nav data-role="footer" data-position="fixed"
 data-id ="pie">
  <div data-role="navbar">
    <ul>
     <li><a href="#eventos">Eventos</a>
     <li><a href="#noticias">Noticias</a>
     <li><a href="#notas" class="ui-btn-active
      ui-state-persist">Notas</a>
     <li><a href="#redes">Redes</a>
    </ul>
   </div>
 </nav>
</div>
```

Nótese que el formulario tiene fijado el método `get` y, demás, cuenta con el atributo `data-ajax="false"`. Estas opciones fijadas de este modo nos permitirán, como veremos más adelante, obtener los datos cuando el usuario presione el botón `submit` y serializarlos utilizando jQuery.

Fig. 5.14. Pantalla Notas, situada en la opción Crear una nota.

El botón **Agregar/Eliminar** permite que el usuario acceda a una pantalla en la cual podrá agregar una nueva categoría o dar de baja una existente. De esta manera, la página de alta y baja de categorías queda definida con la siguiente estructura:

```
<div data-role="page" id="categoria">
 <header data-role="header" data-position="fixed"
 data-id ="cabecera">
  <a href="#principal" data-icon="home"
  data-role="button" title="Inicio"
  data-iconpos="notext">Inicio</a>
  <h1>Logo</h1>
  <a href="#configuracion" data-icon="gear"
  data-role="button" title="Configuración"
  data-iconpos="notext">Configuración</a>
  <div data-role="navbar">
   <ul>
    <li><a href="#notas">Leer Notas</a>
```

```
    <li><a href="#crearnota" class="ui-btn-active
    ui-state-persist">Crear una nota</a>
   </ul>
  </div>
 </header>
 <div data-role="content">
  <form>
   <label>Agregar categoría</label>
   <input type="text" id="agregacatagoria"/>
   <input type="button" id="agregacatagoriabtn"
   name="agregacatagoriabtn" value="Agregar
   categoría"/>
  </form>
  <ul data-role="listview" data-icon="delete"
  data-inset="true" >
   <li data-role="list-divider">Categorías</li>
   <li id="categoria1"><a href="#">
   Categoría 1</a></li>
   <li id="categoria2"><a href="#">
   Categoría 2</a></li>
   <li id="categoria3"><a href="#">
   Categoría 3</a></li>
  </ul>
   <a href="#crearnota" data-role="button">
   Regresar a crear nota</a>
 </div>
 <nav data-role="footer" data-position="fixed"
 data-id ="pie">
  <div data-role="navbar">
   <ul>
     <li><a href="#eventos">Eventos</a>
     <li><a href="#noticias">Noticias</a>
     <li><a href="#notas" class="ui-btn-active ui-state-
persist">Notas</a>
     <li><a href="#redes">Redes</a>
    </ul>
   </div> </nav> </div>
```

Fig. 5.15. Pantalla *Notas*, situada en la opción *agregar o eliminar categorías*.

En la dirección Web http://forum.jquery.com/jquery-mobile, se encuentra el foro dedicado a jQuery Mobile. Allí es posible encontrar hilos con los inconvenientes más frecuentes con los que se han enfrentado los usuarios. Podremos plantear nuestros problemas para ser ayudados por la comunidad a este framework.

Redes sociales

En esta ventana se muestran los enlaces a las reces sociales que están vinculadas a la aplicación. En este caso, nuestro proyecto se vincula a Facebook, Twitter, Google+ y Flickr, pero estas opciones se pueden cambiar y/o modificar de acuerdo con el proyecto.

Esta página cuenta con en el código que sigue a continuación:

```
<div data-role="page" id="redes">
  <header data-role="header"
  data-position="fixed" data-id ="cabecera">
   <a href="#principal" data-icon="home"
```

```
  data-role="button" title="Inicio" data-
  iconpos="notext">Inicio</a>
  <h1>Logo</h1>
  <a href="#configuracion" data-icon="gear"
  data-role="button" title="Configuración" data-
  iconpos="notext">Configuración</a>
 </header>
<div data-role="content">
  <ul data-role="listview">
     <li data-role="list-divider">
     Redes Sociales</li>
     <li><a href="#">
    <img src="social_facebook.png"
     alt="Facebook" class="ui-li-icon">
     Facebook</a></li>
     <li><a href="#">
    <img src="social_twitter.png"
    alt="Twitter" class="ui-li-icon">
    Twitter</a></li>
    <li><a href="#">
    <img src="social_googleplus.png"
    alt="Google Plus" class="ui-li-icon">
    Google+</a></li>
    <li><a href="#">
    <img src="social_flickr.png" alt="Flickr"
    class="ui-li-icon">Flickr</a></li>
     <li><a href="#"><img
    src="social_youtube.png" alt="YouTube"
    class="ui-li-icon">YouTube</a></li>
  </ul>
 </div>
<nav data-role="footer" data-position="fixed"
 data-id ="pie">
  <div data-role="navbar">
     <ul>
      <li><a href="#eventos">Eventos</a>
      <li><a href="#noticias">Noticias</a>
      <li><a href="#notas">Notas</a>
      <li><a href="#redes"
```

```
      class="ui-btn-active ui-state-persist">
      Redes</a>
     </ul>
    </div>
  </nav>
</div>
```

En el código, dentro del contendor definido con `data-role="content"`, el atributo `href` de `<a>` de los elementos de la lista de redes sociales está referenciado a #. Al aplicarlo a un proyecto real, el símbolo # debería ser reemplazado por la dirección URL de la red social correspondiente, referenciando al usuario o página vinculada con la aplicación o servicio que se está ofreciendo.

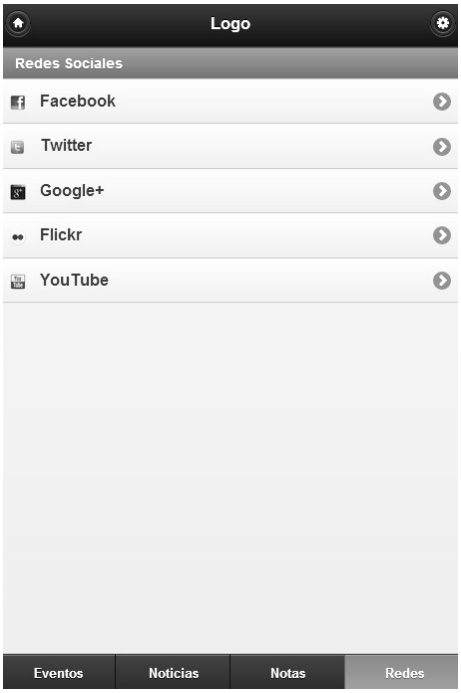

Fig. 5.16. Vista de la pantalla de *Redes sociales en el móvil*.

Dentro de la etiqueta `<img>` la referencia especificada en el atributo `src` corresponde a íconos prediseñados que puede crear un diseñador u obtenerse mediante *packs* de Internet. Es importante verificar el nombre y la ruta de la imagen, dentro del proyecto, para asegurarse que está correctamente referenciada.

Página *Configuración*

Para definir la estructura y los elementos que utilizaremos en la página de **Configuración** de nuestra aplicación nos será muy útil lo aprendido en el apartado dedicado a formularios, en este capítulo, y también lo que explicado en el capítulo anterior, relacionado con las nuevas características que introduce HTML5.

En esta pantalla de configuración, el usuario tendrá la posibilidad de modificar la combinación de colores para la aplicación, elegir una ubicación geográfica y borrar las notas que haya creado. Además, desde aquí podrá acceder a las ventanas de diálogo de ayuda y acerca de. El código para esta página es el que vemos a continuación:

```
<div data-role="page" id="configuracion">
 <header data-role="header"
 data-position="fixed" data-id ="cabecera">
   <a href="#principal" data-icon="home"
   data-role="button" title="Inicio"
   data-iconpos="notext">Inicio</a>
   <h1>Logo</h1>
   <a href="#configuracion" data-icon="gear"
   data-role="button" title="Configuración"
   data-iconpos="notext"
   class="ui-btn-active">Configuración</a>
 </header>
 <div data-role="content">
  <form>
   <label>
   Combinación de colores:</label>
   <fieldset  class="ui-grid-a">
   <div class="ui-block-a">
    <select name="tema">
     <option value="1">Opción 1</option>
     <option value="2">Opción 2</option>
     <option value="3">Opción 3</option>
    </select>
   </div>
   <div class="ui-block-b">
   <input type="button" value="Guardar"
```

```
    id="guardartheme"/></div>
   </fieldset>
   <label>
   Ubicación Geográfica: </label>
   <fieldset  class="ui-grid-a">
    <div class="ui-block-a">
     <select>
      <option value="pais1">Pais 1</option>
      <option value="pais2">Pais 2</option>
     <option value="pais3">Pais 3</option>
    </select>
    </div>
    <div class="ui-block-b"><input type="button"
    value="Guardar" id="guardarpais"/></div>
    </fieldset>
 </form>
 <a href="#" data-role="button">Borrar todas
 las notas</a>
 <a href="#ayuda" data-role="button"
 data-rel="dialog" data-transition="pop">
  Ayuda</a>
 <a href="#acercade" data-role="button"
 data-rel="dialog" data-transition="pop">
 Acerca de</a>
</div>
<nav data-role="footer" data-position="fixed"
 data-id ="pie">
  <div data-role="navbar">
  <ul>
    <li><a href="#eventos">Eventos</a>
    <li><a href="#noticias">Noticias</a>
    <li><a href="#notas">Notas</a>
    <li><a href="#redes">Redes</a>
   </ul>
 </div> </nav> </div>
```

Fig. 5.17. Pantalla de *Configuración* de la aplicación.

Ventanas *Ayuda* y *Acerca de*

Desde la página de configuración, el usuario puede abrir las ventanas **Ayuda** y **Acerca de**. Ambas están definidas en el código como diálogos y su estructura es muy similar.

A continuación, observamos el código para la ventana de ayuda:

```
<div data-role="dialog" id="ayuda">
 <div data-role="header">
  <h1>Ayuda</h1>
 </div>
 <div data-role="content">
  <h2>Cómo utilizar esta aplicación</h2>
  <p>Lorem ipsum dolor sit amet</p>
  <p>Aenean commodo ligula eget dolor.</p>
 </div>
</div>
```

Fig. 5.18. Ventana de *Ayuda* vista en el móvil.

El código para crear la ventana **Acerca de**, es el que sigue a continuación:

```
<div data-role="dialog" id="acercade">
 <div data-role="header">
   <h1>Acerca de</h1>
  </div>
  <div data-role="content">
   <h2>Acerca de esta aplicación</h2>
   <p>Aplicación desarrollada por @damiande.</p>
   <p>Buenos Aires, Argentina – Año 2013</p>
  </div>
</div>
```

Fig. 5.19. Ventana de *Acerca de* vista en el móvil.

Es importante señalar que los enlaces que invocan estas ventanas desde la página **Configuración** cuentan con el atributo `data-rel="dialog"`.

Estilos personalizados

Cuando trabajamos con jQuery Mobile, además de aprovechas los estilos y efectos que contiene la librería, también podremos personalizar estas características mediante CSS y JavaScript.

Al trabajar con estilos CSS tenemos la posibilidad de personalizar la manera en que se muestran los elementos, incluso sobrescribiendo algunos de los especificados por defecto en jQuery Mobile.

Por ejemplo, de manera predeterminada, los elementos de listas de las secciones de noticias y eventos de la pantalla principal de nuestra aplicación se muestran con puntos suspensivos (...) como vemos en la siguiente figura.

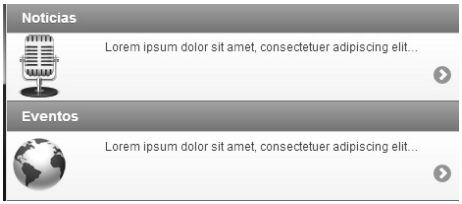

Fig. 5.20. Representación de los estilos de listas de *Noticias* y *Eventos* en la página principal de la aplicación.

En nuestro modelo, optamos porque el texto fluya y no se muestren puntos suspensivos al final, indicando que hay más texto que no puede verse. Para lograr que el texto se muestre de manera completa, debemos introducir unas líneas de CSS para modificar la representación. Luego de incluir las librerías de jQuery Mobile, en la sección de **Estilos** de nuestro documento HTML (o en un documento CSS externo), incluimos las siguientes líneas para lograr el resultado visual deseado:

```
.ui-li-desc{
 text-overflow:ellipsis;
 overflow:visible;
 white-space:normal;}
```

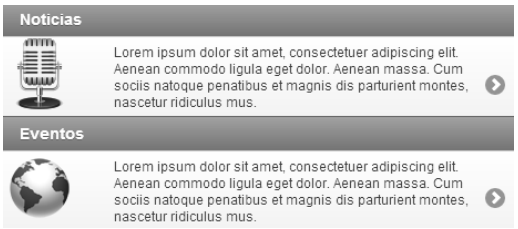

Fig. 5.21. Resultado de la representación de las listas de *Noticias* y *Eventos* en la página principal de la aplicación, luego de aplicar el estilo personalizado.

HTML5 y JavaScript en nuestra aplicación

Con jQuery Mobile es posible crear una estructura de aplicación con varias funcionalidades muy útiles y la apariencia de una aplicación móvil, prácticamente, sin necesidad de meternos en JavaScript. Sin embargo, para lograr que nuestra aplicación pueda ofrecer todas las funcionalidades que proyectamos, necesitaremos involucremos en el uso de JavaScript. En este apartado, veremos cómo programar los comportamientos de los formularios que creamos en la estructura HTML, y cómo guardar datos persistentes en el equipo del usuario.

En el capítulo 3 de este libro conocimos las principales novedades que se introducen mediante las API de HTML5. Para nuestro proyecto, vamos a recurrir a LocalStorage, característica que nos permitirá guardar datos en el equipo del usuario. Para realizar estas operaciones también utilizaremos JavaScript, jQuery y técnicas relacionadas con AJAX. Por esta razón, si el lector tiene dudas sobre cómo se trabaja con estas características, es recomendable que antes de avanzar en este apartado, lea el apéndice dedicado a este tema.

A continuación, veremos las secciones de la aplicación que requieren interacción con las API de HTML5 y JavaScript para analizar cómo resolver cada necesidad en el código.

Obtener el valor de un campo y almacenarlo de manera local

En la pantalla de configuración, tenemos dos controles de formulario del tipo *select*. Uno de ellos permite guardar una combinación de colores para personalizar la apariencia de la aplicación, y el otro da la opción de establecer nuestra ubicación geográfica.

En cada caso, necesitamos obtener el valor del campo *select* al presionar el botón **Guardar** y almacenar el valor en una clave en el LocalStorage. En este proyecto, los temas (combinación de colores) estarán predefinidos desde el código y el usuario podrá elegir uno de los existentes, pero no podrá crear nuevos.

Combinación de colores

Para la combinación de colores el control *select* tiene asignado un atributo `id="tema"`, y el botón **Guardar**, que lanza la acción al ser pulsado por el usuario tiene una `id` cuyo valor es `guardartheme`.

Mediante jQuery podremos "escuchar" cuando el usuario pulsa un botón de la siguiente manera:

```
$('#guardartheme').click(function() {
   //Código
});
```

En la sección donde escribimos el comentario para el código, en primero lugar, debemos obtener el valor del campo, que en este caso lo haremos con jQuery recurriendo a al método `val()` y lo asignamos a una variable en JavaScript, de la siguiente forma:

```
var tema = $("#tema").val();
```

A continuación, dentro de la misma función, guardamos ese valor LocalStorage:

```
localStorage.setItem("op_tema", tema);
```

En la línea de código anterior, `op_tema` es el nombre de la clave que almacenamos en LocalStorage y `tema` es el nombre de la variable JavaScript que creamos anteriormente.

El código completo queda conformado de la siguiente forma:

```
$('#guardartheme').click(function() {
 var tema = $("#tema").val();
 localStorage.setItem("op_tema", tema);
});
```

De la misma manera, actuamos con el campo de ubicación geográfica. En este caso, debemos cambiar el valor de la *id* del elemento que puede ser pulsado por el usuario (`guardarpais`), el nombre de la variable, el identificador del control s*elect* (`ubicacion`) y el nombre de la clave en LocalStorage (`gelolalizacion`), como vemos en el siguiente código:

```
$('#guardarpais').click(function() {
  var geo = $("#ubicacion").val();
```

```
 localStorage.setItem("gelocalizacion", geo);
});
```

El valor que se ha almacenado se mantendrá de manera persistente en la aplicación, aun cuando el usuario la cierre y la vuelva abrir.
El paso siguiente será modificar la apariencia de la aplicación según la elección del usuario. ¿Cómo lo logramos? En la sección **Configuración del Theme**, en este capítulo, aprendimos a configurar un *theme* por defecto al iniciarse la aplicación. En este caso, al recurrir a `mobileinit` primero debemos recuperar la clave almacenada, y luego aplicar la combinación elegida, según su valor. Para recuperar el valor de la clave guardada en LocalStorate recurrimos a la siguiente línea de código:

```
var temalocal = localStorage.getItem("op_tema");
```

Con este valor obtenido, será posible plantear una estructura de control (*Select-Case*) que permita fijar el tema seleccionado:

```
switch (temalocal)
{
case 1:
  //Código de la combinación del Tema 1
  break;
case 2:
  //Código de la combinación del Tema 1
  break;
case 3:
   //Código de la combinación del Tema 1
  break;
}
```

Se pueden agregar tantas combinaciones como se le deseen ofrecer al usuario, y en lugar del comentario se deben especificar las opciones del *theme* que recibirá cada elemento de la aplicación que será personalizado en el tema elegido.

Ubicación geográfica

En nuestra Web App, el dato de ubicación geográfica que indique el usuario se mantendrá almacenado como información que podrá ser utilizada posteriormente en futuras actualizaciones que reciba la aplicación. Por ejemplo, este dato puede resultar útil para filtrar eventos o novedades según la ubicación geográfica del usuario. Vale la pena señalar que los países disponibles, para este proyecto,

deberán estar precargados en la estructura HTML, pero también pueden utilizarse conjuntos de datos obtenidos desde JavaScript u otras fuentes para mantener un conjunto de países actualizados.

Borrar todas las notas

Si observamos, en esta pantalla de configuración también se encuentra otro botón que requiere programación: **Borrar todas las notas**. La opción más sencilla que puede pasar por nuestra mente es escribir una función que borre todos los datos guardados en LocalStorage, mediante el método `localStorage.clear()`. Sin embargo, recurrir a una opción como ésa también borraría los datos almacenados en las claves destinadas a la configuración del tema y la ubicación del usuario. ¿Cuál es la solución entonces? Para encontrar una alternativa adecuada para esta necesidad, primero analizaremos cómo guardar las notas en la aplicación, y luego develaremos cómo conviene borrarlas.

Guardar, leer y borrar notas del usuario

La idea detrás de las notas es que el usuario pueda armar su propio espacio para almacenar apuntes o bien pueda armar su propia agenda dentro de la aplicación. Las categorías también estarán cargadas por el usuario y para esto debemos planear como funcionará este bloque.

A continuación, comenzaremos primero por definir las categorías que tendrán disponibles las notas del usuario y cómo podrán darse de alta.

Las categorías de notas

En primer lugar, creamos en JavaScript un nuevo *array* que tendrá las categorías:

```
var categorias=new Array();
```

Para asignar valores en este *array*, debemos esperar a que el usuario complete el campo identificado como `agregacatagoria` y pulse el botón que tiene la id con el valor `agregacatagoriabtn`. Cuando esto ocurre, se recoge mediante el código el valor del campo de texto y sobre el *array* `categorías`, se aplica el método `push()` de JavaScript, el que recibe el valor del campo que cargó el usuario para especificar la categoría. Luego el *array* se debe convertir en una cadena para así ser guardado en LocalStorage. El código completo de este bloque queda de la siguiente manera:

```
$('#agregacatagoriabtn').click(function() {
  var categorias=new Array();
  var categoria = $("#agregacatagoria").val();
```

```
  categorias.push(categoria);
  categorias.toString();
  localStorage.setItem("cat", categorias);
});
```

El código anterior estaría bien para el estado inicial de la aplicación, cuando no hay categorías cargadas y el usuario comienza la carga ¿pero qué ocurre si ya existen categorías almacenadas en el equipo local? En este caso, deberíamos tratar de obtener el valor de la clave `cat` en LocalStorage. Esto lo podemos lograr mediante un código como el siguiente:

```
var categorias = localStorage.getItem("cat");
if(categorias){console.log(categorias)}
else{console.log("No hay categorias")};
```

En el ejemplo del código anterior, los resultados se imprimen en la consola, pero al llevarlos al proyecto deberíamos integrarlos con el resto de la función. Vale la pena destacar que esa construcción que vemos en el código anterior también nos permite recuperar los valores guardados en la clave para luego mostrarlos en las vistas de categorías, tanto en la lista para borrar categorías de esta pantalla como en las opciones para mostrar categorías en el s*elect* de notas.
Si existe la clave, podemos convertir su valor en un array, ya que para almacenarla la transformamos previamente en un *string*. Para lograr esto recurrimos al método `split()` de JavaScript. Este método debe ser aplicado a la variable que contiene la cadena y le podremos pasar el separador que divide a los valores del *array*, en este caso una coma. El código sería el siguiente:

```
categorias.split(",")
```

Posteriormente, mediante un ciclado, utilizando *for* en JavaScript o el método `each()` (http://api.jquery.com/jQuery.each/) en jQuery, es posible obtener los elementos del *array* y mostrarlos en pantalla o colocarlos dentro del elemento correspondiente en el DOM.

Las notas

Para las notas del usuario, al tener que guardar una colección de datos, podíamos pensar una alternativa que permita almacenar esta información en una base de datos local, con las opciones que nos ofrecen SQL Database o IndexedDB en HTML5.

Debido a que la estructura de datos no reviste complejidad y no tendremos que realizar otros procesos complejos como búsquedas o modificaciones constantes, trabajaremos con el formato JSON y almacenaremos los valores en LocalStorage.

La estructura que vamos a definir para el JSON que contendrá las notas es la que se ejemplifica en el siguiente código:

```
var notas = [
{categoria: "2", nota: "Lorem ipsumdolor sit amet, consectetuer
adipiscing elit.",
url: "http://www.sitio1.com"}
];
```

El código anterior tiene valores de ejemplo para representar la estructura. Los valores reales se obtienen de los campos del formulario `formnota`, que deberá ser completado por el usuario.

Para obtener los valores del formulario, podemos recurrir a jQuery, empleando el método `serializeArray()`. La documentación de este método se encuentra disponible en la dirección Web http://api.jquery.com/serializeArray/. El código que debemos escribir es el siguiente:

```
$('#formunotas').submit(function() {
    var jsonNota = {};
    var arraySerializado = $('#formunotas').serializeArray();
    $.each(arraySerializado, function() {
      jsonNota[this.name] = this.value || '';
    });
    return jsonNota;
});
```

Las notas las almacenaremos en una clave de LocalStorage a la que denominaremos `notas`. Pero antes de guardar el nuevo valor debemos verificar si hay notas anteriores y recuperarlas:

```
if (localStorage.getItem("notas")) {

var notasStorage = localStorage.getItem("notas");

}
```

Si encontramos que existían notas guardadas, luego de recuperar el valor de la clave como vimos en el código anterior, recurrimos a la función `toJSON()`, que transforma la cadena nuevamente en un objeto JSON para poder trabajarlo, como vemos a continuación:

```
var notasStorage = toJSON(notasStorage);
```

Luego, deberemos unir las notas en un mismo JSON antes de guardarlas. Este paso lo podemos hacer mediante el método `concat()` de JavaScript. En nuestro ejemplo el código sería el siguiente:

```
var todasNotas =  notasStorage.concat(jsonNota);
```

Ahora, para almacenar los valores en LocalStorage, primero deberemos convertir el objeto JSON en una cadena JSON. Para esto encontramos la función `JSON.stringify()`:

```
var notasString = JSON.stringify(todasNotas);
```

En resumen, los pasos que realizamos son: obtener los valores del formulario ingresados por el usuario; verificar si hay notas guardadas en LocalStorage y recuperarlas; convertir el valor recuperado a objeto JSON y concatenarlo con la información obtenida del formulario; convertir la variable concatenada en una cadena JSON y almacenar el valor nuevamente en la clave destinada para las notas en LocalStorage.

Para finalizar con este tema, como hemos podidos observar en este apartado, las notas quedan guardadas en una única clave, razón por la cual no es necesario trabajar con método `localStorage.clear()` para borrarlas todas, sino que simplemente, el botón **Borrar todas las notas** de la pantalla de configuración de la aplicación debe llamar a un código como el siguiente:

```
localStorage.removeItem("notas");
```

jQuery Mobile API

Como ya hemos mencionado anteriormente, una de las grandes ventajas de jQuery Mobile es permitir "hacer mucho" sin necesidad de escribir demasiado código. Sin embargo, este *framework* también nos ofrece la posibilidad de acceder a características avanzadas con las que podremos interactuar mediante JavaScript.

jQuery Mobile incorpora el objeto **mobile** y para poder utilizarlo, lo invocamos con `$.mobile` o `jQuery.mobile`.

El *framework*, también mediante JavaScript, nos habilita la posibilidad de acceder a la siguiente lista de *widgets*: `page`, `dialog`, `button`, `navbar`, `listview`, `collapsible`, `fieldcontain`, `checkboxradio`, `slider`, `textinput`, `selectmenu` y `controlgroup`.

La nómina completa se encuentra disponible en la dirección Web: http://api.jquerymobile.com/category/widgets/.

Si han leído las páginas anteriores, varios de estos nombres ya nos serán familiares, porque los hemos utilizado en el proyecto, y ahora vemos que también se encuentran disponibles para ser utilizados desde la API de jQuery Mobile.

¿Para qué nos puede resultar útil esto? Existen varias situaciones donde podremos darle utilidad a estas características, entre ellas, para definir valores por defecto a un tipo de elemento, modificar su apariencia, agregar alguna función o trabajar sobre alguna de las acciones que permiten realizar.

Métodos de jQuery Mobile

Al interactuar con la API de jQuery Mobile disponemos de una serie de métodos que nos posibilitan trabajar con el objeto **$.mobile**. A continuación, podremos ver los principales métodos de jQuery Mobile en una tabla.

Método	Descripción
changePage()	Permite programar el cambio de página. Puede recibir el destino y otras opciones.
loadPage()	Permite cargar una página externa e incorporarla en el DOM. Puede recibir la URL como *string* o como objeto y otras opciones como objeto.
navigate()	Se puede utilizar para manejar el historial. Recibe como *string* la URL y el objeto *data.*
path.get()	Se emplea para determinar el directorio, de una URL dada.
path.isAbsoluteUrl()	Recibe una URL permite saber si el *path* es una ruta absoluta. Devuelve *true* si la URL es absoluta.
path.isRelativeUrl()	Recibe una URL y se utiliza para saber si el *path* es una dirección relativa. Devuelve *true* si la URL es relativa.
path.makeUrlAbsolute()	Permite convertir una URL relativa en absoluta. Recibe como parámetro la URL relativa y la absoluta.
silentScroll()	Permite realizar un *scroll* silencioso a una posición del eje Y (sin que se haya disparado el evento de *scroll* en el navegador).

Fig. 5.22. Métodos disponibles en jQuery Mobile.

Eventos de jQuery Mobile

jQuery Mobile pone a nuestra disposición una serie de eventos que nos facilitarán el trabajo y nos ahorrarán varias horas de desarrollo. En la tabla siguiente, veremos los principales eventos con los que podremos trabajar al usar este *framework*:

Evento	Descripción
pageinit	Se dispara cuando la inicialización de la página ocurre.
pageload	Ocurre después de que la página está cargada en el DOM.
pageremove	Se dispara en el momento anterior a que jQuery Mobile intenta remover una página del DOM.
pageshow	Se activa cuando al cambiar la página, la nueva página se muestra (después de que la transición ocurre).
hashchange	Nos permite trabajar con el historial y el modo que jQuery Mobile maneja el cambio, mediante *hash* (#). Se dispara cuando la ventana carga una nueva página del *framework*.
pagebeforechange	Se dispara antes de cargar una página o que se ejecute la transición en el cambio de página. También se dispara cuando carga la página (antes de ser agregada en el historial).
pagebeforecreate	Ocurre cuando la página se está inicializando, antes de que cualquier *plugin* se ejecute.
pagebeforehide	Se lanza antes de que una página se oculte.
pagebeforeload	Se dispara antes de que un requerimiento de carga se ejecute.
pagebeforeshow	Ocurre antes de que la página que fue invocada se muestre.
pagechange	Se produce cuando la transición ha pasado y la nueva página se ha cargado.
pagechangefailed	Ocurre cuando el cambio de página falla.
pagecreate	Es en el momento en que la página ha sido creada en el DOM.
pagehide	Se produce cuando se oculta la página, luego de que ha pasado la transición.

scrollstart	Ocurre cuando la acción de *scroll* comienza.
scrollstop	Se produce cuando el *scroll* termina.
orientationchange()	Si el dispositivo lo soporta, se dispara cuando el dispositivo cambia su disposición. En caso de no ser soportado debemos recordar adjuntar también el evento resize().
swipe	Es un evento típico de móviles que ocurre cuando el usuario arrastra el dedo de manera horizontal una distancia de 30 px en un segundo.
swipeleft	Ocurre cuando realiza un *swipe* hacia la izquierda.
swiperight	Ocurre cuando realiza un *swipe* hacia la derecha.
tap	Es otro evento típico de móviles y ocurre luego de que el usuario hace un toque rápido sobre la pantalla.
taphold	Se dispara cuando se realiza un toque y el usuario lo mantiene.
vclick	jQuery Mobile introduce eventos virtualizados del *mouse*. Este evento permite manejar un clic virtual.
vmousecancel	Ocurre cuando un clic es cancelado.
vmousedown	Se dispara cuando el botón virtual del *mouse* comienza a bajar.
vmousemove	Simula el evento del movimiento del *mouse* (*mouse move*).
vmouseover	Permite simular el evento de *mouse over* (el mouse encima de un elemento).
vmouseup	Es la situación que simula cuando el usuario levanta el dedo del botón del *mouse*.

Fig. 5.23. Eventos disponibles en jQuery Mobile.

Encontraremos más información acerca de la API de jQuery Mobile en la siguiente dirección web: http://api.jquerymobile.com/.

Incorporación de plugins

Tanto jQuery como jQuery Mobile basan parte de su potencia en la posibilidad de agregar funcionalidades mediante la integración con otras librerías o *plugins* que extienden sus características para ofrecerles a los desarrolladores más opciones.

En nuestro análisis, deberemos evaluar que estos agregados no afecten de manera significativa la *performance* de nuestra aplicación y que nos agreguen más beneficios que dificultades.

Si ingresamos en http://jquerymobile.com/resources/#Plugins encontraremos una lista de *plugins* de terceros, disponibles para crear efectos o lograr características de una manera simple. Entre los plugins que se pueden encontrar, se destacan opciones para paginación, diálogos, menús *slideshows* y manejar fotografías, entre otros.

Para encontrar un listado de *plugins* compatibles con jQuery, también es posible recurrir a The jQuery Plugin Registry (http://plugins.jquery.com/). En esa dirección Web se reúnen alternativas no solamente para móvil, sino también opciones útiles para desarrollos enfocados en plataformas *desktop*.

Paginación

Para nuestro proyecto necesitamos incorporar una opción que nos permite paginar los resultados en las secciones de **Notas**, **Eventos** y **Noticias**. Una alternativa configurable, que combina JavaScript y CSS, es simplePagination.js. Las opciones de descarga, configuración y ejemplos se encuentran disponibles en el sitio http://flaviusmatis.github.io/simplePagination.js/. La inclusión de este *plugin* es muy sencilla y está explicada en tres pasos esenciales: incluir el archivo JavaScript de la librería (**jquery.simplePagination.js**), incorporar la hoja de estilos (**simplePagination.css**) y luego la inicialización y configuración básica del *plugin* mediante un sencillo *script*:

```
$(function() {
    $(selector).pagination({
   //código de inicialización y configuración
    });
});
```

En lugar del `selector` indicamos el nombre del identificador del elemento que deseamos paginar. En la sección de código, utilizamos las opciones ofrecidas en los ejemplos que brinda el sitio Web de Simple Pagination, en la sección titulada **Available methods**.

Scroll

Tanto jQuery como jQuery Mobile cuentan con métodos nativos para trabajar algunas necesidades de *scroll*, pero algunas necesidades nos llevan a evaluar la

opción de incorporar un *plugin*. En especial si estamos trabajando con un *header* y un *footer* marcados como *fixed*, es importante pensar en un opción que nos permita manejar el *scroll* del contenido que queda en el centro de la aplicación, si es que requiere que el usuario pueda desplazarse en el eje Y.

La librería iScroll nos ofrece una solución para este problema con características compatibles con iOS y Android, entre otros dispositivos. Se puede obtener ingresando en: http://cubiq.org/. En el sitio, se pueden ver ejemplos de código y métodos de funcionamiento para integrarlos con el proyecto.

Otra librería interesante, si se necesita realizar efectos de *scroll* hacia un destino (por ejemplo a coordenadas específicas o un elemento del DOM), es jQuery.ScrollTo de Ariel Flesler. Encontraremos ejemplos de uso y las opciones para obtener la librería en la dirección Web: http://flesler.com/jquery/scrollTo/.

Publicación en el servidor Web y empaquetado de la aplicación

6

Pasos para completar el desarrollo

En el capítulo anterior, profundizamos acerca de los detalles necesarios que nos permiten llevar nuestro proyecto al código. Logramos una aplicación funcional que, ahora, está preparada para pasar a la etapa de *testing*. El momento de realizar los procesos *debugging* y *profiling* resulta fundamental para lograr que nuestra aplicación ofrezca un rendimiento adecuado y una mejor experiencia al utilizarla. En la parte final del capítulo 4, hemos podido analizar un conjunto de herramientas que brindan la posibilidad de realizar diversos test y evaluaciones de *performance* de nuestra aplicación.

A lo largo de este capítulo, trabajaremos con simuladores, emuladores y dispositivos para probar nuestro proyecto.

Una vez que la aplicación está probada y cumple con todas las condiciones esperadas, los caminos que se abren en esta instancia son: publicar la aplicación en un servidor Web (de manera pública o privada) o el empaquetado del proyecto para su posterior distribución.

En este capítulo, analizaremos cómo recorrer esos caminos. En primera instancia, veremos los aspectos a tener en cuenta para publicarla en un servidor Web. Luego, analizaremos cómo facilitar al usuario la creación de un acceso en la pantalla de inicio del dispositivo. Finalmente, nos centraremos en los procesos que nos permitirán lograr el empaquetado para cada plataforma móvil.

Publicación de la aplicación en un servidor Web

Para publicar nuestra Web App en Internet debemos disponer de un *hosting* que nos ofrezca un espacio para almacenar los archivos que componen el proyecto final.

Si no contamos con un plan de *hosting*, podremos encontrar en el mercado una buena variedad de alternativas. ¿Qué debemos tener en cuenta para esta elección? Las características de la aplicación que hemos desarrollado no tiene grandes requisitos por parte del servidor; sin embargo, si pensamos dotarla de otras características más adelante, sería una buena idea tener en cuenta los siguientes ítems:

- Espacio de almacenamiento en el servidor: el tamaño de nuestra aplicación, teniendo en cuenta todos los archivos, es menor a 1 MB. Esto quiere decir que cualquier plan de *hosting* actual podrá almacenarlo sin problema. Si posteriormente decidiéramos incorporar mayor cantidad de recursos multimedia, como por ejemplo, imágenes, audios y videos (especialmente éstos últimos), deberíamos considerar el peso que esto agregaría en nuestro proyecto.
- Ancho de banda (transferencia mensual): el límite de transferencia de datos se mide en GB por mes, pero también encontraremos opciones que ofrecen esta característica de manera ilimitada.
- Tecnologías de servidor: en la aplicación que hemos desarrollado no empleamos lenguajes o tecnologías que requieran características especiales de servidor. Si en el futuro optamos por trabajar con contenidos dinámicos desde el servidor (por ejemplo, PHP), deberíamos asegurarnos que el servidor lo soporte.
- Base de datos: al igual que el ítem anterior, para el proyecto no empleamos datos almacenados en el lado servidor. Si planeamos trabajar con una base en el servidor para manejar noticias o eventos desde ese lado, deberemos asegurarnos que el *hosting* ofrezca el servicio. También debemos verificar qué tipo de base de datos tendremos disponible y cuál es su capacidad.
- Otras características: soporte técnico, *uptime* y *backups* son otros detalles a verificar cuando se elige un plan.

Una vez que tenemos definido todo lo referente al servidor, y contratamos el plan adecuado para nuestras necesidades, estaremos en condiciones de publicar los archivos del proyecto. A continuación, veremos cómo subir los archivos.

Publicación mediante FTP

Los archivos que componen el proyecto pueden subirse al servidor mediante FTP o por medio de algunas herramientas basadas en Web que vienen incluidas en algunos planes de *hosting*. En nuestro caso, optaremos por la primera alternativa,

ya que nos brinda mayor versatilidad, y nos permite elegir la aplicación que utilizaremos para la publicación de los archivos en el servidor.

Recordemos que FTP es un protocolo de transferencia de archivos cliente-servidor y que sus siglas se corresponden con *File Tranfer Protocol*. La transferencia FTP generalmente se establece por defecto en el puerto 21, y permite que la aplicación conozca el estado de los archivos que se están pasando.

Una herramienta muy útil para realizar transferencias mediante FTP es FileZilla, que se ofrece como una opción *Open Source* (*GNU General Public License*) y multiplataforma (disponible en Windows, Mac OSX y Linux, entre otros). Su lanzamiento inicial data del 2001, y cuenta con un muy buen ciclo de actualizaciones con mejoras permanentes y solución de *bug*s. Entre las ventajas destacamos:

- Permite administrar varios sitios.
- Brinda la posibilidad de navegar, crear y eliminar carpetas (locales y del servidor).
- Ofrece un panel con registro de mensajes de lo que está ocurriendo al conectarnos a un servidor y/o transferir archivos.
- Emplea una cola de transferencia para ver el avance de la publicación o descarga de archivos. Permite asignar prioridad y programar acciones de transferencia.

FileZilla se encuentra disponible en: https://filezilla-project.org/. Las traducciones a diferentes idiomas se descargan desde: https://filezilla-project.org/translations.php.

Utilizando la versión en español, desde el menú **Archivo** encontraremos el **Gestor de sitios**. Dentro de esta ventana, estableceremos todos los datos del acceso FTP al servidor: dirección del servidor, puerto, protocolo, cifrado, modo de acceso, usuario y contraseña. Los datos para acceder al servidor son provistos por el proveedor de *hosting*. Algunos planes permiten configurar y crear usuarios FTP para administrar sitios.

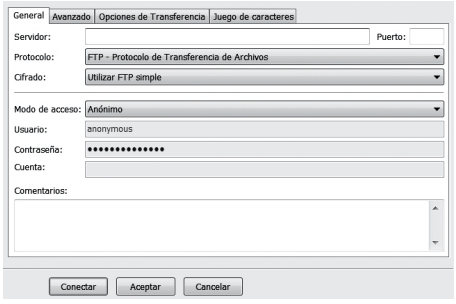

Fig. 6.1. En la solapa General de la ventana Gestor de sitios, podremos configurar las opciones básicas para crear el acceso a la conexión del servidor mediante FTP.

Filezilla cuenta con dos paneles para administrar los archivo, uno de los cuales mostrará nuestras unidades locales; y el otro, una vez que nos conectamos a un servidor remoto, mostrará la ruta donde podremos subir, descargar o borrar archivos y/o crear carpetas, entre otras opciones disponibles.

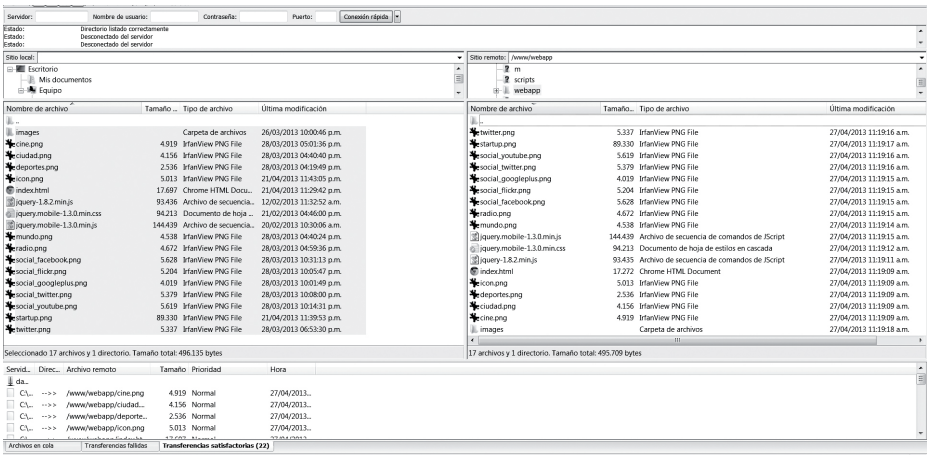

Fig. 6.2. Publicación de archivos en un servidor Web mediante FileZilla.

Si la aplicación que estamos desarrollando estará disponible en un sitio Web donde ya existe contenido, puede ser una buena alternativa crear un subdominio. Para proceder, primero debemos crear un directorio en el servidor. Esto lo podremos realizar desde el programa que utilizamos para acceder mediante FTP o por medio de las herramientas que nos provea el proveedor de *hosting*.

En muchos casos, encontraremos estas opciones disponibles desde las herramientas de administración del sitio, que suele estar disponible en el plan. Estas herramientas pueden estar basadas en Web, y nos ofrecen una interfaz sencilla para trabajar. Dentro de las opciones de administración, también se encuentra la de asociar el subdominio con la carpeta elegida para subir el contenido.

Uno de los paneles de control más utilizado en la Web es cPanel (http://cpanel.net/), cuya primera versión fue publicada en 1996, y actualmente es utilizada por numerosos proveedores de alojamiento Web; además, brinda una alternativa multiplataforma, ya que está disponible para Linux y Windows, entre otros sistemas.

Vale la pena aclarar que si nuestro proveedor cuenta con otro sistema de administración, deberemos verificar cómo configurar estas opciones o consultar al servicio técnico para que nos asesore sobre el tema.

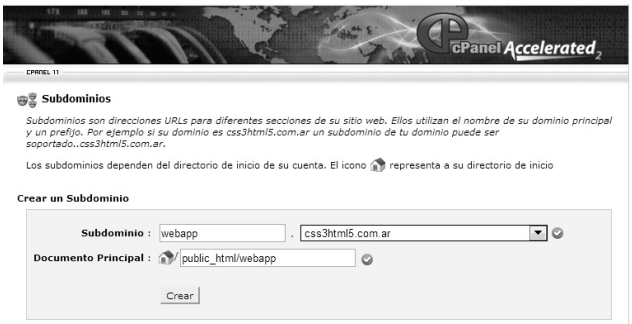

Fig. 6.3. Creación de un subdominio desde cPanel (panel administrador de sitios Web).

Agregar acceso rápido en la pantalla de inicio

Una alternativa para permitir que el usuario pueda tener una Web App en la pantalla de inicio de su dispositivo, es que agregue un acceso. Este es un proceso que el usuario debe realizar desde el navegador de su móvil. Como desarrolladores, podemos facilitarle el camino brindándole instrucciones de cómo hacerlo e incluyendo las etiquetas y contenidos necesarios para personalizar los íconos y la pantalla de inicio (*splash*) de la aplicación.

Un aspecto general que es importante no olvidar al crear la cabecera HTML de nuestra aplicación es especificar un ícono:

```
<link rel="icon" href="icono.png" />
```

Este ícono, además de mostrarse en la pestaña de los navegadores y ser visible al agregar un sitio a favoritos, también puede resultar la opción por defecto que tomen los navegadores móviles para agregar un acceso a la pantalla de inicio, si no tienen asignada otra alternativa más específica.

Más adelante, veremos que en algunas plataformas es recomendable escribir algunas líneas adicionales para personalizar las opciones según el dispositivo, y también deberemos crear íconos de diferentes tamaños.

Agregar acceso rápido en la pantalla de inicio de Android

Los sistemas basados en Android ofrecen la opción de incluir una página Web en la pantalla de inicio desde el navegador Android Browser o mediante Google Chrome en las nuevas versiones del sistema.

En primer lugar, el usuario debe tener la URL de la Web App guardada en favoritos. Una vez allí, debe pulsar sobre el acceso y mantener hasta que aparezca una ventana que ofrece una serie de opciones. Para generar el acceso directo en la

pantalla principal, debe elegir **Add shortcut to home** (**Añadir al escritorio** o **Añadir acceso directo al escritorio**, en algunas versiones en español).

Fig. 6.4. Pantalla de opciones de Bookmark de Android Browser donde se puede ver el ítem Añadir acceso directo al escritorio, para agregar un favorito a la pantalla de inicio del sistema (en este caso Android 4.0.3).

Para personalizar el ícono que se creará como acceso en los dispositivos que cuentan con sistemas Android 1.x hasta Android 2.1, debemos especificar la siguiente línea de código, que deberá estar incluida en la cabecera con el *link* al respectivo ícono:

```
<link rel="apple-touch-icon-precomposed"
href="iconoandroid.png" sizes="1x1">
```

Si bien es una línea dedicada a dispositivos iOS (hasta la versión 4 del sistema móvil de Apple), también es funcional para versiones antiguas de Android.

Las nuevas versiones de Android pueden emplear, por defecto, el ícono establecido para la página. Los tamaños de íconos recomendables para Android arrancan desde 36 x 36 px (dispositivos con pantalla LDPI, de baja resolución) y llegan hasta 96 x 96 px (dispositivos XHDPI, de ultra alta resolución).

Por el momento no es posible especificar los diferentes tamaños de íconos para los diversos dispositivos Android; por tal motivo, se recomienda utilizar el mayor (XHDPI) y el dispositivo lo adaptará automáticamente si es necesario.

Encontraremos más información sobre los íconos para Android en el siguiente enlace:

http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html.

Agregar acceso rápido en la pantalla de inicio en iOS

En los dispositivos Apple que utilizan iOS como sistema operativo, es posible agregar un acceso rápido a la pantalla de inicio (*home screen*) desde Safari. En todos los dispositivos, los pasos que debe seguir el usuario son similares; sin embargo, como desarrolladores debemos asegurarnos de incluir los íconos y las líneas de código para personalizar lo que mostrará cada dispositivo, ya sean iPhone, iPod o iPad en sus versión con pantalla retina o sin ella. Ya que en cada una de ellas podremos encontrar diferencias en el tamaño del ícono que debemos proveer.

Para crear el acceso en la pantalla de inicio, ante todo, el usuario deberá navegar hasta la página que desea agregar, utilizando Safari. Aquí vale la pena aclarar que este procedimiento no se puede realizar mediante otro navegador en iOS.

Una vez que el usuario se encuentra posicionado en la página deseada, deberá pulsar el botón de compartir (**share button**), ubicado en la parte inferior de la pantalla del navegador. Allí se abrirá una nueva ventana donde el usuario deberá elegir **Agregar a la pantalla de inicio** (**Add to home screen**).

Fig. 6.5. El usuario podrá personalizar el nombre del acceso que se creará. Debe pulsar el botón Añadir (Add), ubicado en la parte superior derecha de la pantalla para confirmar la operación; y acto seguido, se agregará el ícono en la pantalla de inicio del dispositivo.

¿Qué debemos hacer como desarrolladores para permitir que el usuario pueda realizar este proceso? En principio, el usuario podrá agregar cualquier página sin necesidad de que se incluyan líneas de código particulares para este fin.

Sin embargo, para ofrecer una mejor experiencia, contamos con una serie de opciones que nos permitirán personalizar algunos detalles para emular una Web App.

Para lograr que la aplicación se vea en pantalla completa, manteniendo la barra de estado superior de iOS, debemos incluir la siguiente etiqueta en nuestro HTML:

```
<meta name="apple-mobile-web-app-capable"
content="yes">
```

Para personalizar la barra de estado, escribimos la siguiente metaetiqueta:

```
<meta name="apple-mobile-web-app-status-bar-style"
content="black" />
```

Los valores que podremos aplicar al atributo `content` en el código anterior son: `default`, `black` o `black-translucent`.

Para establecer los íconos que se agregarán en la pantalla de inicio, deberemos establecer algunas variantes para iPhone o iPod, y también para iPad, tanto para aquellos que cuentan con pantalla retina como para los que no. El código será el siguiente:

```
<link rel="apple-touch-icon"
href=" icono-iphone.png" />
<link rel="apple-touch-icon" sizes="72x72"
href=" icono-ipad.png" />
<link rel="apple-touch-icon" sizes="114x114" href="icono-iphone-
retina.png" />
<link rel="apple-touch-icon" sizes="144x144" href="icono-ipad-
retina.png" />
```

En el código anterior el atributo `size` nos marca cuál es el tamaño para los íconos, luego deberemos especificar la ruta correcta para cada imagen en el atributo `href`, en caso de ser necesario.
Si deseamos establecer una imagen de inicio para la aplicación (*startup image*), necesitaremos indicar la siguiente línea:

```
<link rel="apple-touch-startup-image" href="startup.png">
```

El problema del código anterior es que establece una imagen general para todos los dispositivos; pero, como sabemos, existen varias opciones si tenemos en cuenta iPod Touch/iPhone, iPad y la opción de pantalla retina. La solución es establecer distintas etiquetas y recurrir también al uso de *media queries* para diferenciar los dispositivos. El código que necesitamos en este caso es el siguiente:

```
<!-- iPhone -->
<link rel="apple-touch-startup-image"
media="(device-width: 320px)" href="320x460.png">
```

```
<link rel="apple-touch-startup-image"
media="(device-width: 320px) and (-webkit-device-pixel-ratio: 2)"
href="640x920.png">
<!-- iPad -->
<link rel="apple-touch-startup-image"
media="(device-width: 768px) and (orientation: portrait)"
href="768x1004.png">
<link rel="apple-touch-startup-image"
media="(device-width: 768px) and (orientation: landscape)"
href="748x1024.png">
<link rel="apple-touch-startup-image"
media="(device-width: 768px) and (orientation: portrait) and (-
webkit-device-pixel-ratio: 2)" href="1536x2008.png">
<link rel="apple-touch-startup-image"
media="(device-width: 768px) and (orientation: landscape) and (-
webkit-device-pixel-ratio: 2)" href="1496x2048.png">
```

Todos los métodos detallados en este aparado permiten al usuario tener un acceso directo desde la pantalla principal de su dispositivo. Si deseamos que la aplicación pueda funcionar también estando fuera de línea, podemos repasar lo visto en el capítulo 3 de este libro, en el apartado **Offline Web Application**.

Agregar acceso rápido en la pantalla de inicio de BB10

Los dispositivos que cuentan con BB10 como sistema operativo también tienen la posibilidad de agregar un *shortcut* de una página Web en la pantalla de inicio.

El usuario que desee crear el acceso directo, primero, deberá ingresar en el navegador principal del sistema. A continuación, será el momento de navegar hasta la página que desea agregar. Una vez allí, debe pulsar el ícono que se encuentra en la parte inferior derecha de la pantalla, y aparecerá un *slider* que, entre otras opciones ofrece el ítem **Add to Home Screen**.

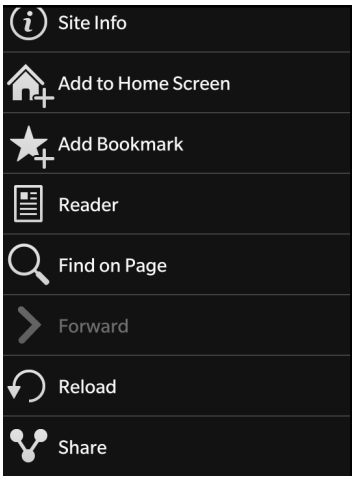

Fig. 6.6. La opción *Add to Home Screen* está disponible en BlackBerry Browser.

Una vez elegida la opción para agregar el ícono a la pantalla de inicio, el usuario puede personalizar el nombre del acceso que está creando. Luego, deberá presionar el botón **Save** para guardar.

Una vez que el usuario acepta crear el acceso, aparecerá un cartel que le indicará que se ha efectuado la operación de manera exitosa. Para verificarlo, se puede cerrar el navegador y comprobar que el ícono está disponible en la pantalla principal de BB10.

Desde el lado desarrollo, podremos agregar un ícono (de un tamaño recomendado de 150 x 150 px). Lo incorporamos en el código de la siguiente forma:

```
<link rel="apple-touch-icon" href="iconobb.png"
sizes="others">
```

Empaquetado de la aplicación

Una alternativa que ha revolucionado el mundo de los móviles es la posibilidad de empaquetar aplicaciones desarrolladas con lenguajes y tecnologías Web a fin de obtener así una aplicación multiplataforma que podrá luego ser distribuida en tiendas o sitios Web.

En la mayoría de los casos, con un mismo código fuente podremos obtener una Web App *Cross-platform* que podrá funcionar con apariencia nativa en *smartphones* y *tablets*. Claro está, que en algunos casos será necesario agregar ciertos códigos y/o *plugins* para lograr que todas las características incluidas funcionen correctamente en todas las plataformas, pero en otros, solamente será cuestión de crear el proyecto, empaquetarlo, probarlo y distribuirlo.

Como hemos mencionado en los primeros capítulos del libro, para nuestro proyecto utilizaremos PhoneGap como herramienta y librería para empaquetar y hacer compatible nuestro proyecto para diferentes sistemas operativos móviles.

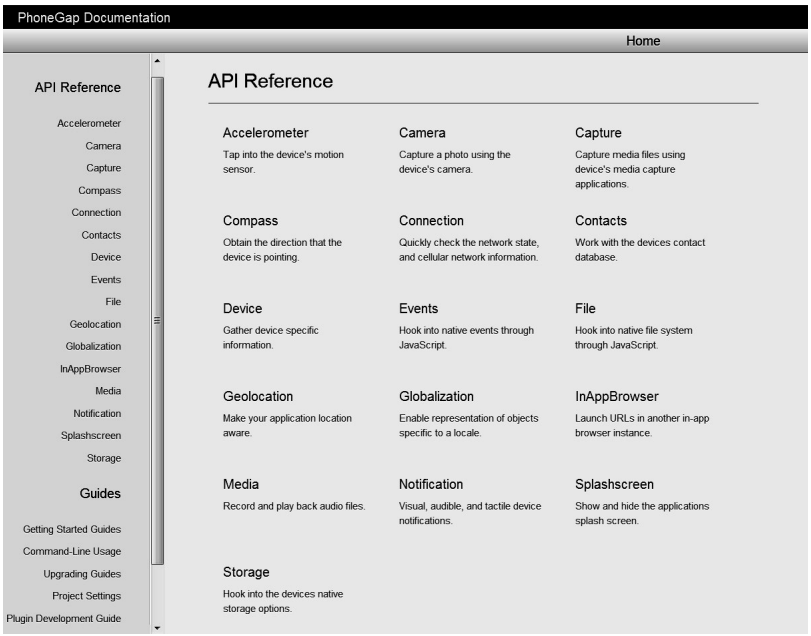

Fig. 6.7. Para el acceso a algunas características del dispositivo (como por ejemplo multimedia, cámara, acelerómetro o contactos) podremos recurrir a las API de PhoneGap (http://docs.phonegap.com/en/). En nuestro proyecto no será necesario utilizar estas API.

En el capítulo 4, dimos los primeros pasos con PhoneGap al descargarlo y configurarlo en los entornos que necesitábamos para proceder al empaquetado.

A continuación, veremos de qué manera debemos proceder para realizar el empaquetado de nuestro proyecto.

Empaquetado para Android

Como hemos podido aprender en el **capítulo 4**, un proyecto de Android puede ser empaquetado mediante PhoneGap utilizando Eclipse. Esta IDE es multiplataforma; por lo cual, podremos realizar los pasos tanto en Windows, Mac OSX como en Linux. A continuación, veremos el ejemplo de cómo hacerlo en Windows.

Primero nos aseguramos que PhoneGap se encuentre instalado en el equipo (http://phonegap.com/install/). Luego tendremos que verificar que Android SDK está instalado y configurado correctamente en nuestro sistema:

(http://developer.android.com/sdk/index.html).

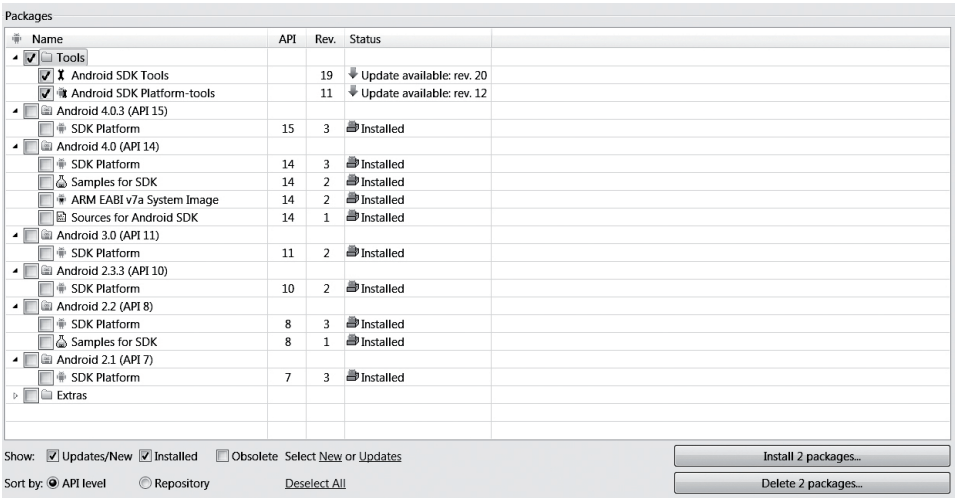

Fig. 6.8. Dentro de las herramientas del SDK de Android, se encuentra Android SDK Manager que permite descargar, instalar y actualizar diferentes versiones de Android parar probar los proyectos en el emulador.

Es posible descargar diferentes API Level para estar en condiciones de utilizar y probar el proyecto en un mayor número de plataformas. Lo que tendremos que tener en cuenta en este caso es el tamaño de los archivos y el tiempo de descarga, ya que, según la velocidad de conexión y la cantidad de paquetes que bajemos, este proceso puede demorar.

Es importante señalar que las versiones de Android cuentan con una numeración para identificarlas, y un número de API Level asociada, pero también existen un nombre que las identifica:

- Android 2.2.x (Froyo): API Level 8.
- Android 2.3 a 2.3.2 (Gingerbread): API Level 9.
- Android 2.3.3 a 2.3.7 (Gingerbread): API Level 10.
- Android 3.0.x (Honeycomb): API Level 11.
- Android 3.1.x (Honeycomb): API Level 12.
- Android 3.2 (Honeycomb): API Level 13.
- Android 4.0 a 4.0.2 (Ice Cream Sandwich): API Level 14.
- Android 4.0.3 y 4.0.4 (Ice Cream Sandwich): API Level 15.
- Android 4.1.x (Jelly Bean): API Level 16.
- Android 4.2 (Jelly Bean): API Level 17.
- Android 4.3 (Jelly Bean): API Level 18.

Es importante destacar que en la actualidad las versiones inferiores a Android 2.2 tienen muy baja cuota de mercado. La línea 2.x de Android está pensada para *smartphones*, pero también se puede encontrar instalada en algunas *tablets*. La línea 3.x (Honeycomb) se desarrolló especialmente para *tablets*. Al momento de escribir esta obra, las versiones identificadas como 4.x estaban expandiéndose en el mercado y siendo adaptadas tanto para *smartphones* como para *tablets*.

Para generar un proyecto de Android será fundamental contar con al menos un dispositivo virtual de Android creado en el **Android Virtual Device Manager**, así podrá actuar como destino cuando probemos nuestro proyecto.

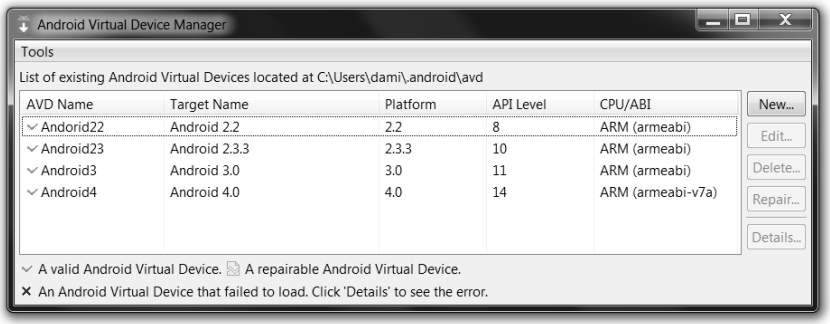

Fig. 6.9. Android Virtual Device Manager es parte del SDK de Android y permite crear nuevos dispositivos virtuales, editar, borrar o reparar los existentes.

Para la creación de un dispositivo virtual, debemos pulsar sobre el botón **New...** y en la pantalla que se abre indicamos el nombre con el que identificaremos el dispositivo virtual, *un target* (el API Level), el espacio que le asignaremos (en MB o GB), el *skin* que tendrá (incluyendo la resolución) y el hardware que utilizará, entre otras opciones.

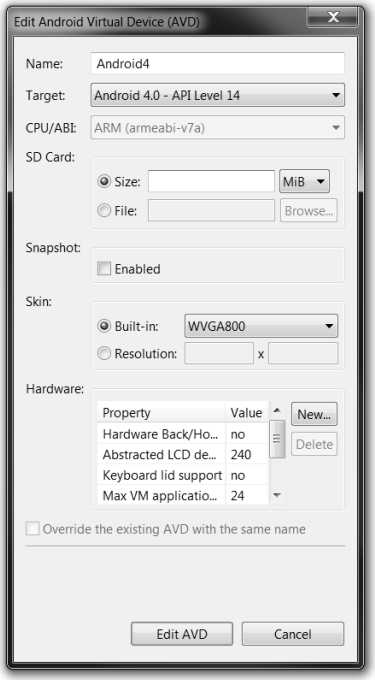

Fig. 6.10. Las características del dispositivo virtual creado podrán ser editadas y modificadas posteriormente de ser necesario.

Cuando estamos realizando la configuración inicial de todas las aplicaciones que necesitaremos para el empaquetado y las pruebas, debemos verificar que estén correctamente configuradas las rutas de las herramientas que necesitaremos para crear y correr el proyecto.

En las variables de entorno del sistema, en la clave `PATH`, deben estar configuradas las rutas completas de los siguientes directorios:

`android-sdk-windows\platform-tools` y

`android-sdk-windows\tools`.

Para acceder a las variables de sistema en Windows 7, hay que dirigirse a **Inicio**, hacer clic con el botón derecho del *mouse* sobre **Equipo**; y en el desplegable, elegir **Propiedades**. En el panel de la izquierda, hay que pulsar sobre **Configuración avanzada del sistema**. En la nueva ventana que sea abre, hay que hacer clic sobre el botón **Variables de entorno...**. Allí, en la sección **Variables de sistema**, hay que desplazarse hasta **Path** y pulsar el botón **Editar...**

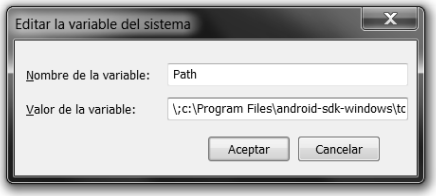

Fig. 6.11. En la ventana Editar la variable del sistema, al agregar una nueva ruta en Valor de la variable, al final de la línea es necesario colocar un punto y coma (;) y luego escribir los *paths* que se deben agregar.

En el equipo también debe estar instalado Java y Apache Ant. En las variables del sistema deben estar configuradas las variables de `%JAVA_HOME%`, `%JAVA_HOME%\`, `%ANT_HOME%` y `%ANT_HOME%\bin`, que además deben estar en el *path*. Esta configuración se puede realizar con el mismo procedimiento que hemos visto en la figura anterior. Para más información sobre esta configuración y todo el proceso, se puede recurrir a la documentación que ofrece PhoneGap las guías de inicio para Android (*Getting Started Guides*) en: http://docs.phonegap.com/en/.

A continuación, debemos crear una carpeta para nuestros proyectos en el disco duro de nuestro equipo. Para este ejemplo, utilizaremos la carpeta `c:\proyectos`.

Una vez que todo está correctamente configurado, debemos descomprimir el zip que descargamos del sitio de PhoneGap en una carpeta. Abrimos la terminal del sistema, y dentro de la carpeta recién creada, nos posicionamos en `lib/android/bin`.

Fig. 6.12. En Windows 7, la terminal se encuentra en el menú Inicio, dentro de la carpeta Accesorios con el nombre Símbolo del sistema. Para acceder a la ruta de un directorio desde esta pantalla hay que utilizar el comando cd y luego especificar la ruta.

Una vez posicionados en el directorio `bin`, deberemos escribir el comando que nos permitirá la creación del proyecto. En las versiones 2.x de PhoneGap el comando es:
`create carpeta_del_proyecto nombre_del_paquete nombre_del_proyecto`.

Para nuestro ejemplo, podríamos escribir:

```
create c:\proyectos\webapp
com.damiandeluca.webapp primerawebapp
```

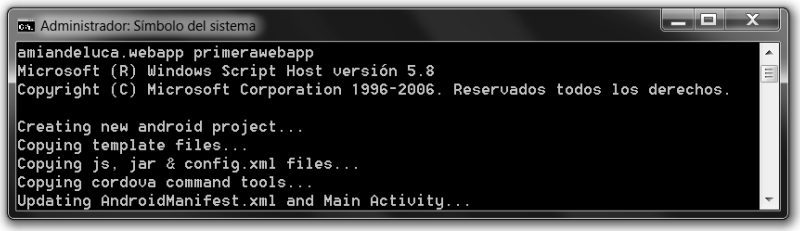

Fig. 6.13. La creación del proyecto demora unos pocos segundos, y en pantalla veremos los mensajes tanto si la creación es exitosa como si hay algún problema.

El proyecto ahora estará creado en la carpeta especificada (en nuestro ejemplo dentro de `c:\proyectos\webapp`).

A partir de PhoneGap 3.x los comandos de creación se pueden correr directamente desde la consola. Para nuestro ejemplo son las siguientes líneas:

```
$ cordova create webapp com.damiandeluca.webapp "primerawebapp"
  $ cd webapp
  $ cordova platform add android
  $ cordova build
```

Dentro de la carpeta `assets\www`, deberemos asegurarnos que se encuentren los archivos de nuestro desarrollo (HTML, CSS, JavaScript, imágenes, etc.). Si utilizaremos alguna de las API de Phonegap/Cordova, deberemos mantener el archivo `cordova-x.x.x.js` (las x identifican la versión) y agregarlo en el documento HTML de nuestro desarrollo de la siguiente forma:

```
<script type="text/javascript" src="cordova-
x.x.x.js"></script>
```

El paso siguiente será ingresar en Eclipse, y en el menú **File**, elegimos **New**. Dentro de las opciones que nos ofrece el programa, escogemos **Project...** y en la ventana siguiente, dentro de la carpeta **Android**, seleccionamos **Android Project,** y luego pulsamos el botón **Next >**. En la ventana siguiente será necesario escribir el nombre del proyecto y especificar la ruta donde se encuentra.

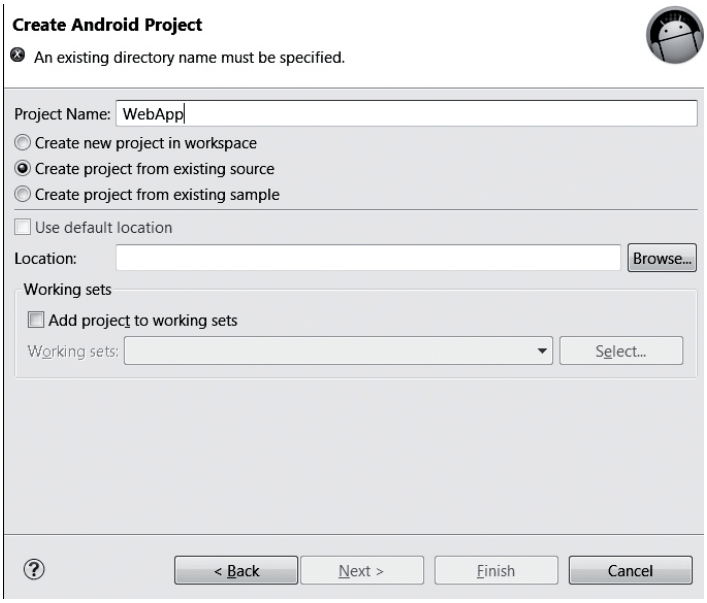

Fig. 6.14. Por defecto, la ventana Create Android Project muestra seleccionada la opción Create Project In workpace. Para indicar la ruta del proyecto se debe elegir Create Project from existing source. Luego, en el campo Location se debe escribir la ruta del proyecto o seleccionarla utilizando el botón Browse.... Una vez especificada la ruta, hay que presionar el botón Next para acceder a la pantalla que permitirá elegir el *target*.

Antes de completar la configuración de la creación del proyecto en Eclipse, será necesario indicar el SDK *target*, es decir, el dispositivo con el cual se probará la aplicación cuando se cree el paquete. Este procedimiento los realizamos desde la pantalla Select Build Target.

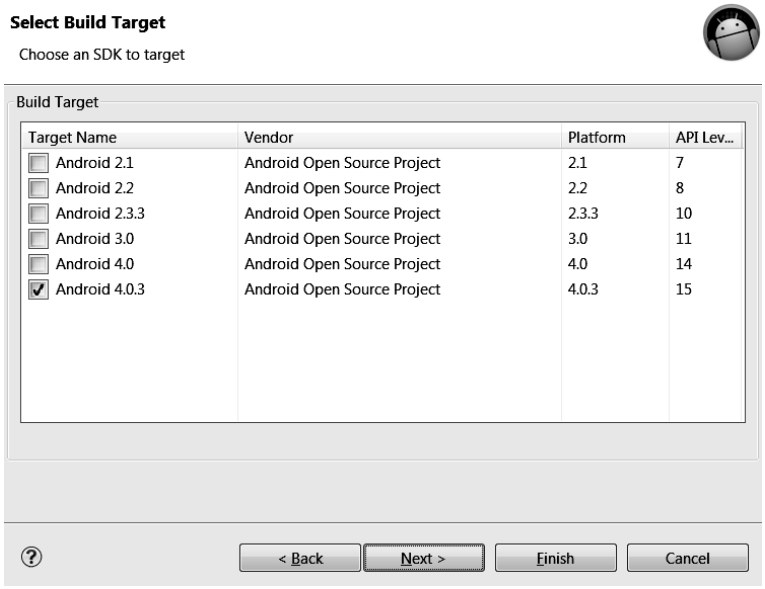

Fig. 6.15. La ventana Select Build Target permite elegir el destino. Luego, se debe presionar el botón Next> para completar la configuración de la aplicación. La siguiente ventana pedirá la confirmación de nombre, paquete y SDK mínimo.

A continuación, el proyecto estará disponible y lo veremos en el panel de la izquierda de la pantalla principal de Eclipse. En la parte inferior de la ventana, está la consola que nos informará si han ocurrido sucesos inesperados en la creación del proyecto o errores.

Para el ejemplo que estamos desarrollando en el libro no es necesario permisos especiales, solo acceso a Internet para las redes sociales y *links* externos.

En la ventana de navegación de carpetas de eclipse, ubicada por defecto en el lado izquierdo de la pantalla, navegamos hasta el proyecto, ubicamos el archivo llamado `AndroidManifest.xml` y hacemos doble clic sobre él. En la parte central de la pantalla, se mostrará una ventana para configurar y definir información sobre este archivo manifiesto. Allí no debemos modificar nada, pero si observamos abajo, veremos varias solapas. Elegimos AndroidManifest.xml.

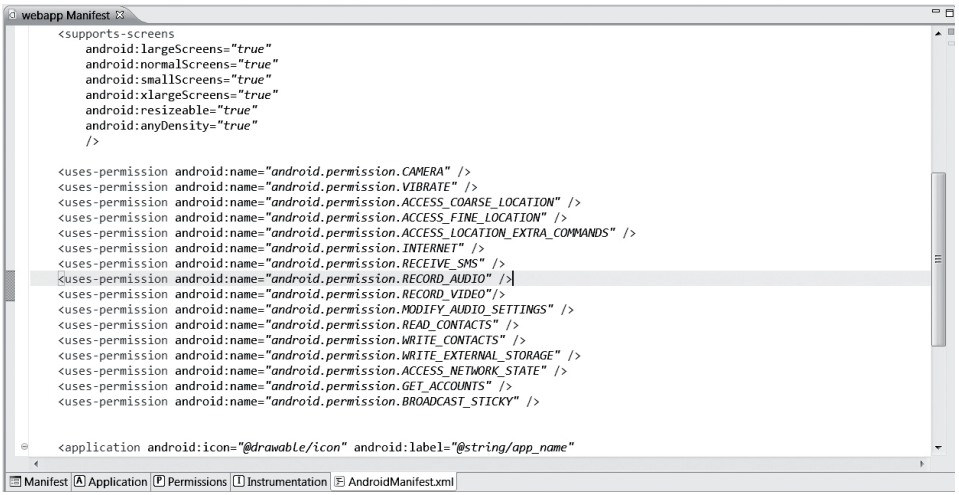

Fig. 6.16. En el archivo AndroidManifest.xml se definen etiquetas con atributos para configurar las características de la aplicación. La etiqueta <uses-permission> cuenta con el atributo android:name para definir los permisos que tendrá la aplicación. El archivo ya posee permisos predefinidos por defecto, debemos borrar aquellos que no sean necesarios para la Web App.

Es importante señalar que los permisos que requiere la aplicación se mostrarán al usuario antes de que realice la instalación. El usuario podrá aceptar o rechazar la instalación en dicho paso. Por esta razón, es importante personalizar y dejar solamente los permisos que utilice la aplicación.

Los permisos que pueden utilizarse en una aplicación están descriptos en el documento *Manifest.permission*, de la guía para desarrolladores de Android y su dirección Web es:

http://developer.android.com/reference/android/Manifest.permission.html

Si deseamos que nuestra aplicación solo funcione en formato *portrait* (o *landscape*), en el archivo `AndroidManifest.xml` podremos recurrir a la etiqueta `<uses-feature>` y establecer en `android:name` el valor (en nuestro caso será `portrait`):

```
<uses-feature android:name=
"android.hardware.screen.portrait" />
```

También, en el archivo `AndroidManifest.xml` es posible especificar la compatibilidad de la aplicación respecto a la versión de API soportada. En la etiqueta `<uses-sdk>` se puede especificar:

Versión mínima: `andorid:minSdkVersion`.

Versión destino: `android:targetSdkVersion`.

Versión máxima: `android:maxSdkVersion`.

El valor asignado a cada uno de estos atributos debe ser el número entero que corresponda al número de versión de API. Encontraremos más información sobre la configuración de `<uses-sdk>` en la documentación de Android para desarrolladores:

http://developer.android.com/guide/topics/manifest/uses-sdk-element.html.

Para probar los proyectos que hemos empaquetado, como ya hemos visto, tenemos disponibles los emuladores de Android. Desde Eclipse podremos ir al menú **Run** y elegir, nuevamente, **Run**. La primera vez, es posible que el programa nos muestre una pantalla titulada **Run As**, con una serie de opciones. Debemos elegir **Android Aplication** y pulsar el botón **Ok**.

Fig. 6.17. En el emulador, es posible navegar la aplicación y probar su funcionamiento.

Para tener en cuenta: la performance en este dispositivo virtual no es comparable con la de un *smartphone* o *tablet*, por lo cual se sugiere realizar pruebas también en dispositivos físicos. En la consola de Eclipse, podremos seguir los sucesos y comprobar si se han producido errores al generar y transmitir el archivo en el emulador.

Para personalizar los íconos de la aplicación, dentro de la carpeta **res** del proyecto, se encuentran los íconos de la aplicación, cuyas medidas son las siguientes: 36 px (ldpi), 48 px (mdpi), 72 px (hdpi) y 96 px (xhdpi).

Si deseamos conectar nuestro *smartphone* o *tablet* con sistema operativo Android en sistemas operativos con Microsoft Windows, será necesario instalar drivers USB. En la siguiente dirección, encontraremos los detalles para obtener e instalar los *drivers* para Windows: http://developer.android.com/tools/extras/oem-usb.html.

Para realizar la depuración en el dispositivo, será necesario verificar que dentro del archivo `AndroidManifest.xml` esté correctamente especificado el atributo `android:debuggable="true"` en el elemento `<application>`.

Fig. 6.18. Los dispositivos Android 4.x, dentro de la pantalla Ajustes cuentan con Opciones de desarrollo. Desde allí, se puede habilitar Depuración USB para configurar el modo de depuración de un dispositivo que está conectado mediante el puerto USB a la computadora. En algunas versiones de Android 4.2.x es necesario buscar esta opción, ya que se no se encuentra visible por defecto (http://developer.android.com/tools/device.html#setting-up).

Encontraremos más información sobre cómo configurar los archivos necesarios del proyecto y los diferentes dispositivos para realizar pruebas en la dirección Web: http://developer.android.com/tools/device.html.
Actualmente, PhoneGap ya no soporta la línea 1.x de Android y los sistemas con Android 2.1 y 3.x han sido declarados *deprecated* (obsoletos) a partir de mayo de 2013.

El detalle de la configuración de los proyectos para todas las plataformas soportadas por PhoneGap se encuentra disponible en la dirección URL: http://docs.phonegap.com/en/edge/guide_platforms_android_index.md.html.

Es importante seguir las novedades que se producen en la documentación de PhoneGap, ya que está plataforma tiene un proceso de actualización continuo y pueden existir cambios entre versiones.

Empaquetado para iOS

Para generar un proyecto compatible con iOS, necesitaremos contar con una computadora que trabaje con sistema Mac OSX, ya que el software necesario para este proceso solo corre en este sistema.

En el capítulo 4, hemos observado que era recomendable contar con Mac OSX 10.7 (o superior), Xcode 4.5 (o superior) y las herramientas de línea de comando de Xcode (*Xcode Command Line Tools*). También hemos visto que era necesario descargar PhoneGap y tener la carpeta `lib/ios` disponible para realizar el proceso.

En este apartado, una vez que tenemos todos los elementos disponibles, profundizaremos en cómo crear y configurar el proyecto en Xcode para luego incluir nuestros archivos en él.

En primer lugar, si trabajamos con PhoneGap 2.x debemos asegurarnos de tener el archivo zip con PhoneGap/Cordova. Es necesario descomprimir el archivo y obtener la mencionada la carpeta `lib/ios`, que luego se utilizará en la terminal.

Ahora, dentro de la carpeta **Documentos** de la computadora, hay que crear otra para PhoneGap (es importante recordar esta ubicación). A continuación, se debe abrir la terminal del sistema, ubicada dentro de **Aplicaciones**, en la carpeta **Utilidades**. Una vez que la terminal se muestra en el *dock* del sistema (en la parte inferior de la pantalla), es necesario arrastrar la ubicación del directorio `bin` (ubicado dentro de `lib/ios`) hacia la terminal. Este paso establecerá la ruta a este directorio y será el momento de escribir lo siguiente en la terminal (y pulsar la tecla **Return** para ejecutar):

```
./create ~/Documents/cordova/webapp
com.miempresa.primerawebapp primerawebapp
```

El detalle anterior corresponde a las versiones 2.x de PhoneGap. Para comprender la orden anterior, el comando `create` realiza la creación del paquete. Recibe como primera especificación la ruta donde estará el proyecto, en el directorio que creamos en la carpeta documentos (en el código del ejemplo anterior `~/Documents/Cordova/webapp`), luego se le pasa el Package Name o nombre del paquete (en el ejemplo `com.miempresa.primerawebapp`); y finalmente, el nombre del proyecto (en nuestro caso, `primerawebapp`).

A partir de la versión 3.x de PhoneGap los comandos de creación para iOS desde la consola de Mac OSX son:

```
$ cordova create webapp com.damiandeluca.webapp "primerawebapp"
$ cd webapp
$ cordova platform add ios
$ cordova prepare
```

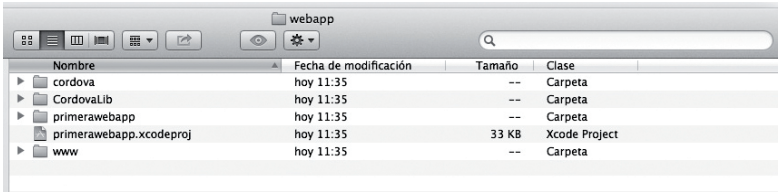

Fig. 6.19. En la carpeta webapp aparecerá creado el proyecto y las carpetas necesarias. Al ejecutar el archivo con extensión .xcodeproj, se abrirá el proyecto en Xcode.

Una vez dentro de Xcode, podremos configurar el proyecto y navegar las carpetas que lo componen.

Fig. 6.20. Pantalla de Xcode donde se pueden visualizar las carpetas del proyecto, el *target* definido para la aplicación y otras opciones acerca de la orientación del dispositivo, íconos e imágenes.

Para probar la aplicación en el simulador, desde la pantalla de Xcode se puede pulsar el botón **Run**, ubicado en la parte superior izquierda de la aplicación. Junto a este botón, se encuentra el botón **Stop** (para detener la ejecución) y un desplegable donde se puede elegir el destino (por defecto, el simulador predefinido).

Mediante el archivo **config.xml**, es posible configurar preferencias, permisos y *plugins* utilizados en la aplicación. El formato de este archivo es XML y sigue la especificación del W3C para *Packaged Web Apps* (*Widgets*), cuya documentación se encuentra disponible en: http://www.w3.org/TR/widgets/.

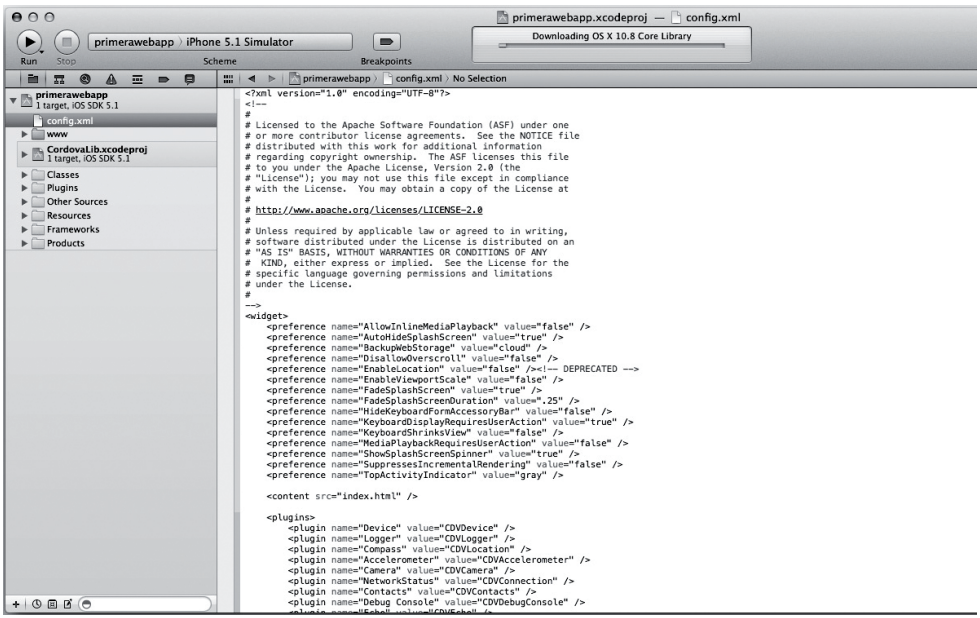

Fig. 6.21. Se puede acceder al archivo config.xml navegando las carpetas del proyecto en XCode. Se encuentra en la raíz de la carpeta del proyecto creado.

A partir de Cordova 2.3, la versión mínima de iOS soportada es la 5. Para Xcode podemos obtener diferentes versiones del SDK de esta plataforma; y de esta manera, probar nuestros desarrollos en los simuladores de iOS. Para esto, debemos estar previamente registrados en iOS Dev Center, cuyo enlace es: https://developer.apple.com/devcenter/ios/index.action.

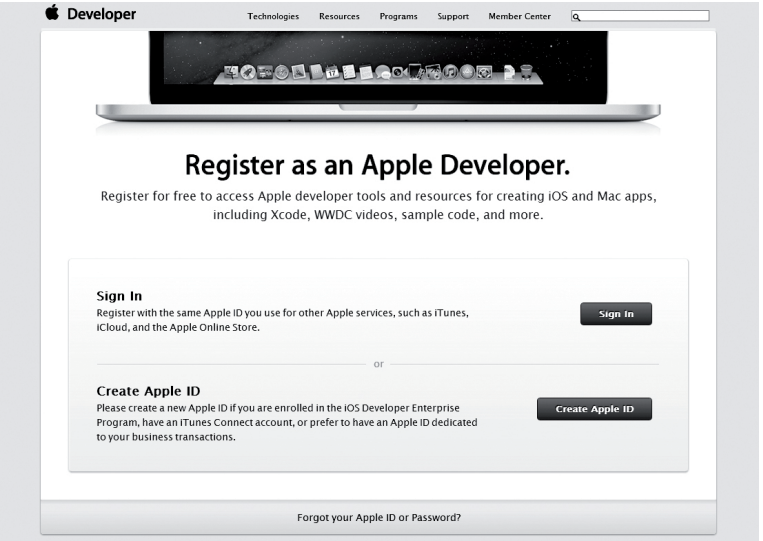

Fig. 6.22. La registración es gratuita para el acceso a las herramientas de desarrollo de Apple.

Para realizar pruebas en dispositivos y, posteriormente, distribuir la aplicación en la tienda de Apple, será necesario registrarnos en iOS Developer Program. Este servicio tiene un costo anual para los desarrolladores, y los datos están disponibles en: https://developer.apple.com/programs/ios/. Aprenderemos más sobre este servicio en el próximo capítulo, al introducirnos en cómo publicar en las tiendas.

Empaquetado para BlackBerry

PhoneGap ofrece compatibilidad para el empaquetado de aplicaciones para dispositivos BlackBerry que cuenten con sistemas BlackBerry OS 5.0 (o superior), BlackBerry PlayBook o BlackBerry 10 (QNX). En nuestro caso, no trabajaremos con PhoneGap para el empaquetado de BlackBerry, sino que lo haremos con WebWorks.

En Windows, para realizar este proceso, necesitamos tener instalado Oracle JDK (http://www.oracle.com/technetwork/java/javase/downloads/index.html#jdk) y en Mac OSX es requisito fundamental tener instalado Java:

(http://support.apple.com/kb/DL1572?viewlocale=en_US).

Adicionalmente, en sistemas Windows, debemos tener Apache Ant (http://ant.apache.org/bindownload.cgi); en Mac OSX, este software viene incluido con la instalación de Java.

Para publicar posteriormente nuestras aplicaciones en BlackBerry World, debemos firmarlas. Todo el procedimiento para obtener y configurar las firmas para las diferentes plataformas se encuentra detallado en el siguiente documento: http://developer.blackberry.com/html5/documentation/set_up_for_signing.html.

Si deseamos probar el proyecto en un simulador, lo encontramos en: http://developer.blackberry.com/devzone/develop/simulator/simulator_installing.html

En nuestro caso, nos centraremos en el desarrollo de la nueva línea de productos BlackBerry (BlackBerry 10 y BlackBerry PlayBook) para los cuales necesitaremos todo lo relacionado con WebWorks:

https://developer.blackberry.com/html5/download/.

Para nuestro proyecto debemos crear un XML (siguiendo el formato para *Widgets* del W3C) y ubicarlo en la carpeta donde se encuentra nuestra aplicación. Para conformar el archivo debemos especificar el encabezado de *Widget*, donde debemos especificar la versión (este número se incrementa en cada versión firmada) y la *id* de la aplicación.

Además, debemos indicar el nombre de la aplicación, la descripción y el autor. Es fundamental especificar en el elemento `<content>` el valor de `src`, indicando el nombre del archivo principal de la aplicación, para que cargue una vez que se inicia. Para nuestra app, el archivo `config.xml` quedaría conformado de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns="http://www.w3.org/ns/widgets"
xmlns:rim="http://www.blackberry.com/ns/widgets"
version="1.0.0.1" id="WebApp">
  <name>WebApp</name>
  <description>Mi primera WebApp</description>
  <author>Damian De Luca</author>
  <content src="index.html"/>
  <icon src="icono.png"/>
  <rim:splash src="splash-1280x768.png"/>
  <feature id="blackberry.app.orientation">
   <param name="mode" value="portrait" />
  </feature>
</widget>
```

En el código anterior, indicamos los archivos que se utilizarán como ícono de la aplicación y también como pantalla *splash* o de carga al inicio. En la sección del elemento `<feature>` indicamos que la aplicación se mostrará en modo

`portrait`. En este archivo de configuración, adicionalmente, se pueden especificar permisos para acceso a hardware u otras opciones del dispositivo.

En la carpeta de la aplicación, debemos incluir un ícono en formato PNG de 114 x 114 px y una pantalla *splash*, también en formato PNG, pero con un tamaño de 1280 x 768 px.

Para más información sobre cómo crear y configurar el archivo `config.xml` para las diferentes versiones de sistemas BlackBerry podemos leer el siguiente documento:
https://developer.blackberry.com/html5/documentation/working_with_config_xml_file_1866970_11.html. El detalle de los elementos que pueden utilizarse en el archivo config.xml se encuentra en la tabla publicada en el documento: https://developer.blackberry.com/html5/documentation/config_doc_elements.html
.

Una vez descargado e instalado BlackBerry 10 WebWorks SDK, debemos dirigirnos a la carpeta especificada para dicha instalación. En esa ubicación, encontraremos el archivo `bbwp.bat`. Este archivo lo debemos correr desde la línea de comando con los parámetros necesarios para realizar el empaquetado de nuestra aplicación. Para acceder desde Windows a la consola de comando nos dirigimos al menú `Inicio` y a la carpeta `Todos los programas/Accesorios`, allí hacemos clic con el *mouse* sobre el ícono `Símbolo del sistema`.

Desde la pantalla de símbolo de sistema, debemos acceder a la carpeta del SDK. Por defecto, el comando que debemos escribir sería (la ruta varía según la especificada en la instalación del SKD de BlackBerry 10):

```
cd c:\Program Files\Research In Motion\BlackBerry 10
WebWorks SDK 1.0.4.11\
```

Ahora nos encontramos en la carpeta del SDK y podremos ejecutar el archivo `bbwp.bat`, indicando también la ruta de nuestro proyecto y la contraseña para firmar. Verificamos que estamos conectados a Internet y escribimos:

```
bbwp c:\proyectos\webapp –o password
```

La ruta indicada debe coincidir con la de nuestro proyecto, y el *password* será el que utilizamos para empaquetar. En la pantalla de consola, veremos el progreso en la generación; y si todo está en orden, nos informará en la ruta donde se guardan los archivos generados. El resultado serán dos archivos con extensión `BAR` que quedarán guardados en la carpeta `device` (para el dispositivo) y en la carpeta `simulator` (para probar en el simulador).

Para cada nueva actualización de la aplicación, debemos modificar la versión en el archivo `config.xml` y volver a realizar la firma de la aplicación para así generar de nuevo los archivos `.BAR`.

Si deseamos probar el proyecto en el simulador (previamente descargado e instalado en el equipo) recurrimos a las herramientas del SDK, ubicadas en la carpeta `dependencies\tools\bin`. Posicionados en la ruta detallada anteriormente, escribimos las siguientes órdenes desde línea de comando (indicando las contraseñas, dirección IP y ruta correspondiente):

```
blackberry-deploy -installApp -password
password_del_simulador -device ip_del_simulador
-package ruta_del_archivo_bar
```

Fig. 6.23. Tanto en el simulador como en el dispositivo, dentro de Systems Settings ubicamos Security and Privacy . Allí se encuentra el acceso a Development Mode. Dentro de esa opción se puede ver Development IP adress y habilitar el modo de desarrollo.

Encontraremos más información sobre la configuración, uso de las firmas, empaquetado y *testing* de aplicaciones para BlackBerry en la documentación publicada en:

http://developer.blackberry.com/html5/documentation/webworks_testing.html.

Empaquetado para otras plataformas

PhoneGap, en sus diferentes versiones, además de ofrecer soporte para empaquetado de aplicaciones para Android, iOS y BlackBerry, también permite trabajar con desarrollos para las siguientes plataformas: Symbian, WebOS, Windows Phone 7, Windows Phone 8, Windows 8, Bada y Tizen.

Es importante destacar que PhoneGap cuenta con actualizaciones permanentes y algunas plataformas o versiones de sistemas pueden dejar de ser soportados, como

el caso de Symbian, que ya no está disponible en las últimas versiones del paquete de PhoneGap. El índice de guías para configurar los entornos, conocer los recursos necesarios y realizar el empaquetado para cada plataforma se encuentra disponible en: http://docs.phonegap.com/en/edge/guide_platforms_index.md.html.

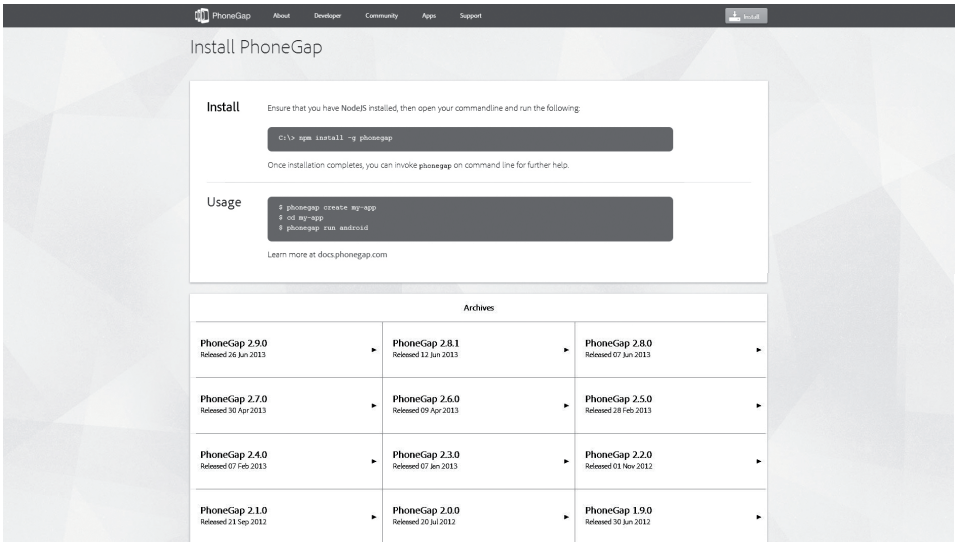

Fig. 6.24. La última versión, versión *release candidate* y versiones anteriores de PhoneGap se encuentran disponibles en: http://phonegap.com/install/.

Los procesos de empaquetado con PhoneGap también pueden realizarse mediante PhoneGap Build. En el capítulo 4, aprendimos cómo acceder, registrarnos y subir un archivo a PhoneGap Build. El detalle sobre cómo configurar el archivo **config.xml** para el empaquetado *online* se encuentra detallado en el siguiente documento: https://build.phonegap.com/docs/config-xml.

Distribución en las tiendas online

7

La era de las tiendas online

En el siglo XXI, las tiendas en línea han renovado el modelo de distribución y negocio en lo que se refiere a contenidos para dispositivos electrónicos. El CD y el DVD son soportes que han quedado obsoletos para un usuario que necesita los contenidos al instante, tanto en lo que se refiere a aplicaciones, libros, música, o videos. Después de muchos años de estar atados al soporte físico, las opciones que residen en la nube registran un éxito que ha llegado para quedarse.

En esencia, el modelo no es nuevo, ya que los primeros intentos de comercializar catálogos de manera electrónica se vieron en los años noventa. Sin embargo, el éxito llegaría en la siguiente década de la mano de Apple. En la cronología, primero aparece iTunes Store, originalmente una tienda *online* para distribuir música. Posteriormente, la ampliación del modelo de negocio de Apple llegaría en julio de 2008, con el App Store, que comienza a ofrecer aplicaciones para iPhone. Esta tienda se potenciaría luego con el resto de los dispositivos móviles de Apple, y hoy ofrece contenidos para iPod Touch, iPhone, y también para iPad. En agosto de 2008, Google anuncia Android Market, tienda que abriría sus puertas *online* en octubre de ese año. Actualmente, su nombre es Google Play, y es el principal canal de distribución de aplicación para dispositivos Android. En el mapa de alternativas para ofrecer contenidos enfocados en móviles y *tablets*, destacan también BlackBerry World y Windows Phone Store, entre otras tiendas en línea que tienen una presencia relevante en la actualidad.

En las próximas páginas del libro, analizaremos el esquema de negocio, y explicaremos detalladamente cómo publicar una aplicación en las tiendas en línea.

Apple App Store

Como comentamos anteriormente, el App Store de Apple se encuentra abierto como tienda de aplicaciones para dispositivos móviles desde el año 2008, y es sin dudas uno de los grandes aciertos comerciales de Steve Jobs. En la actualidad, la tienda de Apple ofrece más de 825.000 apps para los usuarios de iOS.

Fig. 7.1. Los dispositivos iOS cuentan con una aplicación para acceder al App Store que está disponible desde la pantalla principal.

Las opciones para registrarse como desarrolladores, con derecho a publicar en la tienda, son dos. Con un valor de 99 dólares estadounidenses como pago anual, encontramos el **Standard Package** (https://developer.apple.com/programs/ios/), alternativa disponible para personas físicas y empresas. Este plan seguramente será el que podemos optar en nuestro caso para publicar nuestras aplicaciones en el App Store, ya sean apps gratuitas o pagas. En el caso de las aplicaciones pagas, Apple recibe el cobro, y cuando llega el momento de la liquidación nos entrega un porcentaje de la venta

(https://developer.apple.com/programs/ios/distribute.html).

Otra alternativa a la que se puede acceder, con un valor establecido en 299 dólares estadounidenses anuales, es el plan denominado **Enterprise Package** (https://developer.apple.com/programs/ios/enterprise/). Esta opción brinda ventajas para empresas que realizan desarrollo de *In-house Apps* y pueden requerir soporte de parte de los ingenieros de Apple (*Code Level Technical Support*). Para acceder a este plan, además de abonar el importe anual, es necesario ser una compañía u organización con D-U-N-S (identificador para entidades de negocios).

Para conocer las características y una comparación entre los planes que ofrece Apple podemos ingresar en: https://developer.apple.com/programs/start/ios/.

Fig. 7.2. El Standard Package permite registrarse como individuo o como empresa.

Al registrarnos como desarrolladores iOS, el sistema nos permitirá asociar este proceso con una Apple ID existente, por ejemplo, la que utilizamos con los servicios o dispositivos desarrollados por esta empresa. Si deseamos crear una nueva cuenta, el sistema nos solicitará nuestros datos personales para avanzar.

Un aspecto importante a tener en cuenta es que si el período como miembros en el iOS Developer Program concluye y no renovamos la suscripción, nuestras aplicaciones dejarán de estar disponibles en el App Store. Además, claro está, si no realizamos la renovación, no tendremos el derecho a publicar nuevas aplicaciones ni podremos acceder al soporte técnico.

En cuanto a la publicación de una aplicación en el Store de Apple, los pasos se pueden resumir en la creación de los certificados y perfil de distribución, generación del empaquetado de la aplicación y envío de la aplicación para su revisión y posterior publicación en la tienda.

Los certificados permiten comprobar la autenticidad de la aplicación y van al desarrollador del producto. Es importante resaltar que cuando generamos una aplicación, tanto en el caso de etapa de desarrollo como de distribución, debe estar firmada para poder colocarse en los dispositivos de prueba o en el Store de Apple.

Para comenzar con este proceso, ingresamos en https://developer.apple.com/ y nos dirigimos a la sección **iOS Dev Center**. Una vez que nos autenticamos (**Login**) con nuestro Apple Id (vinculada con la cuenta de desarrolladores iOS), obtendremos acceso a descarga del último SDK de iOS, guías, recursos, foros, centro de soporte, iTunes Connect y el acceso al iOS Provisioning Portal, entre otras opciones.

Accedemos a **iOS Provisioning Portal** que funciona como un centro de administración de certificados, autorización de dispositivos, perfiles y opciones de distribución de las aplicaciones que desarrollemos. Dentro de este portal, en el panel de la izquierda, vamos a la sección **Certificates**, y nos posicionamos en la solapa **Development**. Vale la pena aclarar, que los certificados nos permitirán registrar la computadora que utilizaremos para el desarrollo, y en caso de cambiarla o utilizar otra, también necesitaremos certificados.

La primera vez que se realiza este proceso será necesario descargar un archivo con extensión **.cer** (mencionado anteriormente). Haciendo doble clic sobre ese archivo, se abrirá la aplicación Acceso a Llaveros (*Keychain Access*).

Fig. 7.3. En caso de ser necesario ingresar manualmente, la aplicación Acceso a Llaveros se encuentra dentro de Programas en Utilidades.

Dentro de la aplicación Acceso a Llaveros, en el panel de la izquierda, es necesario pulsar **Todos los ítems** y desde el menú superior, elegimos la opción **Acceso a llaveros / Asistente para certificados / Solicitar un certificado de una autoridad de certificación....** La ventana que se abre a continuación solicitará algunos datos, entre ellos, nombre y correo electrónico. En esta ventana, es necesario indicar, en la parte de palabra clave, que se guarde en disco. Este archivo generado es una solicitud para certificado de desarrollo que deberemos subir al Portal de Apple (desde el navegador) en la misma sección donde obtuvimos el archivo **.cer** (**Certificates**, posicionados en solapa **Development**). Allí debemos subir el archivo generado por el Acceso a Llaveros mediante el botón **Request Certificate**, y seleccionando el archivo con el botón ubicado en la parte inferior de la pantalla siguiente. Confirmamos luego con el botón **Submit**. Una vez subido y aprobado, podremos ver la fecha de expiración del certificado. Este certificado lo tendremos disponible por doce meses para las aplicaciones que desarrollemos. Lo podremos descargar con el botón **Download** o bien revocarlo mediante el enlace **Revoke**. El archivo que descargamos en esta etapa, nuevamente será de extensión .**cer**, y lo abrimos, como lo hicimos anteriormente con el programa Acceso a Llaveros. En el panel de la izquierda, se puede ir a la sección **Claves** y, una vez elegida esta opción, se accede en el centro de la pantalla de la aplicación a la clave pública y privada de nuestro perfil de desarrollador. Sobre la clave privada hacemos clic derecho y en el menú emergente elegimos **Exportar....** El proceso nos solicitará que indiquemos una contraseña. Al completar este paso tendremos generado el archivo de extensión

.p12 en la ruta que hayamos especificado para que sea guardado. Recordemos que este certificado es de desarrollo.

El proceso para obtener un certificado de distribución, arranca nuevamente desde el **iOS Provisioning Portal**, en la sección **Certificates** y los pasos son similares a los detallados anteriormente para la creación de un certificado de desarrollo, incluyendo la exportación de las claves desde el Acceso a Llaveros. Es recomendable, al guardar los certificados, colocarles nombres descriptivos que nos permitan diferenciarlos.

Ahora, podemos comenzar a trabajar sobre los datos de la aplicación para crearla en el portal. Ingresamos en **iOS Provisioning Portal**. En el panel de la izquierda, elegimos **App IDs**, que será individual para cada aplicación creada, y luego pulsamos en el botón New App ID. En la descripción del producto, debemos indicar el nombre (alfanumérico) y el identificador de la aplicación que se solicita en el formulario con el nombre **Bundle indentifier** (por ejemplo: `com.miempresa.nombreaplicacion`). Confirmamos la creación con el botón **Submit**.

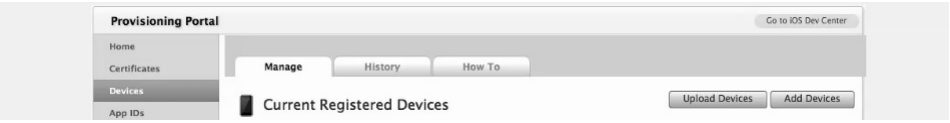

Fig. 7.4. Para agregar dispositivos en los cuales probar una aplicación, dentro de iOS Provisioning Portal, en el panel de la izquierda, elegimos Devices y podremos agregar nuevos con Add Devices. Luego, deberemos indicar el nombre del dispositivo y los 40 caracteres de su UDID. Se pueden registrar hasta 100 dispositivos.

Ahora podremos preparar un perfil nuevo para la aplicación. En la sección **Provisioning** del portal, creamos un nuevo perfil mediante el botón **New Profile**, dentro de la solapa **Development**. A continuación, indicamos el nombre del perfil, el certificado que utilizaremos, el App ID y los dispositivos que autorizaremos para probar (podemos elegir también **Select All**). Confirmamos con el botón **Submit**. A continuación, veremos los detalles cargados, y la opción para descargar el perfil mediante el botón **Download**. Estos archivos de perfil cuentan con la extensión **.mobileprovision**. Si hacemos doble clic sobre el archivo, se abrirá **Organizer** y quedará instalado en el Xcode.

Para crear ahora la distribución de una App, desde la sección **Provisioning**, elegimos la solapa **Distribution**, donde pulsamos el botón **New Profile**. En la pantalla que sigue, es necesario elegir el método de distribución, entre **App Store** y **Ad Hoc** (en nuestro caso, elegimos la opción para distribuir mediante la tienda). Luego establecemos un nombre de perfil y elegimos entre las App ID existentes. Confirmamos con **Submit**, y luego se puede descargar el perfil con el botón **Download**. De la misma forma que hicimos con el certificado de desarrollo, debemos hacer doble clic para instalar este certificado de producción en Xcode.

Ahora desde Xcode accedemos al menú **Product** y elegimos **Archive**. Esta opción genera un archivo compilado con extensión **.ipa**. Para acceder a este archivo, desde el Xcode vamos a **Organizer** (el acceso se encuentra entre los botones que están ubicados en la parte superior derecha de Xcode). Dentro de **Organizer** elegimos **Archives**. Desde allí podremos acceder a las opciones **Validate** (validar), **Share** (compartir) y **Submit** (enviar). Con la opción de compartir podremos utilizar los certificados de desarrollo para distribuir una versión de la aplicación que será probada en los dispositivos iOS que autorizamos. Para enviar la aplicación a la tienda, deberíamos generar el empaquetado con un certificado de distribución.

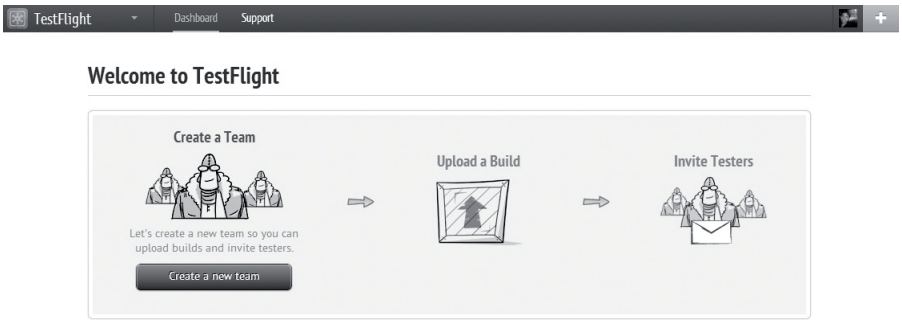

Fig. 7.5. TestFlight (https://testflightapp.com/) es una opción que facilita la administración y publicación de apps en la etapa de *testing* entre los dispositivos que hayamos registrado.

Una vez probada la app, podremos avanzar en el trámite de publicación, para el cual deberemos llenar algunos formularios en línea para aportar más detalles de la aplicación a la tienda. Entre los datos que deberemos indicar se encuentran: nombre de la aplicación, SKU, fecha de publicación (a partir de que esté aprobada o posterior), precio (gratuita o paga, con valores establecidos en una escala que establece Apple), disponibilidad mundial, número de versión, *copyright*, categoría (principal y secundaria, si corresponde), calificación, metadatos (descripción, palabras clave, URL de soporte, etc.), datos del contacto (nombre, apellido, correo electrónico y teléfono), íconos de la aplicación y capturas de pantalla.

Una vez que la aplicación es enviada, el equipo de Apple la pone a prueba a fin de verificar que cumpla con los requisitos técnicos y las directrices establecidas para que una aplicación pueda ser publicada:

(https://developer.apple.com/appstore/guidelines.html)

Este proceso puede demandar varios días y estar sujeto a pedido de modificaciones.

Un tema importante a destacar, antes de publicar una aplicación en cualquiera de las plataformas mencionadas en el libro, es que si la app utilizará recursos externos o necesitará consumir información de manera remota, dichas funcionalidades deberán estar en producción antes de su revisión.

Google Play

Google Play es la tienda que permite a los desarrolladores ofrecer sus aplicaciones y juegos para Android. Es el portal ideal para distribuir nuestra aplicación si pensamos en esta plataforma, por su visibilidad en Internet y, principalmente, por la integración con el dispositivo, ya que el usuario cuenta con la app de la tienda incluida en su teléfono o *tablet*.

Google Play es una de las tiendas de aplicaciones de mayor importancia en la actualidad, ya que cuenta con más de 850.000 apps activas disponibles para su descarga. Para registrarnos como desarrolladores en Google Play, debemos abonar la suma de 25 dólares estadounidenses.

Fig. 7.6. Para registrarnos como desarrolladores en Google Play, podremos ingresar en https://play.google.com/apps/publish/signup/ con nuestra cuenta de Google. Allí tendremos las opciones para verificar en qué países se pueden distribuir aplicaciones así como instrucciones sobre el servicio y las opciones para continuar con el pago.

Un dato importante sobre esta tienda es que, en la actualidad, existe una lista extensa de países cuyos desarrolladores pueden publicar aplicaciones gratuitas, pero es recomendable chequear el listado de países admitidos para que un desarrollador pueda publicar una aplicación paga. La lista de ubicaciones admitidas

para usuarios que pueden vender apps en Google Play se encuentra disponible en la siguiente dirección Web:
https://support.google.com/googleplay/android-developer/answer/150324.
Una vez que estemos registrados en Google Play, podremos acceder a la consola para administrar nuestras aplicaciones. Desde allí podremos visualizar la lista de aplicaciones que tendremos publicadas o en proceso de publicación. Además, tendremos acceso al precio con el que se encuentra publicada la App, las instalaciones activas, el *rating* y la fecha de la última actualización, entre otras opciones.
Recordemos que el formato de las aplicaciones que podremos subir en la tienda de Android es APK, y que puede ser generado desde Eclipse, Android Studio o mediante PhoneGap Build, entre otras posibilidades. Antes de generar el archivo definitivo, debemos editar los archivos del proyecto para quitar todas las llamadas a logs o alertas creadas para realizar verificaciones en la aplicación en la etapa de desarrollo. También es importante remover en el archivo manifiesto del proyecto (**AndroidManifest.xml**) el atributo `android:debuggable`, si estuviera presente. Además, debemos verificar que, en la versión inicial que generemos, los valores de `versionName` y de `versionCode` se encuentren fijados de la siguiente manera (por defecto):

```
android:versionName="1.0" android:versionCode="1"
```

Para el proyecto que estamos siguiendo en el libro, debemos asegurarnos que esté firmada la aplicación que hemos generado con PhoneGap utilizando Eclipse para realizar el empaquetado, ¿Por qué razón? Esta medida de seguridad le brinda mayor seguridad al archivo que estamos distribuyendo, pues permite que solo el desarrollador que firmó la aplicación pueda modificar su contenido. Google Play ha establecido como medida de validación para publicar una aplicación que ésta se encuentra firmada.

Para generar el APK con una firma, utilizando Eclipse, nos dirigimos al menú **File**, y elegimos la opción **Export**. Desde allí, dentro de la carpeta **Android**, se debe elegir **Export Android Application**. Al presionar el botón **Next**, nos solicitará elegir el proyecto, y luego el certificado (**keystore**).

Fig. 7.7. La firma puede crearse en ese momento o puede utilizarse una previamente creada para añadirla a la aplicación que se está empaquetando.

Todos los detalles sobre la firma de una aplicación Android se encuentran disponibles en: http://developer.android.com/tools/publishing/app-signing.html.

Una vez que contamos con el archivo APK firmado y probado su correcto funcionamiento, procederemos a la publicación de la aplicación en la tienda.

En primer lugar, es recomendable configurar la cuenta de Google Wallet Merchant. Para esto, ingresamos con nuestro usuario en:
https://play.google.com/apps/publish/.
Así, accedemos, desde la sección **Financial reports**, a la opción **Setup a Merchant Account now**. Luego de completar este paso, nos podremos dirigir a la consola de Google Play (**Developer Console**). En **All applications** veremos la lista de aplicaciones que hemos publicado, si ya lo hemos hecho anteriormente, y sus detalles.

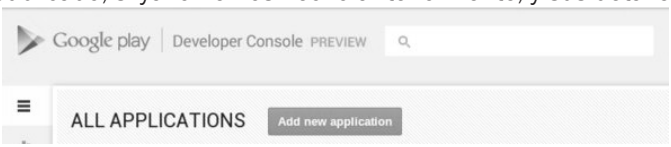

Fig. 7.8. Mediante el botón Add new application podremos comenzar el proceso de publicación de una nueva aplicación en Google Play.

Una pantalla nos solicitará el idioma de la aplicación y el título. Luego, podremos presionar el botón **Upload APK** para subir a la tienda el archivo ubicado en nuestro disco. Si el archivo no está corrupto, luego de unos momentos, una pantalla nos mostrará los datos extraídos del manifiesto: nombre del paquete, código de versión, nombre de versión, dispositivos soportados y dispositivos excluidos. En esta pantalla, también podremos verificar detalles adicionales si así lo deseamos.

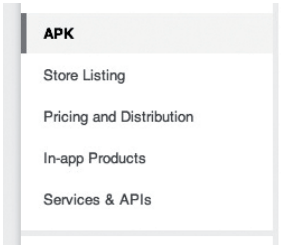

Fig. 7.9. En el panel izquierdo de la pantalla encontraremos los pasos que debemos seguir para la publicación.

En el siguiente paso encontraremos, en primer lugar, **Store listing**, espacio en el cual deberemos establecer detalles del producto, entre los que se encuentran: título (10 a 30 caracteres), descripción (5 a 4000 caracteres) y texto promocional (0 a 80 caracteres). Allí también podríamos especificar traducciones en otros idiomas. En esa misma pantalla, encontraremos la sección **Graphics Assets**, en la cual tendremos que incluir el material gráfico: capturas de pantalla (son soportadas diferentes resoluciones en archivos JPG o PNG sin transparencias), ícono en alta resolución (512 x 512 px en PNG con canal alfa), gráfico de características (1024 x 500 px en formato JPG o PNG sin transparencias), gráfico de promoción (180 x 120 px en formato PNG sin transparencias) y un video promocional (URL de un tráiler de la aplicación, previamente subido en YouTube). En la sección titulada **Categorization** debemos especificar el tipo de aplicación (aplicación o juego), la categoría en la que deseamos incluirla y la calificación del contenido (*rating*). Encontraremos luego **Contact Details** donde deberemos incluir los datos de nuestro *Website* (o el de la aplicación), dirección de correo electrónico y teléfono. Para finalizar esta etapa, se puede especificar un enlace a la política de privacidad (**Privacy Policy**) que se utilizará para el contenido que se está publicando, o bien elegir el *checkbox* que posibilita subir esta información en otro momento. Una vez finalizado el proceso de todas estas secciones, en la parte superior encontraremos el botón **Save** para guardar toda la información que hemos indicado.

Ahora es el momento de especificar los datos relacionados con precio y distribución (**Pricing and distribution**). En esta parte, podremos establecer si la aplicación la pondremos en Google Play como gratuita o paga. En caso de encontrarse esta última opción disponible para nuestro país, tendremos la posibilidad de indicar el valor de venta de la *app*. También podremos indicar en qué países estará disponible (o elegir **Select All Countries**). Si desplegamos las opciones, también estarán disponibles en algunos casos los operadores de telefonía (*carriers*) que estarán habilitados para la descarga. Es posible también establecer valores diferenciados (en valor y moneda) según el país. En cada paso, es importante recordar el presionar **Save**, para guardar las especificaciones que hemos cargado en cada sección.

Una vez completados todos los datos, en la esquina superior derecha, el botón **Draft** pasará a ser **Ready to publish**, y lo podremos pulsar para que nos ofrezca la opción **Publish this app**, y así publicar nuestro producto, que estará disponible luego de que el equipo de Google Play realice la verificación de la aplicación y lo admita en la tienda.

Posteriormente, en el caso de realizar una nueva versión o actualización de la aplicación, antes de generar nuevamente el archivo APK y subirlo a la tienda, deberemos incrementar la versión. Este procedimiento puede realizarse desde Eclipse. Para esto editamos **AndroidManifest.xml**, y buscamos la etiqueta `<manifest>`. Allí será necesario incrementar el valor que tiene asignado `android:versionName` (soporta números decimales, es la versión que puede ver el usuario), y también el valor de `android:versionCode` (versión interna, requiere un número entero).

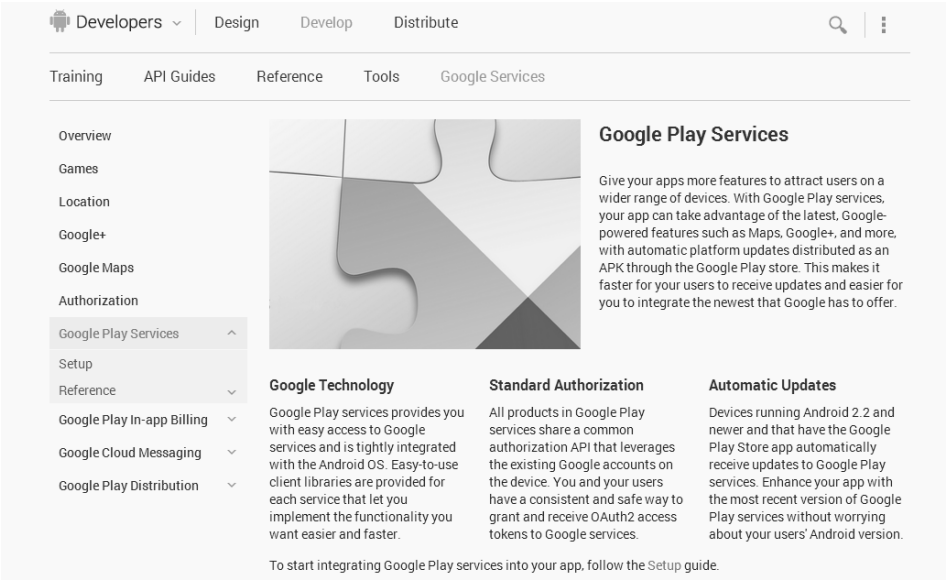

Fig. 7.10. Para saber más sobre la publicación y los servicios de Google Play podemos consultar la documentación disponible en la dirección Web: http://developer.android.com/google/play-services/index.html.

En nuestro ejemplo anterior, habíamos fijado la versión 1, al subir por primera vez la App a la tienda. Para la actualización podríamos especificar el siguiente código dentro de la etiqueta `<manifest>`:

```
android:versionName="1.1" android:versionCode="2"
```

BlackBerry World

Con el lanzamiento de BlackBerry 10, llega también una renovación importante para la tienda, que pasa a llamarse BlackBerry World, y que en la actualidad, permite distribuir aplicaciones, juegos, *themes*, música y videos (películas y episodios de series). El servicio está disponible en diferentes idiomas, entre ellos, español, inglés, alemán, italiano, francés y portugués.

Los contenidos para este *store online* pueden ser publicados tanto en formato gratuito como de pago por desarrolladores y compañías. En mayo de 2013, se anunciaba que la tienda ya contaba con más de 120.000 aplicaciones disponibles para la plataforma BlackBerry 10.

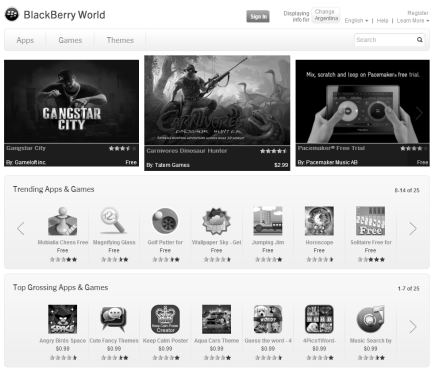

Fig. 7.11. BlackBerry World está preinstalada como aplicación en los nuevos dispositivos BlackBerry 10; también es posible acceder directamente desde la Web para ver las novedades y registrarse desde: http://appworld.blackberry.com/webstore/?&lang=es.

BlackBerry World se encuentra disponible además en la siguiente dirección Web: http://ar.blackberry.com/apps/app-world/download.html. Allí se pueden encontrar las opciones para obtener la aplicación para la computadora y las alternativas para la descarga en un Smartphone. BlackBerry App World 2.1 requiere BlackBerry 4.5 o superior como sistema operativo. Por su parte, BlackBerry App World 3.1 requiere que el usuario cuente con un sistema BlackBerry 5.0 o superior.

Como desarrolladores podremos registrarnos de manera gratuita en la tienda de BlackBerry para publicar nuestras aplicaciones. Para esto, ingresamos en https://appworld.blackberry.com/isvportal/vendor/reg_terms.do y nos registramos como Vendor de BlackBerry. Luego de leer y aceptar las condiciones, el sistema nos solicitará nuestros datos personales. Una vez completados los pasos requeridos, nos llegará un correo electrónico solicitando que enviemos una copia digitalizada de nuestro documento de identidad a fin de dar curso a nuestra solicitud.

Para publicar una aplicación en BlackBerry World, es fundamental que el archivo que generamos, por ejemplo con PhoneGap o BlackBerry WebWorks se encuentre firmado. Las firmas se pueden solicitar mediante el formulario *online* que se encuentra en: https://www.blackberry.com/SignedKeys/codesigning.html. Luego de completado, nos llegarán dos correos con las firmas (este proceso suele demorar solo unas horas). Cada correo electrónico llegará con un archivo de extensión **.CSJ** que deberemos instalar en el equipo que genera los archivos para publicar en la tienda. Dentro de la documentación que brinda BlackBerry para el desarrollo de aplicaciones con HTML5 (https://developer.blackberry.com/html5/documentation/) se encuentra un apartado con las explicación de cómo firmar las aplicaciones para las diferentes plataformas (**Sign you app**).

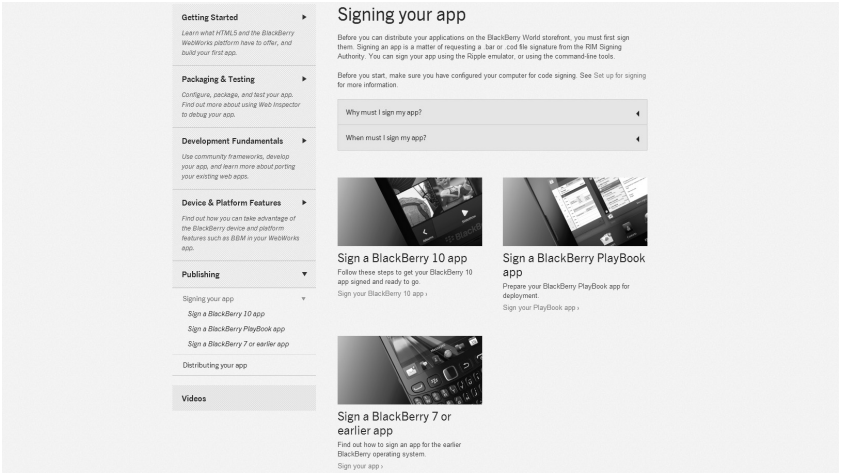

Fig. 7.12. El proceso para registrar las firmas y luego firmar nuestras aplicaciones puede tener diferencias según la plataforma utilizada. Por esta razón, es recomendable leer antes la documentación actualizada que ofrece BlackBerry sobre este tema.

Superados todos los pasos anteriores, y una vez recibida la confirmación de que todo está correcto, es el momento de comenzar el proceso de publicación. Ingresamos en: https://appworld.blackberry.com/isvportal/login_input.do. Allí debemos indicar nuestra BlackBerry ID (correo electrónico y contraseña) para iniciar sesión para acceder a la pantalla principal del Vendor Portal.

Fig. 7.13. Desde la pantalla principal del Vendor Portal es posible acceder a la administración de productos publicados (Manage Products), descarga de reportes (Download Reports) y administración de cuentas de usuario (Manage Accounts Details y Manage User Accounts).

Dentro de las opciones que nos ofrece la pantalla principal, elegimos la opción Manage Productos para comenzar a publicar una nueva aplicación.

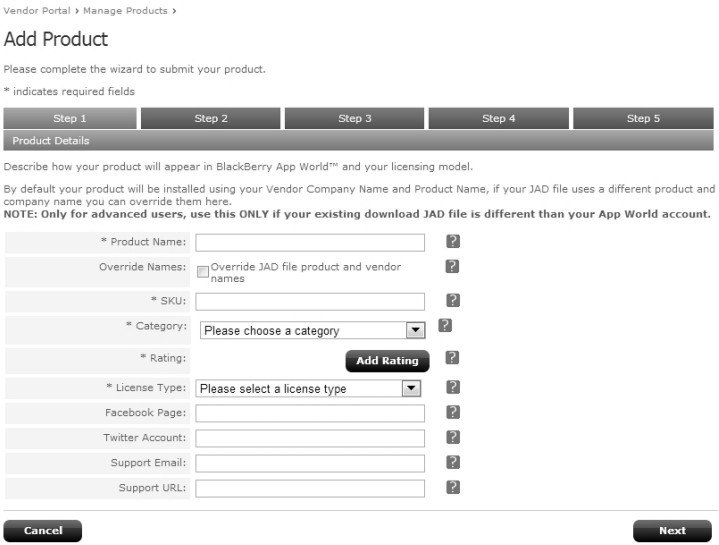

Fig. 7.14. El primer paso pedirá: nombre del producto, nombre corto de la aplicación ("SKU"), categoría, calificación y licencia (como datos obligatorios). También podremos especificar página de Facebook, cuenta de Twitter, correo electrónico y URL de soporte.

Luego de presionar el botón Next en la pantalla anterior, será el momento de escribir una descripción corta (máximo 50 caracteres) y otra larga (máximo 4.000 caracteres) del producto. También deberemos especificar las palabras clave (hasta 10 frases, separadas por coma) relacionadas con la aplicación. Una ventaja

interesante de este apartado es que permite agregar datos en más de un idioma por medio del botón **Add Language**.

En el tercer paso de este proceso, nos solicitarán imágenes. El ícono del producto (**Icon Product**) debe ser provisto en formato PNG con una resolución de 480 x 480 px. La imagen destacada, que será utilizada si el producto llega a la portada, se denomina **Featured Image**, y debe subirse en formato PNG con una resolución de 1920 x 1186 px. También podremos incluir de manera opcional, hasta 50 capturas de pantalla de la aplicación (**Screenshots**). Si bien este último ítem no es obligatorio, es recomendable realizar algunas capturas atractivas y subirlas, ya que esto puede ser un condimento muy importante para que el usuario decida descargar la aplicación en su dispositivo.

El cuarto paso del proceso está relacionado con la posibilidad de restringir compañías (**carriers**) y/o países en la distribución del producto.

Luego, podremos verificar los datos e indicar si deseamos (o no) que la aplicación sea publicada una vez aprobada. Este último detalle puede ser útil si nuestro desarrollo debe estar disponible en una fecha indicada, por ejemplo, para un evento.

Una vez que el producto está creado en la tienda, debemos subir la primera versión o *realese*, que luego podremos ir actualizando. Para esto podremos utilizar el archivo que hemos generado mediante PhoneGap o utilizando BlackBerry WebWorks.

Entre los aspectos que deberemos confirmar antes de publicar la aplicación, se encuentran las indicaciones que establecen si el producto tiene alguna seguridad, control de autenticación o encriptación. También deberemos indicar si la aplicación tiene material de terceros (**third party content**), y si la aplicación genera algún tipo de información o contenido a través del usuario, en cuyo caso deberemos hacernos cargo de que dicho material esté debidamente protegido.

En la pantalla de la versión, podremos agregar el producto como **Add filebundle**, indicar el número de versión y las novedades que ofrece. También podremos especificar el lenguaje y las plataformas que soporta (incluyendo la versión mínima de sistema operativo). Con el botón Files podremos acceder a la ventana para subir el archivo de extensión **.bar** firmado (es condición que esté firmado para ser publicado en la tienda).

Cuando una nueva versión es agregada, puede ser enviada para aprobación mediante el botón **Submit Releases for Approval**. Esta parte del proceso es similar a cuando subimos nuevas versiones y/o modificaciones en la aplicación, donde también deberemos ratificar los datos en el sumario del producto.

La aprobación o rechazo nos llegará por correo electrónico, indicando los motivos en el caso de que no fuera aprobada. El proceso puede demorar algunos días hasta recibir la confirmación, y podremos subir una corrección en caso de no ser aceptada la aplicación en primera instancia.

Fig. 7.15. Dentro de las opciones de Manage Products es posible seguir las revisiones pendientes y conocer sus estados.

Para más información sobre temas relacionados con la publicación y la tienda, es posible consultar la documentación publicada en *BlackBerry Developer Frequently Asked Questions* http://developer.blackberry.com/devzone/blackberryworld/faq/.

Otras tiendas online

Dentro de las alternativas disponibles en el mercado, Windows Phone Store es la tienda de Microsoft que ofrece aplicaciones y juegos para la plataforma Windows Phone. Lanzada en 2010, cuenta desde el 2013 con una oferta que supera las 130.000 apps.

El desarrollo de aplicaciones para Windows Phone puede realizarse mediante Visual Studio, pero también podremos lograr aplicaciones basadas en tecnología Web que luego son empaquetadas con PhoneGap.

Recordemos que esta herramienta permite la creación de aplicaciones para Windows Phone 7, Windows Phone 8, e incluso para la nueva generación de apps de Windows 8, denominadas Windows Store apps.

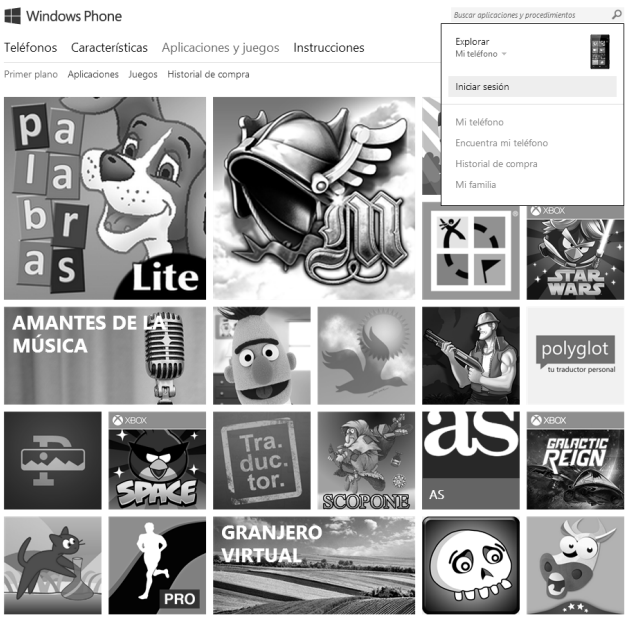

Fig. 7.16. Windows Phone Store se encuentra disponible en la dirección Web: http://www.windowsphone.com/store.

En el Windows Phone Dev Center (http://developer.windowsphone.com/) es posible obtener los SDK para desarrollar apps para Windows Phone (**Get SDK**), y también encontraremos las opciones para publicar una aplicación como desarrolladores.

Para acceder a estas opciones, en primer lugar, podremos utilizar una cuenta de servicios Microsoft (anteriormente conocida como Windows Live ID), y luego suscribirnos para publicar. El valor de la suscripción anual es de 99 dólares estadounidenses.

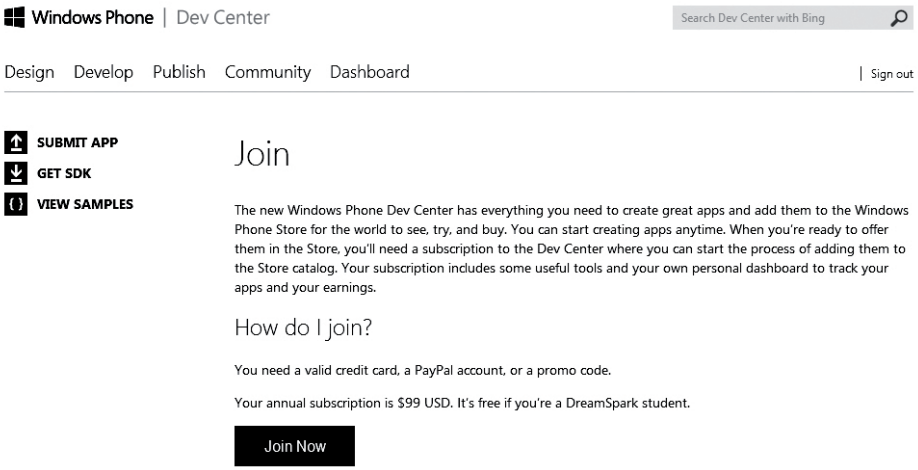

Fig. 7.17. Al adquirir una suscripción como desarrolladores para Windows Phone podremos acceder a herramientas, un panel de control personal y un seguimiento de la evolución de las ganancias.

Por su parte, Amazon es un popular portal de venta de libros y otros productos mediante Internet. Sus dispositivos Kindle cuentan con un sistema basado en Android. Amazon ofrece la posibilidad de distribuir aplicación mediante su tienda Amazon Apps. Las aplicaciones que se suban a esta tienda deben ser compatibles con Android 2.2 o superior. Para registrarnos como desarrolladores debemos ingresar en: http://developer.amazon.com/apps/apps. En este portal, es posible publicar aplicaciones gratuitas o de pago. El costo de registro es de pago anual y equivale a 99 dólares estadounidenses.

Nokia cuenta con una tienda (http://store.ovi.com/) como opción para los usuarios de los dispositivos de esta empresa (previos a los que utilizan sistema Windows). Como desarrolladores podemos registrarnos en: http://www.developer.nokia.com/. El costo para acceder como desarrolladores a este portal es de 1 euro. Las aplicaciones para estas plataformas pueden ser creadas con extensión **.sis** o **.sisx** (para Symbian); **.deb** (para MeeGo); **.sis**, **.sisx**, o **.deb** (para Qt); **.jad** y **.jar** (dispositivos basados en Java), entre otras opciones. La tienda también permite

subir *wallpapers* y contenidos multimedia, como por ejemplo: *ringtones* en formato MP3 o videos en MP4.

Visibilidad en las tiendas

Cuando hablamos de aplicaciones para móviles, su ámbito natural son las tiendas. Por esa razón, los usuarios buscarán las aplicaciones para sus móviles en la versión Web de la tienda o, en muchos casos, mediante la app que ofrece el propio dispositivo para acceder al *store*.

¿Qué podemos hacer para que nuestra aplicación logre una buena visibilidad en la tienda? Como base, deberemos asegurarnos que al realizar los pasos de publicación todos los datos estén completos. La elección del nombre es uno de los puntos clave para la aplicación, ya que será el que la identifique. Es fundamental también escoger las palabras clave que los usuarios suelen buscar en las tiendas, que la aplicación se encuentre en la categoría adecuada, y que las imágenes de la app sean atractivas para que el usuario elija descargar el producto. Recordemos siempre que la mayoría de los usuarios llega a una aplicación por medio de los destacados o por una búsqueda dentro de la tienda.

Vale la pena señalar que en la mayoría de las tiendas los destacados no son pagos, sino que son el resultado de una elección de los editores del *store* o del volumen de descargas que tenga tiene el producto. Por esta razón, resulta siempre importante incluir las imágenes para destacados cuando estamos publicando una aplicación en una tienda, ya que si resulta exitosa, estos contenidos serán los que se utilicen.

Las siglas ASO (*App Store Optimization*) son la nueva tendencia en el universo del posicionamiento de contenidos. La competencia en este campo es cada vez mayor y lograr resaltar nuestro producto y posicionarlo en las tiendas es una tarea importante que no podemos dejar de lado.

Un elemento clave que juega a favor de que una aplicación sea descargada es la calidad, ya que si brinda una buena experiencia a los usuarios atraerá comentarios positivos y también una buena valoración. Estos son datos que los demás usuarios suelen tener en cuenta a la hora de descargar una app.

Finalmente, si decidimos cobrar por las descargas de la aplicación, el precio es un factor que puede influir de manera decisiva en el éxito (o no) del producto. En ocasiones, un precio accesible puede ayudar a ganar un buen volumen de descargas, y de esa manera, equilibrar la balanza para que sea un negocio rentable.

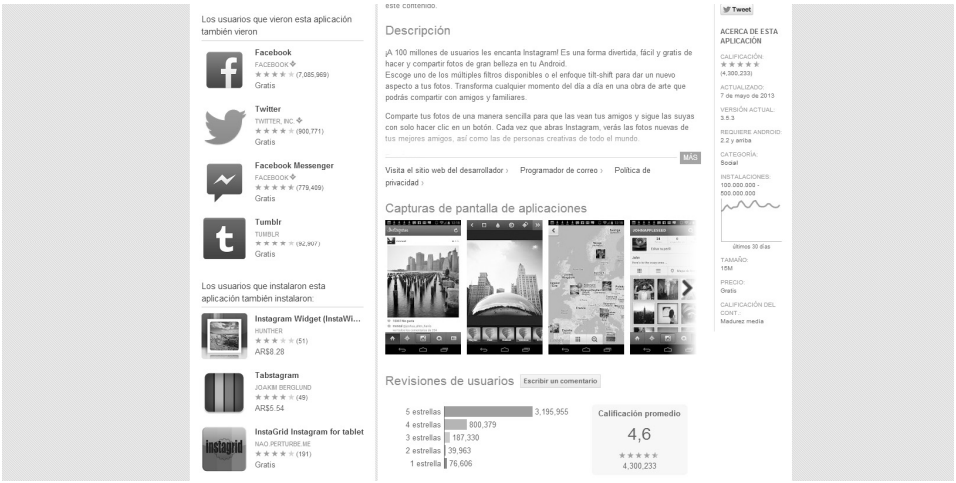

Fig. 7.18. El sistema de calificación por estrellas, las pantallas capturadas de la aplicación, las estadísticas de descargas, y una buena descripción ayudan al éxito de una aplicación dentro de la tienda.

Los servicios sociales. Web social

No es novedad decir que las redes sociales modificaron el mapa de Internet desde su nacimiento en la primera década del siglo XXI. Hoy en día, a pesar de algunos cambios y redefiniciones, siguen siendo un imán muy importante para una gran comunidad de usuarios que utilizan Internet.

Para que nuestra app se haga conocida hay que lograr que todo el mundo "hable de ella". En este campo, el efecto "viral" es fundamental para que otros usuarios deseen descargar nuestro producto. Solo por citar algunos ejemplos, redes sociales como Facebook o Twitter cuentan con millones de usuarios activos, quienes podrían resultar un canal muy efectivo para dar a conocer y difundir nuestra creación.

Otros *tips* interesantes que pueden ayudar a difundir nuestras aplicaciones son:

- Agregar en nuestros perfiles de redes sociales la URL para obtener nuestra aplicación.
- Crear una página en Facebook (https://www.facebook.com/pages/create/) para promocionar nuestra aplicación. Aquí es importante ofrecer mucha información y seguir los comentarios de los usuarios para mantenerlos al tanto de todo y ofrecer respuestas a las consultas.
- Registrar una cuenta de Twitter (https://twitter.com/) para ofrecer a los usuarios una vía de comunicación ágil y rápida.

- Google Plus (https://plus.google.com/) es una alternativa social que permite compartir contenidos y noticias. Es una buena vía para aprovecharla en favor de nuestra aplicación.
- Preparar un tráiler con las bondades de nuestra aplicación y distribuirlo en sitios dedicados a videos como YouTube (http://www.youtube.com/) o Vimeo (https://vimeo.com/).
- Si contamos con sitios o blogs relacionados con el tema, es importante promover la aplicación mediante su presencia en sectores destacados, y también a través de la redacción de artículos que expliquen sus características.
- Crear una micrositio sobre la aplicación o una *Landing Page* permitiría destacar el producto, y funcionar como una pieza importante dentro de una campaña viral.

De cara al futuro

La vida cambia y comienza muchas veces. La evolución de los dispositivos móviles, y las opciones para distribuir y comercializar aplicaciones se encuentran en auge, pero no podemos dejar de estar atentos a los cambios que ocurren en este universo.

¿Qué significa esto? Principalmente, que si nos dedicamos a este rubro no debemos mantenernos quietos, es imprescindible estar atentos a todo lo que está ocurriendo a nuestro alrededor. Sitios especializados, eventos, lanzamientos, capacitación permanente y bibliografía actualizada son algunos de los elementos que nos ayudarán a avanzar en este camino.

En este libro, abordamos el desarrollo para las plataformas que cuentan con la mayor atención del mercado; sin embargo, como hemos visto, existe un gran número de opciones que debemos tener en cuenta para ampliar nuestros horizontes como desarrolladores.

Las técnicas que nos permiten crear aplicaciones para múltiples plataformas se actualizan y evolucionan constantemente. Si deseamos triunfar en este ámbito y no naufragar en el intento debemos estar atentos y seguir las novedades acerca de las plataformas y herramientas que utilizamos para el desarrollo. En este mundo, estar al día no es una opción, es una necesidad que nos brindará frutos hoy y mañana.

Apéndice

A

AJAX

Asynchronous JavaScript And XML (AJAX) es el nombre con el que se conocen las técnicas que revolucionaron el desarrollo Web a mediados de la primera década del siglo XXI. Mediante el uso de estas tecnologías comenzó a ser posible crear aplicaciones enriquecidas, conocidas como *Rich Internet Applications*, de una manera más práctica y eficaz para los desarrolladores.

La evolución de las tecnologías que componen el corazón de AJAX, hoy en día forman parte importante del desarrollo de aplicaciones Web que funcionan tanto en modo *online* como *offline*, y que podemos encontrar en soluciones para diversas plataformas.

A lo largo del presente libro se mencionan conceptos y técnicas relacionadas con AJAX. En este apéndice, encontraremos los aspectos fundamentales para apoyar estos conocimientos y comprender mejor estos lenguajes y tecnologías.

JavaScript

Es el lenguaje que forma parte del corazón de AJAX. Por su versatilidad, JavaScript ha logrado no solo mantener su vigencia en el complejo mundo del desarrollo Web, sino que ha sabido reinventarse en diferentes etapas para ser una solución de muchas necesidades actuales. Es importante destacar que JavaScript es un lenguaje interpretado, y que está orientado a objetos, pero se basa en prototipos.

En este apartado, repasaremos algunas de sus características principales para su correcto uso e implementación en un proyecto.

Variables

Las variables en JavaScript permiten almacenar un valor (numérico, texto, booleano, etc.) y se definen anteponiendo la palabra reservada `var`. Un ejemplo de declaración es el que vemos a continuación:

```
var nombre;
```

El punto y coma (`;`) al final de una declaración es opcional en este lenguaje. Tampoco es necesario declarar un variable previamente a la asignación de su valor. El valor de una variable se asigna con `=`. Por ejemplo:

```
var nombre = "Damian";
```

En JavaScript no es necesario indicar el tipo de dato para las variables que se declaren. Las cadenas de texto deben tener su valor entre comillas. Por su parte, los valores numéricos y booleanos no llevan comillas.

Si deseamos saber el tipo de dato que tiene asignado una variable, podremos recurrir al operador `typeof`, que nos dará como resultado uno de los siguientes valores: `undefined`, `boolean`, `number`, `string`, `object` o `null`.

Las variables tienen un ámbito. Si están declaradas dentro de una función serán locales; y globales, si no se encuentran en dicho ámbito. Si al declarar una variable no se antepone `var`, JavaScript crea una variable global con el identificador especificado y aplica el valor asignado. Por ejemplo:

```
edad = 36;
```

Array

Los *array* son colecciones de datos; y en JavaScript, pueden ser asignados a una variable de diversas formas. Una manera es declarar la variable que contendrá el *array* e inicilizar el objeto *array*, luego entre paréntesis se agregan los valores:

```
var nombres = new Array("Gael","Luis","Mateo");
```

Al igual que una variable, un *array* puede contener diferentes tipos de datos, ya sean cadenas de texto, números, booleanos, etcétera. Para obtener uno de los valores del *array* es necesario indicar el nombre de la variable que lo contiene y, entre corchetes (`[]`), la posición. Vale la pena recordar que los *array* comienzan la cuenta en cero; por lo cual, si deseamos mostrar en consola el primer elemento del ejemplo anterior, escribimos:

```
console.log(elementos[0]);
```

Para asignar un nuevo elemento en el *array* procedemos de la siguiente forma:

```
elementos[3]= "Otro elemento";
```

Para saber la cantidad de elementos que contiene un *array*, utilizamos la propiedad `length`, como vemos a continuación:

```
var cantidadElementos = elementos.length;
```

Estructuras de control

En el mundo de la programación, las estructuras de control son las que permiten modificar el flujo que está en ejecución.

Para evaluar una condición y ejecutar un código (si es verdadera) recurrimos a una estructura `if`, como vemos en el código a continuación:

```
if(condicion) {
  // Código si la condición es verdadera.
}
```

Para establecer una alternativa, si la condición evaluada es falsa, utilizamos `else`:

```
if(condicion) {
  // Código si la condición es verdadera.
}
else {
 // Código si la condición devuelve falso.
}
```

También es posible especificar otras condiciones para evaluar con `else if`:

```
if(condicion 1) {
  // Código si la condición es verdadera.
}
else if(condicion 2) {
 // Código si la condición devuelve falso.
}
else {
```

```
 // Código si la condición devuelve falso.
}
```

Para comparar una expresión con varios valores puede ser recomendable recurrir a una estructura del tipo `select / case`, como vemos en el siguiente ejemplo:

```
switch(variable)
{
case 1:
  // Código para el primer caso.
  break;
case 2:
  // Código para el primer segundo.
  break;
default:
  //Código para el caso que no coincida
    con ninguno de los casos anteriores.
}
```

Para realizar un *loop*, contamos con diferentes alternativas. Si aplicamos `while`, podremos evaluar la condición hasta la cual se realizará el ciclo al comienzo:

```
while (condicion)
  {
  //Código que se ejecuta.
  }
```

Si necesitamos que la evaluación se realice al final, debemos optar por una construcción del tipo `do/while`, como la que vemos en el siguiente ejemplo:

```
do
  {
  //Código que se ejecuta.
  }
while (condicion);
```

Cuando tenemos que construir un ciclo que se cumplirá hasta que se alcance cierto valor, encontraremos una solución eficiente empleando `for`, al que debemos especificarle el valor inicial de la variable, la condición que se debe evaluar hasta

completar el ciclo y el incremento (o decremento). Veamos un ejemplo donde se imprime en pantalla los valores que toma la variable `i` (entre `0` y `10`):

```
for(i = 0; i <= 10; i++) {
document.write(i);
}
```

Si deseamos recorrer todo el *array* y mostrar todos los elementos, por ejemplo, en la consola del navegador, recurrimos a un bucle construido con `for / in`, como vemos en el siguiente código:

```
for(i in elementos){console.log(elementos[i]);}
```

Este tipo de construcción resulta muy útil para diversas necesidades en las cuales es necesario obtener los valores de los elementos de un *array*, y mostrarlos o colocarlos, por ejemplo, dentro de elementos del documento HTML.

Operadores

Como en todo lenguaje de programación, en JavaScript los operadores tienen un rol clave. Los operadores son los que nos permiten asignar valores, plantear comparaciones y realizar operaciones matemáticas. Existen diversos tipos de operadores, que analizaremos a continuación para comprender su correcta aplicación.

Como ya hemos visto, el operador de asignación es el signo igual (`=`). Los operadores de incremento (`++`) y decremento (`--`) nos permiten aumentar o disminuir en uno un valor. Son útiles, por ejemplo, al realizar ciclos. Cuando se ubican delante de la variable actúan como prefijos e incrementan o disminuyen el valor (según corresponda al signo) antes de realizar otra operación. Si por el contrario, estos operadores de incremento o decremento se ubican después de la variable (como sufijo), realizarán la operación que les corresponde después de la sentencia de la cual están declaradas.

Un caso de uso podría ser una estructura `for` donde incrementamos un valor numérico e imprimimos el resultado, como se ejemplifica en el siguiente código:

```
for (var i=1;i<=100;i++)
{document.write(i) + "</ br>";}
```

El ejemplo anterior imprime en pantalla una secuencia de números que van del 1 al 100 (uno debajo del otro). Para hacerlo de manera inversa, es decir, del 100 al 1 (descendente) escribiríamos el siguiente código:

```
for (var i=100;i>=1;i--)
{document.write(i) + "</ br>";}
```

Por otra parte, para evaluar relaciones, tenemos disponibles los siguientes operadores:

- Igual que (==)
- Distinto de (!=)
- Mayor que (>)
- Menor que (<)
- Mayor o igual (>=)
- Menor o igual (<=)

Los operadores lógicos serán nuestros grandes aliados para realizar comparaciones en las estructuras de control que empleemos en nuestro código.

El operador de negación se define con un signo de admiración que cierra (!) y nos cambiar un valor verdadero por uno falso o viceversa.

El operador lógico AND (&&) funciona como un Y que nos devuelve verdadero si los dos elementos que se evalúan son verdaderos. Por ejemplo:

```
if(numero == 1 && color == "black")
{console.log("¡Correcto!");}
```

En el caso del operador lógico OR (||) obtendremos verdadero si al menos uno de los elementos evaluados resulta ser verdadero. Solo devuelve falso si ambos son falsos. Por ejemplo:

```
if(dinero > 1000000 || fortuna == true)
{console.log("¡Eres millonario!");}
```

Los operadores matemáticos en JavaScript están representados por: suma (+), resta (–), multiplicación (*), división (/) y módulo (%).

Declaración de funciones

En JavaScript, podemos trabajar con funciones que permiten ahorrar varias líneas de código y estructurar una aplicación de manera modular. Para quién no está acostumbrado a trabajar con funciones, podemos pensar que están formadas por un grupo de instrucciones que permiten resolver una necesidad de programación específica dentro de nuestro proyecto. La gran ventaja es que pueden ser llamadas desde diferentes lugares de la aplicación y esto hace que puedan ser reutilizadas.

Esto nos permite escribir menos código, facilita las modificaciones, y nos posibilita tener nuestro proyecto organizado.

Para declarar una función se utiliza la palabra reservada `function`, se escribe el nombre que le asignamos a la función y se pasan los argumentos o parámetros que requiere entre paréntesis `()`. Si la función no recibe parámetros, se deben usar los paréntesis vacíos. A continuación, veremos un ejemplo de una función que imprime un mensaje en consola:

```
function consola(){
 var mensaje = "Imprime este mensaje en consola";
 console.log(mensaje);
}
```

Vale la pena señalar que la variable `mensaje` actúa como local dentro de la función `consola()`.

DOM

Las siglas DOM definen el modelo de objetos para representación de los elementos que conforman un documento HTML, XHTML o XML. Para acceder y/o manipular esta estructura se puede trabajar con JavaScript.

Las versiones publicadas de DOM se encuentran clasificadas en diferentes niveles (*level*) por el W3C:

- **DOM Level 1**: recomendación del W3C en octubre de 1998.
- **DOM Level 2**: recomendación del W3C en noviembre de 2000.
- **DOM Level 3**: recomendación del W3C en abril de 2004.
- **DOM Level 4**: al momento de escribir este libro en borrador de trabajo (http://www.w3.org/TR/domcore/).

El árbol del DOM de un documento HTML/XHTML puede estar constituido por los siguientes nodos:

- **Document**: nodo raíz.
- **Element**: elemento (puede contener atributos y otros nodos como hijos).
- **Attr**: atributo de un elemento (es posible acceder al nombre del atributo y a su valor).

- **Text**: texto contenido en un elemento (ubicados entre las etiquetas de apertura y cierre del elemento).

Encontraremos más referencias sobre *Document Object Model* en el documento del W3C: http://www.w3.org/DOM/.

Métodos para acceder a los nodos

La API estándar de DOM nos provee de características que nos brindan acceso directo a los nodos. Los principales métodos son:

- **getElementById()**: obtiene el elemento que coincide con la `id` dada. Ejemplo de uso para obtener un elemento cuya `id` es `caja1`:

```
document.getElementById("caja1");
```

getElementsByTagName(): obtiene los elementos que coinciden con el nombre de etiqueta especificado. Ejemplo de uso para obtener los párrafos (etiqueta `<p>`):

```
document.getElementsByTagName("p");
```

- **getElementsByName()**: obtiene los elementos que cuentan con el valor especificado en el atributo `name` . Ejemplo de uso para obtener los elementos cuyo atributo `name` tiene el valor `producto`:

```
document.getElementsByName("producto");
```

- **getElementsByClassName()**: obtiene los elementos por su clase (este método no funciona en navegadores antiguos). Ejemplo de uso para obtener los elementos que contiene la clase llamada `clase1`:

```
document.getElementsByClassName("clase1")
```

Al acceder a un elemento del DOM es posible modificar características de sus estilos (CSS), como por ejemplo, el color:

```
document.getElementById("p1").style.color="red";
```

También es posible agregar o modificar el texto que contiene en su interior, empleando la propiedad `innerHTML`:

```
document.getElementById("p1").innerHTML="Nuevo texto para
el elemento";
```

Otra posibilidad es obtener (`getAttribute`) o asignar (`setAttribute`) el valor al atributo de un elemento seleccionado:

```
var textoalternativo = document.getElementById
("imagen1").getAttribute("alt");
document.getElementById("imagen1").setAttribute
("src","imagen.png");
```

API de selectores

A partir de HTML5 es posible utilizar métodos nativos para obtener elementos del documento mediante los selectores:

- **querySelector()**: devuelve el primer elemento que coincida con el que se ha especificado como selector. Ejemplo para obtener el primer elemento que contenga la `clase2`:

```
document.querySelector(".clase2");
```

- **querySelectorAll()**: devuelve un *array* con los elementos que coinciden con los que se hayan indicado. Devolverá un *array* vacío en caso de no encontrar coincidencias en la búsqueda. A continuación, vemos un ejemplo para obtener todos los elementos `div` que tienen asignada `clase3`:

```
document.querySelectorAll("div.clase3");
```

Selectors API Level 1 es W3C Recommendation desde el 21 de febrero de 2013: http://www.w3.org/TR/selectors-api/.

La idea detrás de la estandarización de *Selectors API* es proveer una interfaz nativa para realizar selecciones simples o complejas de elementos. Si se está trabajando con jQuery se puede recurrir también a los métodos que ofrece esta librería para acceder a los elementos del DOM: http://api.jquery.com/category/selectors/.

El objeto *classList*

Con HTML5 se incorpora el objeto `classList` para trabajar con clases sobre los elementos del DOM. Los métodos que nos ofrece son:

- **classList.add()**: agrega una clase al elemento seleccionado. Ejemplo para agregar la `clase1` a un elemento

```
elemento.classList.add("clase1");
```

- **classList.remove():** quita la clase especificada del elemento seleccionado. Ejemplo para remover la `clase1` a un elemento:

```
elemento.classList.remove("clase1");
```

- **classList.toggle():** agrega o quita (según corresponda) la clase dada al elemento seleccionado. Ejemplo para agregar/quitar la `clase1` a un elemento

```
elemento.classList.toggle("clase1");
```

- **classList.contains():** devuelve *true* si el elemento elegido tiene la clase especificada; en caso contrario dará *false*. Ejemplo para verificar si el elemento contiene `clase1`:

```
elemento.classList.contains("clase1");
```

Para los ejemplos de código vistos en este apartado, `elemento` puede ser el nombre de una variable previamente definida donde se ha seleccionado la `id` de un elemento, por ejemplo:

```
var elemento = document.getElementById("div1");
```

Los métodos introducidos por HTML5 para acceso o manipulación del DOM pueden no estar disponibles en algunos navegadores antiguos.

JSON

JavaScript Object Notation, conocido por sus siglas JSON, es un formato de intercambio de información que se ha popularizado gracias a AJAX, y se ha transformado en una alternativa eficiente a lo que ofrece XML.

Si bien JSON está basado, de una manera no estricta, en un subconjunto de características de JavaScript, y principalmente lo veremos trabajando con este lenguaje, al ser un formato de intercambio también puede operar como opción para interactuar con otros lenguajes y tecnologías (como PHP o .NET, entre otras).

Es posible crear un objeto de manera simple en JSON de la siguiente forma:

```
{"nombre":"Damian" , "edad":36}
```

Siguiendo el ejemplo anterior, donde indicamos nombre y edad, para incluir un conjunto de valores y conformar una colección de datos en JSON podremos recurrir a la siguiente estructura:

```
{
"personas": [
{"nombre":"Gael" , "edad":7},
{"nombre":"Damian" , "edad":36},
{"nombre":"Horacio" , "edad":60},
{"nombre":"Viviana" , "edad":60}
]
}
```

Al ser un formato nativo de JavaScript podremos aplicarlo sin necesidad de librerías adicionales. Para aplicar lo visto anteriormente en un objeto en JavaScript, lo asignamos a una variable de la siguiente forma:

```
var personas = [
{"nombre":"Gael" , "edad":7},
{"nombre":"Damian" , "edad":36},
{"nombre":"Horacio" , "edad":60},
{"nombre":"Viviana" , "edad":60}
];
```

Para acceder al primer valor y mostrarlo en la consola:
`console.log("Nombre: " + personas[0].nombre + " Edad: " + personas[0].edad);`

Para acceder a todos los valores y mostrarlos en la consola, podremos recurrir a un ciclo `for` que nos permitirá recorrer la colección de datos:

```
for(i in personas)
{console.log("Nombre: " + personas[i].nombre + " Edad: " +
personas[i].edad);}
```

Si necesitamos transformar un objeto JSON en una cadena JSON (por ejemplo para almacenarlo en una base de datos local) recurrimos al método `JSON.stringify()`. Par volver a transformarlo en objeto JSON, empleamos `toJSON()`. Para observar un ejemplo que aplica estos métodos en casos concretos, es recomendable dirigirse al **capítulo 5** de este libro (apartado **HTML5 y JavaScript en nuestra aplicación**).
Para conocer más sobre el formato JSON: http://www.json.org/ (en inglés) y http://www.json.org/json-es.html (en español).

XML es un formato disponible para realizar intercambio de datos mediante AJAX. Precisamente la "X" de AJAX responde a este lenguaje de marcado extensible. La estandarización de XML se encuentra bajo la órbita del W3C, y la documentación de su especificación está disponible en: http://www.w3.org/XML/.

Intercambio de datos de manera asincrónica

XMLHttpRequest (XHR) es la interfaz empleada mediante técnicas AJAX que permite realizar peticiones al servidor. Puede trabajar con los protocolos HTTP y HTTPS. Su desarrollo inicial llegó de la mano de Microsoft para la versión 5 de Internet Explorer, y luego ha sido estandarizado en el resto de los navegadores Web.

Las propiedades con las que cuenta este objeto son las siguientes:

- **readyState**: valor que define el estado de la petición. El valor es numérico y puede ser: **0** (no inicializado), **1** (cargando), **2** (cargado), **3** (Interactivo) o **4** (operación completada).
- **responseText**: respuesta del servidor en formato cadena de texto.
- **responseXML**: respuesta del servidor en formato XML.
- **status**: código de estado HTTP que devuelve el servidor (por ejemplo: 404 para recurso no encontrado).
- **statusText**: código de estado HTTP en forma de cadena de texto.

Los métodos disponibles para este objeto son:

- **abort()**: aborta la petición.
- **getAllResponseHeaders()**: devuelve una cadena que contiene las cabeceras de la respuesta.
- **getResponseHeader()**: devuelve una cadena con la cabecera especificada.
- **open()**: se pasan los parámetros de la petición: método HTTP (GET, POST o PUT) y la URL destino.
- **send()**: hace el envío de la petición HTTP (se pasan los datos).
- **setRequestHeader()**: permite especificar cabeceras personalizadas en la petición.

Los eventos relacionados con este objeto son:

- **onreadystatechange**: ocurre cuando hay un cambio de estado.
- **onabort**: ocurre cuando se aborta una operación.
- **onload**: ocurre cuando se completa la carga.

- **onloadstart**: ocurre cuando se inicia la carga.
- **onprogress**: se dispara con la información de estado al progresar la carga.

A continuación, analizaremos un ejemplo en el cual, empleando esta técnica, traeremos un archivo llamado **datos.txt**, y lo mostraremos en un elemento del DOM, cuya `id` es `div1`:

```
function cargardatos(){
var xmlhttpreq = new XMLHttpRequest();
xmlhttpreq.onreadystatechange=function()
  {document.getElementById("div1").innerHTML=
    xmlhttpreq.responseText;}
xmlhttpreq.open("GET","datos.txt",true);
xmlhttpreq.send();
}
```

La función `cargardatos()`, al ser invocada, en primer lugar, crea una instancia del objeto XMLHttpRequest. Luego se especifica en qué elemento se colocará el contenido que se obtenga (en este caso, un contenido en formato de texto). El archivo **datos.txt** se obtiene mediante GET.
La documentación de esta especificación está publicada en:

http://www.w3.org/TR/XMLHttpRequest/.

Al escribir este libro XMLHttpRequest Level 2 se encontraba disponible como borrador de trabajo ()(http://www.w3.org/TR/XMLHttpRequest2/). Esta actualización del estándar incorpora ventajas para el trabajo *cross-origin* y el manejo de flujos de datos.

jQuery y AJAX

Una librería que ha sido utilizada en algunos pasajes de este libro para facilitar el trabajo con JavaScript y AJAX, es jQuery (http://jquery.com/). Para intercambiar información cliente/servidor mediante AJAX, jQuery ofrece varios métodos que facilitan este tipo de operaciones. Los principales métodos son los siguientes:

- **get()**: permite cargar información desde el servidor mediante una petición HTTP del tipo GET. Más información y ejemplos de uso de este método en la documentación de jQuery: http://api.jquery.com/jQuery.get/.

- **getJSON()**: permite cargar datos codificados en formato JSON mediante una petición al servidor utilizando GET. Más información y ejemplos de uso en la documentación de jQuery: http://api.jquery.com/jQuery.getJSON/.
- **post()**: permite cargar información desde el servidor mediante una petición HTTP del tipo POST. Más información y ejemplos de uso de este método en la documentación de jQuery: http://api.jquery.com/jQuery.post/.
- **load()**: permite cargar datos e insertarlos en el elemento del DOM seleccionado. Más información y ejemplos de uso de este método en la documentación de jQuery: http://api.jquery.com/load/.

Para entender mejor estas características y comprender cómo se puede obtener información situada en el servidor, veremos un ejemplo. Para este caso, combinaremos el uso del método `$.getJSON()` de jQuery con la posibilidad de obtener un conjunto de valores en formato JSON generados en el servidor, mediante PHP.

Como no es objeto de este libro entrar en detalle de PHP, utilizaremos un ejemplo sencillo donde generaremos un *array* asociativo y lo convertiremos en JSON mediante el método `json_encode()`. De esta manera, el código para el archivo que llamaremos **json-datos.php** queda definido de la siguiente forma:

```
<?php
    $persona = array(
    'nombre'=>"Damian",
    'apellido'=>"De Luca",
    'pais'=>"Argentina");
    echo json_encode($persona);
?>
```

Ahora, podremos crear el archivo HTML que interactuará con el PHP. En este caso, es importante recordar la inclusión en la cabecera de la llamada al archivo jQuery para trabajar, y luego se puede escribir el *script* como vemos a continuación:

```
<script>
$(document).ready(function(){
  $('#boton').on('click', function(){
  $.getJSON('json-datos.php', function(data) {
  $('#info').html("<p>Nombre: "+data.nombre+"
  Apellido: "+data.apellido+" Pais:
  "+data.pais+"</p>");
        });
```

```
    });
});
</script>
```

El código anterior se dispara una vez que el usuario hace clic sobre un elemento cuya `id` es `boton`. En ese momento, se llama al archivo que hemos definido anteriormente como **json-datos.php** para obtener el *array* que escribimos en PHP.
Una vez que se obtienen los datos, empleando el método de `html()` de jQuery, son colocados dentro de un elemento HTML cuya `id` es `info` (podría ser, por ejemplo, un `<div>`).

Como podemos observar, `data` contiene los valores extraídos del JSON, y para acceder a cada uno se utilizan: `nombre`, `apellido` y `pais`, es decir, los identificadores que se establecieron en PHP.