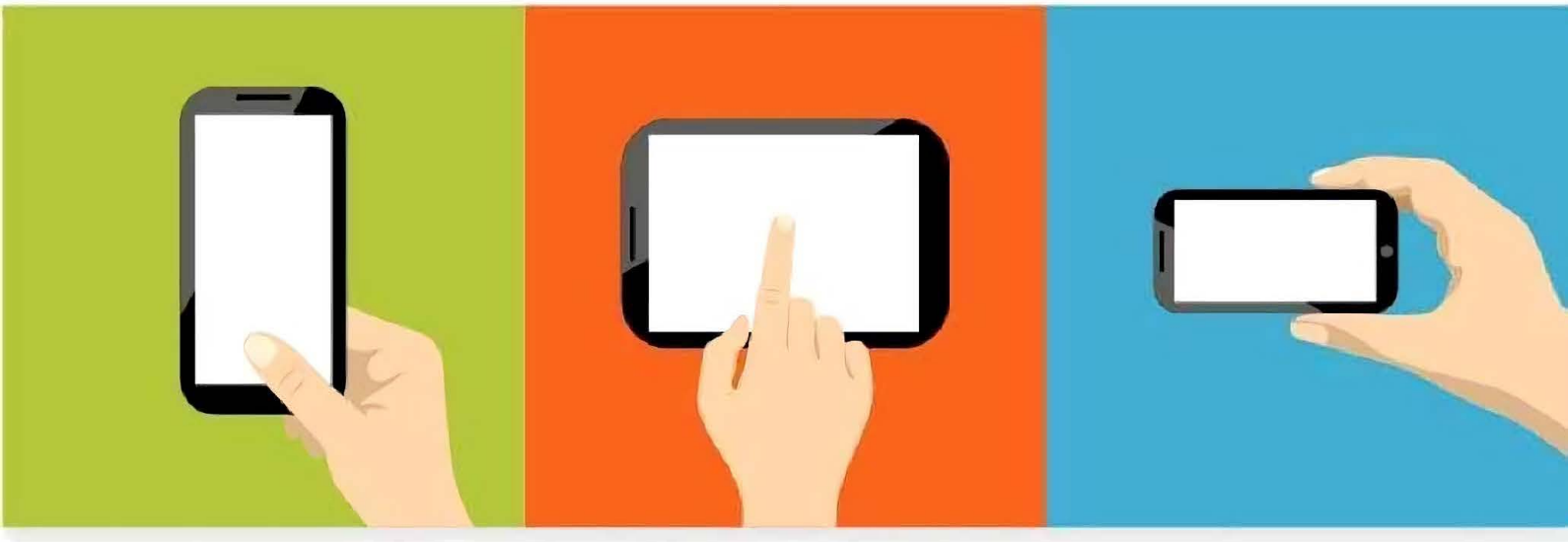
USERS

INCLUYE VERSIÓN DIGITAL GRATIS

DESARROLLO WEB PARA DISPOSITIVOS MÓVILES

GEOLOCALIZACIÓN Y MAPAS WEB + HTML5 PARA MÓVILES + JQUERY MOBILE PARA SITIOS MULTIPLATAFORMA + INTEGRACIÓN DE WEBAPPS CON REDES SOCIALES + WEBAPPS COMO APLICACIONES NATIVAS + ACCESO A DATOS LOCALES Y REMOTOS + USO DE SISTEMAS DE MENSAJERÍA + PHONEGAP Y APLICACIONES HÍBRIDAS



PROGRAMACIÓN WEB PARA SMARTPHONES Y TABLETS

RU

DESARROLLO WEB PARA DISPOSITIVOS MÓVILES

HERRAMIENTAS PARA DISEÑAR Y PROGRAMAR WEBAPPS

por Fernando Luna

RedUSERS

TÍTULO:	Desarrollo web para dispositivos móviles
AUTORES:	Fernando Luna
COLECCIÓN:	Manuales Users
FORMATO:	24 x 17 cm
PÁGINAS:	320

ISBN 978-987-1949-83-0

Luna, Fernando

Desarrollo web para dispositivos móviles. - 1a ed. - Ciudad Autónoma de Buenos Aires : Fox Andina; Buenos Aires: Dalaga, 2014.

320 p. ; 24x17 cm. - (Manual users; 269)

ISBN 978-987-1949-83-0

1. Informática. I. Título

CDD 005.3

Fernando Luna

Es diplomado en desarrolllo de aplicaciones para dispositivos móviles y actualmente se desempeña como analista funcional de sistemas. Realiza tareas de relevamiento, diseño, prototipado y especificaciones técnicas para brindar soluciones informáticas.

Colabora con las revistas *USERS* y *POWER USERS* y en varios blogs de programación y tecnología.

En el año 2011 lanzó su primer libro, *VISUAL BASIC 2010 – Manual del programador.*

E-mail: fernando@vidamobile.com.ar

Twitter: @mobilepadawan

Facebook: facebook.com/ferpro

Dedicatoria

Esta nueva obra está dedicada a mis hijos Nicolás y Julián, que son el combustible que impulsa mi motor de aprendizaje y constante mejora.

A mi compañera de vida, Laura, quien me brinda la perfecta armonía emocional; y a mis viejos, quienes siguen iluminando desde el éter mi camino por este mundo.

Agradecimientos

Sin duda no puedo dejar de agradecer a la editorial por el constante apoyo y orientación para que mi segundo sueño se convierta en realidad. También a mi familia, quienes siguen respetando mis tiempos y acompañándome en el sueño de ser autor.

Prólogo

Mi infancia estuvo acompañada de algún que otro juguete electrónico y también de la pasión de mi padre por la música contemporánea, quien hizo que me criara desde temprana edad rodeado de tocadiscos, combinados y grabadores monoaurales.

Por suerte (o por desgracia), la profesión de mi progenitor lo obligó a tener soldadores y destornilladores de todo tipo, y fueron estas herramientas las culpables de incitarme a saciar mi fanatismo por la electrónica, a conocer más de cerca cada componente que conformaba los artefactos.

En mi adolescencia me ocupé de aprender los secretos de la electrónica aplicados al mundo de la radio, el audio y la computación, en el momento en que a esta última aún se la llamaba "electrónica digital".

Ya con la pedagogía moldeada, gracias a mi amigo Hernán, del maravilloso mundo del silicio, las placas conductoras y las membranas dieléctricas, realicé un paso fugaz por el mundo de las comunicaciones entre radioaficionados.

Luego llegó mi primera computadora, una 80286 con 2 MB de memoria RAM, con la que comencé a dar mis primeros pasos en el ámbito de la computación: primero, como operador ofimático y diseñador gráfico; después llegaron la animación y el diseño asistido por computadora; y, por último, la programación. En mi caso, la última fue la vencida.

Los últimos protagonistas que generaron interés en mí fueron los BBS, seguidos de internet. Estas inmensas centrales de información que crecen día a día terminaron de forjar mis conocimientos y mi pasión por este maravilloso mundo digital.

La evolución tecnológica siguió avanzando, y hoy encuentro conjugados, en cualquier dispositivo móvil, los cuatro elementos que marcaron mi pasión por el desarrollo de aplicaciones para este pequeño mundo de bolsillo: la electrónica, la informática, las comunicaciones y la programación. Esta última es, sin dudas, la que me permite estructurar y moldear la manera en que quiero disfrutar de las otras tres.

Fernando Omar Luna

El libro de un vistazo

Esta obra está destinado a diseñadores web, programadores y entusiastas de la tecnología que deseen crear o adaptar un sitio web a los dispositivos móviles. A lo largo del libro brindaremos conocimientos útiles para diseñar y programar WebApps dinámicas y explotaremos las capacidades del hardware de estos dispositivos.

*01 PLATAFORMAS Y TECNOLOGÍAS MÓVILES

Como usuarios de equipos móviles, seguramente pensamos que la revolución de estos es muy reciente, pero en verdad lleva décadas gestándose. Conoceremos la historia y evolución de los dispositivos móviles, los principales competidores y las herramientas necesarias para ingresar en el desarrollo de aplicaciones.

*02 HTML5

HTML5 es el lenguaje que conjugó la transición de sitios web, de escritorio a móviles. Conoceremos sus funciones destacadas e ingresaremos al mundo de las comunicaciones de la mano de la geolocalización.

*03 FUNDAMENTOS DE JQUERY MOBILE

jQuery Mobile brinda a diseñadores y programadores herramientas para crear una estructura web móvil y funcional. Aprenderemos a utilizar este framework, que nos solucionará fácilmente la ardua tarea de diseñar estéticamente una web homogénea y multiplataforma.

*04 COMPONENETES DE JQUERY MOBILE

Con las nociones básicas de este framework, explotaremos las opciones que este nos brinda a través de los widgets. Componentes que nos permitirán desarrollar soluciones para las diferentes plataformas móviles en las que se desplegará nuestro proyecto web.

*05 INTERACCIÓN CON EL HARDWARE DE COMUNICACIONES

Habiendo consolidado el lenguaje HTML5 con jQuery Mobile y JavaScript, este capítulo nos permitirá explorar al máximo las diversas capacidades de comunicación que puede desarrollar una web móvil en los smartphones y tablets.

*06 LENGUAJES DE PROGRAMACIÓN

El contenido dinámico de la Web nos permite complementar HTML5, jQuery Mobile, JavaScript y el acceso al hardware del dispositivo con el lenguaje de programación PHP y la base de datos MySQL, para desarrollar soluciones móviles desentendidas del contenido estático.

*07 ALMACENAMIENTO LOCAL Y APLICACIONES OFFLINE

Otro avance surgido de la plataforma HTML5 es la capacidad de almacenar información en el navegador del cliente. Aquí nos introduciremos en los fundamentos básicos de la utilización de Web Storage, Web SQL, Indexed Database y AppCache.

*08 WEBAPPS PARA iOS

El sistema operativo móvil de Apple presta novedosas capacidades para WebApps. Aquí las aprovecharemos, modificando de forma transparente nuestras aplicaciones web móviles para que formen parte del escritorio de iOS como una aplicación nativa.

*09 WEBAPPS PARA ANDROID Y OTROS DISPOSITIVOS

Android adoptó las capacidades prestadas por el navegador web Safari. Conoceremos cómo agregar una aplicación web al escritorio de Android. Luego repasaremos las soluciones que otras plataformas ofrecen en este terreno.

*10 PHONEGAP

En este capítulo introduciremos el framework PhoneGap y el servicio PhoneGap Build, que permite convertir una WebApp para móviles en una app que podrá distribuirse en las principales tiendas de aplicaciones móviles.

*ApA BB10, WINDOWS PHONE Y EMULADORES WEB

Agrupamos aquí las bases para utilizar las herramientas de testing que ofrecen los diferentes fabricantes de dispositivos móviles. Esto nos permitirá probar los desarrollos en las distintas plataformas virtuales, en caso de no poseer los dispositivos físicos.

*ApB FIREFOX OS

Este apéndice engloba los fundamentos de Firefox OS, su simulador y la arquitectura de sus aplicaciones. También veremos las bases para distribuir nuestras WebApps en Mozilla Marketplace.

INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual, podrá encontrar una serie de recuadros que le brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados. Para que pueda distinguirlos en forma más sencilla, cada recuadro está identificado con diferentes iconos:


CURIOSIDADES E IDEAS


ATENCIÓN


DATOS ÚTILES Y NOVEDADES

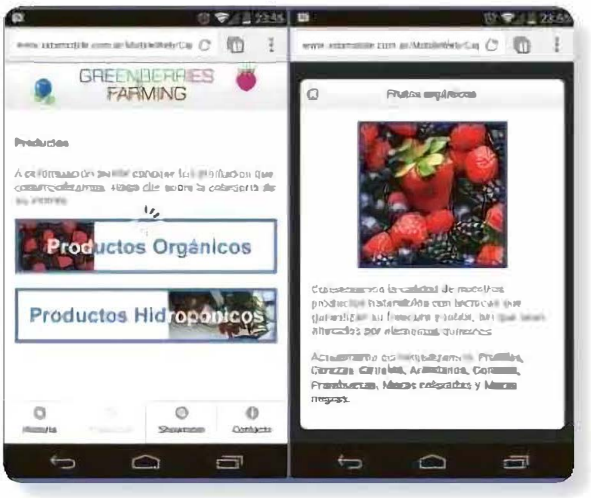

SITIOS WEB

Contenido

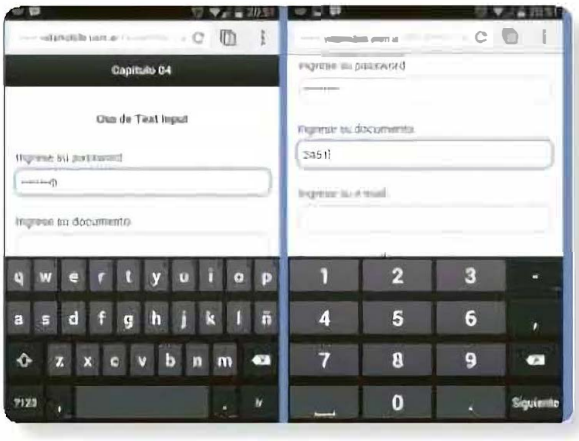
*01 Plataformas y tecnologías móviles

*02 HTML5

*03 Fundamentos de jQuery Mobile

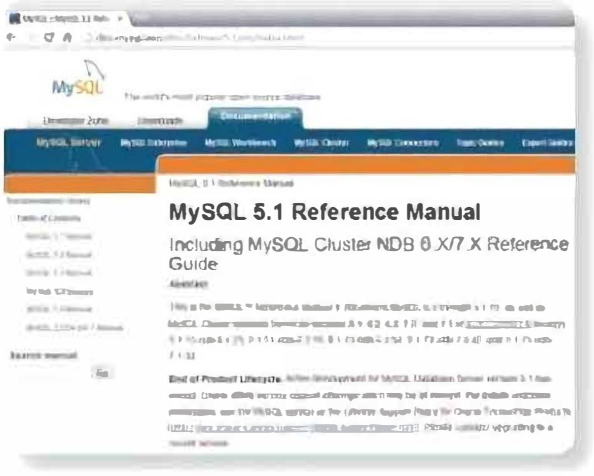


*04 Componentes de jQuery Mobile

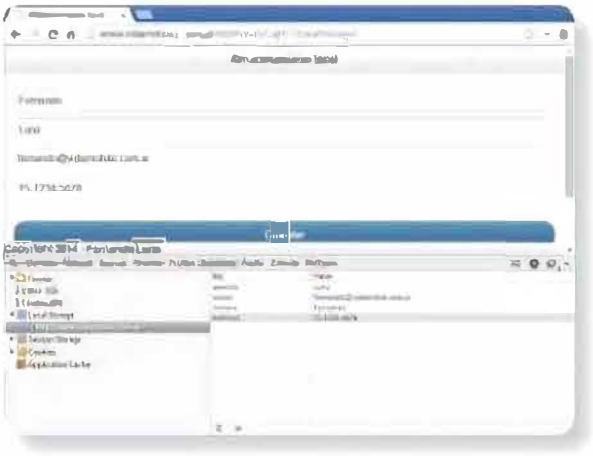


*05 Interacción con el hardware de comunicaciones

*06
Lenguajes de programación



*07
Almacenamiento local y aplicaciones offline



*08
WebApps para iOS

*09
WebApps para Android y otros dispositivos

*10

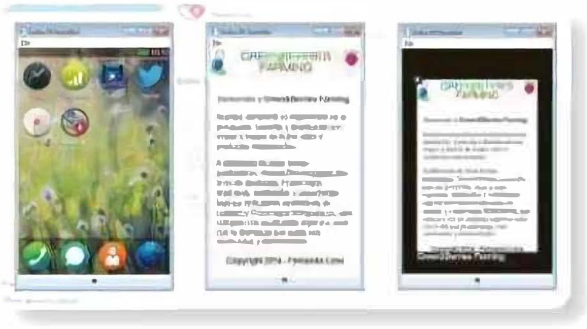
PhoneGap

*ApA

BB10, Windows Phone y emuladores web

*ApB

Firefox OS



Introducción

Si segmentamos los notables avances que tuvieron lugar en las últimas dos décadas en materia de evolución tecnológica, podemos resumirlos en tres actores importantes: la computadora personal, internet y la telefonía móvil. Debemos destacar, además, que este último actor tuvo su propio Big Bang dentro de la evolución mencionada, dado que no solo permitió comunicar a las personas en cualquier punto donde se encuentren, sino que también supo tomar lo mejor de los primeros dos actores y conjugarlo en un único producto.

De este modo, la telefonía móvil logró consolidarse como el punto de partida de todas las actividades relacionadas al uso cotidiano de servicios. A través de ella, hoy en día podemos realizar múltiples y variadas tareas, entre las que podemos mencionar: establecer contacto social con nuestros pares, enviar y recibir mensajes de correo electrónico, mirar nuestras series o películas favoritas, realizar llamados telefónicos, leer libros electrónicos, establecer una agenda de actividades, comprar productos y servicios, navegar por internet y hasta preparar una receta de cocina.

Como desarrolladores de aplicaciones informáticas buscaremos, a través de esta obra, explotar estas nuevas tecnologías a partir del desarrollo de soluciones que aprovechen los recursos que nos brinda hoy un teléfono inteligente o una tablet.

Gracias a la constante actualización de la tecnología basada en la Web, estas soluciones pueden llevarse adelante tanto desde una aplicación nativa instalada en un dispositivo como desde una página web cargada en nuestro navegador móvil favorito.

Los invitamos a descubrir, en las próximas páginas, cómo podemos explotar cada una de las características que brindan los dispositivos móviles, enfocándonos en la arquitectura de hardware y software que cada uno de estos pequeños artefactos pone a disposición del usuario.

*01

Plataformas y tecnologías móviles

En este capítulo conoceremos la evolución de la tecnología móvil y web. Veremos cómo esta última se adaptó al pequeño mundo de los dispositivos móviles haciendo que un simple teléfono se convirtiera, en corto tiempo, en un smartphone. También conoceremos los principales sistemas operativos móviles, las diferencias entre WebApp, app nativa y app híbrida, y cuáles son los frameworks más populares.

Introducción a la Web móvil

Para quienes somos fanáticos de la tecnología móvil, "**mobile**" es la palabra que más suena en nuestras mentes desde, aproximadamente, el año 2007. Si bien los dispositivos móviles se integraron en nuestra vida dos décadas antes, podemos tomar como quiebre, en el segmento de las tecnologías de bolsillo, la presentación de **iPhone** dentro de este mercado, en aquel año.

El mundo IT ya venía trabajando, desde algunos años antes, en la reinvención del terreno de dispositivos móviles, para que estos ganaran un papel mucho más importante en nuestras vidas. Dentro de los dispositivos que más se vendieron en esta última década, podemos destacar **netbooks**, **tablets**, **handhelds** y **smartphones**.

Algunos de estos dispositivos lograron aceptación, mientras que otros tuvieron que esperar un poco la evolución del mercado y encontrar alternativas que permitiesen involucrar más la atención del usuario final.

Dentro de aquellos que llegaron a destiempo, podemos destacar las tablets, dado que Apple había lanzado una primera versión de estas en los inicios de la década del 90, con un equipo bautizado como **Apple Newton**. Si bien la iniciativa no tuvo, en general, mucha aceptación, comparada con el auge de **iPad** o **Android** actualmente en este segmento, al menos podemos reconocer que dio pie al surgimiento de otras ideas que, con el tiempo, tomaron vuelo y ganaron más interés por parte de los usuarios finales, como la **PDA Palm Pilot**.

> LA PRIMERA VERSIÓN DE UNA TABLET FUE LA APPLE NEWTON, LANZADA EN LA DÉCADA DE 1990

Los equipos ultra portátiles o netbooks hicieron su primera aparición de la mano de los handhelds presentados por **Casio** y **COMPAQ**, a fines de la década del 90. **Casiopeia A-11** y **COMPAQ C-2010** fueron los primeros dispositivos móviles que buscaron acaparar el lugar que las netbooks alcanzaron entre 2006 y 2011 en el mercado de computadoras.

Este tipo de dispositivos, junto con los escáneres de mano **Psion**, rescataron el sistema operativo **Windows CE** pensado originalmente para los handhelds de Casio y COMPAQ. Hoy, Windows CE sigue vigente y presta servicios a equipos Psion, entre otros.

Figura 1. **COMPAQ C2010** y **Casiopeia A-11** (1998) fueron las primeras netbooks, mientras que **Apple Newton** (1993) fue la primera tablet que inspiró el lanzamiento de los **PDA**.

En el segmento de los smartphones, podemos destacar que la llegada temprana de estos dispositivos al mercado de telefonía móvil fue de la mano de **Nokia**. Algunos años más tarde se sumó la firma **Sony Ericsson**.

Nokia lanzó, en marzo de 1996, el teléfono computadora **Nokia 9000 Communicator**, dispositivo que tuvo algunas evoluciones y finalizó su existencia en el 2004 con el modelo **Nokia 9500 Communicator**. Su última versión tenía **Bluetooth**, **Wi-Fi**, teclado físico **QWERTY** y un sistema operativo gráfico cuyo navegador soportaba páginas en **Flash**. El Nokia 9000, el primero de la serie, pudo conocerse en la película *El Santo*, en la que el personaje principal utilizaba el servicio de múltiples cuentas de **e-mail** y hasta recibía faxes a través de este teléfono.

Algunos años después, Sony Ericsson llegó al mercado con su modelo **P990**, un teléfono táctil con pantalla gráfica de 256 colores y **stylus pen** para el manejo de las aplicaciones en pantalla.

Figura 2. De izquierda a derecha: **Sony Ericsson p990**, **Palm Pilot** y **Nokia Communicator 9500**, equipos considerados como los primeros smartphones.

Hoy podemos decir que gran parte de los fracasos de esa época pudo deberse a la ausencia de aplicaciones que dieran valor agregado a estos dispositivos. Algunos de estos terminales tuvieron una tienda de aplicaciones propia, pero estaba limitada a los desarrollos de las respectivas empresas o de algunos socios muy cercanos a estas.

Un nuevo panorama

En 2007, el **panorama mobile** cambió por completo. **Apple** presentó el iPhone, las netbooks explotaron en ventas y, un año más tarde, aparecieron las tablets. Así fue como la tecnología que había surgido entre diez y quince años antes reapareció en el mercado comercial con otro formato y de la mano de nuevos jugadores. Junto con ellos, llegaron los **app stores** o tiendas de aplicaciones. Estos factores permitieron abrir el mercado de ganancias, no solo a las corporaciones que desarrollan estos productos, sino a cualquier particular o pequeña corporación. Dentro de estos mercados, los que más generan ingresos hoy son los de la telefonía móvil, con su batería de apps gratuitas y pagas, seguidos por el mercado de las tablets.

En algunos países como Argentina, la telefonía celular acaparó el mercado de manera impensada, superando en número a la telefonía fija e incluso a la población de toda una nación.

En un estudio llevado adelante en 2013, la población argentina se estimó en 45 millones de habitantes. La telefonía fija se calculó en 12.5 millones, mientras que la telefonía móvil era de 52.3 millones de celulares en uso. En el último lustro, la telefonía móvil superó con creces a la cantidad de líneas telefónicas de hogares y, en muchos casos, el teléfono celular reemplazó por completo a las líneas fijas.

Con la proliferación de líneas telefónicas móviles y abonos para las conexiones de datos como **2G**, **3G** y **4G**, algunas netbooks y tablets incorporaron en sus modelos la opción de instalar un chip **GSM** para realizar llamadas, enviar SMS o navegar por la Web.

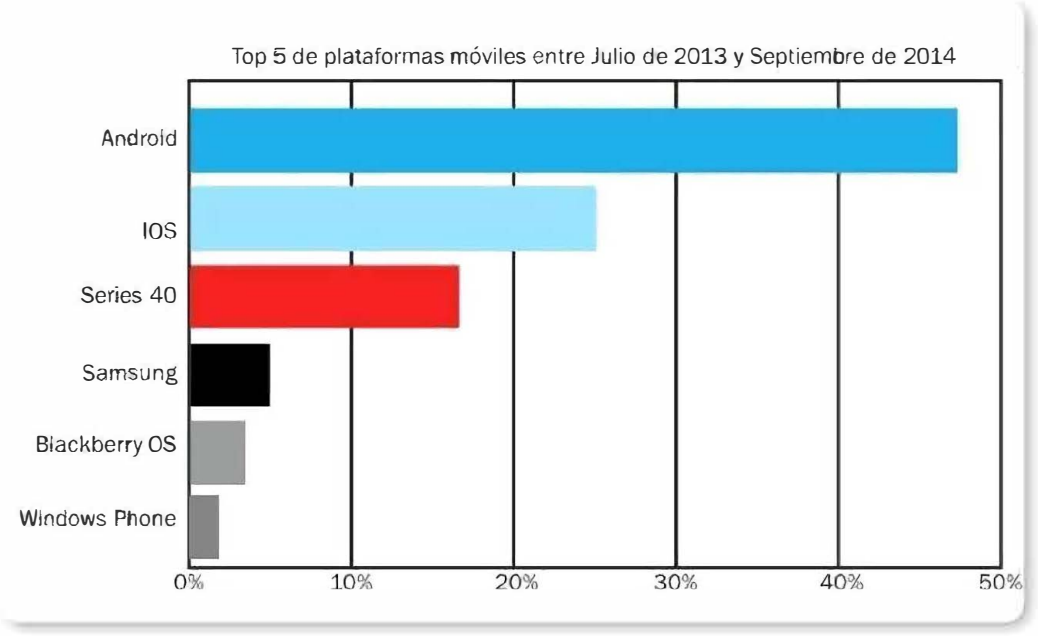


Figura 3. Con una clara ventaja por sobre la competencia, **Android** dominaba, a fines de 2013, el mercado de los smartphones, donde lleva más de 250 millones de equipos vendidos.

En 2009, previo al lanzamiento del **iPad**, muchas empresas quisieron adelantarse al rumor de la tablet de Apple y presentaron su alternativa: hicieron funcionar, en modo táctil, sistemas operativos como **Windows Vista** y **Seven**, y algunos, Android. También llegaron los híbridos: netbooks que permitían torcer su pantalla para convertirla en un dispositivo táctil.

Pese a los esfuerzos realizados por las firmas IT, iPad rompió todas las reglas y no consiguió tener un competidor decente que le hiciera frente.

El mundo web

Si bien en las líneas anteriores describimos lo que es una **plataforma móvil**, podríamos, ahora, definir el concepto: se trata de todo aquel dispositivo de fácil traslado, que dispone de determinadas prestaciones que nos ayudan en el desempeño diario de nuestras actividades laborales y/o académicas.

Entre las características más importantes de una plataforma móvil, podemos destacar:

- Sistema operativo amigable.
- Conectividad a internet.
- Disponibilidad de aplicaciones para múltiples propósitos.
- Múltiples opciones de comunicación con nuestros pares.
- Integración de correo electrónico y otros medios de comunicación.
- Almacenamiento de documentos varios.
- Acceso a contenidos multimedia.

Dependiendo de la empresa fabricante por la que optemos, podremos tener más o menos servicios integrados en el dispositivo. Sin embargo, en las plataformas móviles consideradas de "gama media" en adelante, contamos con todas las características listadas.

Hoy el mayor uso de plataformas móviles pasa por los smartphones o teléfonos inteligentes; luego, por las tablets; y, por último, las netbooks o **ultrabooks** (una categoría que se integró en 2012).

Las tablets vienen creciendo fuertemente en el campo de implementación académico, laboral y, por último, personal, y desde el

INICIOS DE LA TECNOLOGÍA WAP

Wireless Application Protocol nació como un estándar abierto para aplicaciones que se comunican de manera inalámbrica. Así, se estableció un conjunto de protocolos que normalizó la manera de entenderse entre dispositivos inalámbricos, con el objetivo de acceder a los principales servicios web (e-mail, newsletter, web).

año 2012 su venta supera al comercio de computadoras de escritorio y portátiles.

Entre todas las prestaciones que los smartphones nos brindan, vale destacar que la mayoría de estas pasan hoy por un mismo canal: la Web.

El mundo web se inició en 1994 de forma masiva y, en 1995, ya se había expandido hacia numerosos nichos, tanto corporativos como académicos y particulares.

El desarrollo de la Web, tal como la conocemos, partió de la masificación de sitios corporativos, del uso cotidiano del correo electrónico, de la integración de salas de chat y sistemas de mensajería instantánea, dentro de los cuales destacamos a **ICQ** como el pionero en este campo.

Con el tiempo, también se introdujeron mejoras, como ser: la música digital, la visualización de videos, la integración de plataformas de desarrollo de webs dinámicas de manera más amigable y, por supuesto, la potenciación de la estética de los sitios web de la mano de la casi extinta tecnología Flash.

Figura 4. **ICQ**, la **Web estática, Internet Explorer** y **Macromedia Shockwave Player** son algunas de las primeras tecnologías web en el nacimiento de internet.

El mundo mobile web

Si bien iPhone replanteó el concepto de la Web en los dispositivos móviles al modificar su navegador web, **Safari**, con determinadas características **HTML** y **CSS** propias de una pantalla de menos de 4 pulgadas, la Web móvil ya existía en la telefonía celular desde inicios de la década de 2000.

Muchos teléfonos –como el Nokia Communicator o Sony Ericsson p990– ya incorporaban un navegador web en sus equipos móviles. Y los handheld COMPAQ y Casio de fines de la década del 90 también poseían una versión reducida de IE, bautizada como **Pocket Internet Explorer**.

Hacia fines de los 90, los handheld poseían una pantalla de 7 pulgadas que permitía contar con resoluciones de **800x480** u **800x600 pixeles**, casi iguales a la resolución de un monitor de 14 pulgadas de aquella época. Esto no impedía que la Web en sí se reprodujera en casi toda su totalidad en estos equipos.

Sin embargo, con el tiempo los smartphones coparon el terreno y achicaron sus pantallas con el afán de ocupar lo mínimo indispensable, y la Web en sí debió mutar hacia la Web móvil, una alternativa visible, rápida y de muy bajo consumo en KB, que podía reproducirse de manera decente en las pantallas de los teléfonos móviles.

La **tecnología WAP** fue la que llevó internet a los teléfonos móviles. Era muy lenta y el valor de la facturación por uso de kilobytes de consumo era igual o más alto que un minuto de aire de una llamada móvil. Aun así, logró una fuerte adopción en un corto tiempo.

¿Cuándo se conjugaron web y mobile web?

Las tecnologías 2G y 3G, junto a la conectividad Wi-Fi portada a los móviles, hicieron posible que la telefonía móvil explotara en todo su esplendor.

iPhone 1 implementó una serie de **<tags HTML>** a la versión móvil del navegador web Safari. Este fue el inicio de las investigaciones de nuevas opciones que permitirían distinguir una conexión de computadora normal de una conexión móvil.

EL IPAD EN LA UNIVERSIDAD

Desde el lanzamiento de iPad en 2010, muchas universidades empezaron a tomar en serio este dispositivo como herramienta de aprendizaje. Una de las primeras fue Seton Hill University (**www.setonhill.edu**), que desde abril de ese año comenzó a utilizar este dispositivo como herramienta para todos sus alumnos.

Con la implementación de **tags** propietarios para mobile, sumada al hecho de que **JavaScript** siempre estuvo presente en este tipo de dispositivos, comenzaron a crearse alternativas que combinaban las tecnologías HTML, CSS y JavaScript para mejorar el terreno móvil y lograr, así, mejores productos basados en la Web.

Figura 5. La evolución de la Web móvil desde la era **WAP** a inicios de 2000, escalando en la evolución **iPhone** y finalizando en las actuales aplicaciones web.

La tecnología Flash no se vio promovida sobre la plataforma web porque, para la carga de un **sitio Flash** en una computadora de escritorio, se requería de un tiempo que iba de unos segundos hasta al menos un minuto. Esto se traducía en una importante cantidad de minutos de demora, sumados a la baja memoria **RAM** y los escasos **Megahertz** de los procesadores de aquella época.

Luego, para terminar de promover las tecnologías móviles y conjugarlas con el menor impacto posible con las tecnologías de la Web básica, se inició un plan que permitiera, mediante HTML y CSS, unificar el desarrollo web para más de una plataforma.

Esto es lo que se conoce hoy como **HTML5**. No solo es el principal responsable de la unificación web (desktop y mobile), sino que se integra con CSS y JavaScript de manera tal que las tres tecnologías brindan las herramientas necesarias para lograr una única web, con el menor esfuerzo posible, para todas las plataformas.

Plataformas móviles

El mercado mobile dispone de un número importante (y difícil de calcular) de plataformas móviles que agrupan tanto a los teléfonos inteligentes como a las tablets y dispositivos híbridos (aquellos que conjugan una tablet con una computadora portátil).

Dentro de este mercado amplio y lleno de marcas y modelos, lo que más importa a un desarrollador es la plataforma basada en software, ya que la Web móvil se enfoca siempre en el software, que es el encargado de interactuar con el hardware en sí.

Por ello, nos centraremos en conocer los sistemas operativos que actualmente existen, con sus limitaciones y coincidencias en el mundo móvil.

iOS

Este sistema operativo es propio de la firma Apple. Fue desarrollado en 2006, para ser integrado al lanzamiento de iPhone en el año siguiente. Inicialmente este sistema operativo no tenía un nombre definido tal como lo conocemos ahora; simplemente, se lo llamaba **iPhone OS**.

Apple afirmaba que iPhone corría una versión adaptada de OSX. **iOS** obtuvo su nombre oficial recién a principios de 2008, cuando Apple oficializó el lanzamiento del **iPhone SDK**, que permitía a cualquier desarrollador crear aplicaciones para este mercado.

Con el tiempo, iOS se adaptó a las versiones táctiles de iPod, iPad y, finalmente, a **Apple TV**, aunque este último no puede ejecutar las aplicaciones iOS creadas para el ecosistema iPhone y iPad.

Al momento de escribir esta publicación, iOS alcanza la versión 7.1, la cual funciona en dispositivos iPhone 4S, iPad 2 o superior y Apple

LA TECNOLOGÍA WAP DÍA A DÍA

La primera llegada de internet a los teléfonos móviles fue a través de la tecnología WAP, que utilizaba la frecuencia portadora de señal telefónica para enviar y recibir datos. Los navegadores web mostraban páginas similares (texto e imágenes) a las utilizadas en la década del 90.

TV 2.0 o superior. La firma de Cupertino ha dejado de dar soporte para el sistema operativo iOS 4.0 o anterior y promueve en su sitio web la adaptación de apps para iPad, iPod Touch y iPhone a la última versión disponible.

Apple es desarrolladora tanto del hardware como del sistema operativo de todos sus dispositivos móviles.

Android

El sistema operativo Android, actualmente propiedad de **Google Inc.**, tuvo sus orígenes como un sistema operativo móvil independiente, propiedad de **Android Inc.**. En base al interés de Google por el mercado de dispositivos móviles, la firma vio que el mundo sucumbió a la propuesta de Apple cuando se rumoreaba, en 2006, sobre el posible lanzamiento de iPhone, y decidió adquirir un sistema operativo avanzado en cuanto a prestaciones, para poder entrar de lleno al mercado mobile, con una herramienta sólida.

> CUANDO GOOGLE COMPRÓ ANDROID, COMENZÓ A ADAPTAR EL SISTEMA A SUS NECESIDADES

Así fue como Google compró la firma Android Inc. y comenzó a adaptar este sistema operativo en base a sus necesidades. Android, al momento de escribir esta publicación, se encuentra en su versión 4.4 (**Kit Kat**). La primera versión (1.5) fue lanzada en 2008, y, a partir de la versión 2.1, Android comenzó a ganar popularidad. Sus versiones 2.2 y 2.3 fueron adaptadas –gracias a que se sigue manteniendo como código **open source**– a tablets y netbooks sin soporte oficial de Google.

Al ver que el mercado prometía más que un teléfono inteligente, Google decidió lanzar **Android 3.0**, específicamente diseñado para tablets. Luego de su actualización a la versión 3.1 (**Honeycomb**), Android saltó directamente a la versión 4.0, unificando su núcleo para smartphones y tablets, de manera inteligente. Es por ello que, hoy, cualquier aplicación desarrollada para esta plataforma puede instalarse tanto en teléfonos móviles como en tablets.

Gracias a su versión open source, Android consiguió muchos clones alternativos, adaptados en su mayoría por empresas asiáticas, que brindan el mismo servicio que la versión promovida por Google,

aunque sin el soporte oficial de esta empresa (motivo por el cual deben tener su mercado independiente de aplicaciones).

Dentro de las versiones alternativas de Android, podemos destacar la utilizada por la firma **XIAOMI**, muy popular en China, con un mercado de más de 40 millones de dispositivos vendidos, y la adaptación **Fire OS**, bautizada así por **Amazon**, quien tomó el código fuente de Android y creó sus propias tablets e **e-readers**, como así también su propio ecosistema de aplicaciones.

Figura 6. iOS 7.1 y **Android 4.4**, los actuales sistemas operativos móviles que reinan el mercado de smartphones y tablets.

Windows Phone

Microsoft también irrumpió, en 2010, en el mercado de la telefonía móvil con su sistema operativo **Windows Phone**. Este fue una reingeniería completa del viejo sistema operativo **Windows Mobile**, que no había logrado interés en la población y que Microsoft también había dejado abandonado en su momento.

Acorralada por el creciente uso de dispositivos móviles, la firma de Redmond decidió ingresar en este terreno desde cero, con un nuevo desarrollo. Windows Phone, al momento de escribir esta publicación, se encuentra en su versión 8.1, aunque en el mercado se encuentran muchos dispositivos de versión 7.x activos.

BlackBerry

RIM se rebautizó en 2013 como **BlackBerry** para unificar la imagen de la compañía con la de sus productos estrella. BlackBerry actualmente se encuentra en la versión 10.2.1. A partir de la versión 10.0, realizó una reingeniería completa de su plataforma móvil, al tomar el sistema operativo **QNX** como base.

Este sistema operativo garantiza, desde hace más de una década, un correcto funcionamiento multitarea en tiempo real. A nivel mundial, BlackBerry perdió gran parte de su mercado por el poco interés en actualizar a tiempo su sistema operativo.

> BLACKBERRY PERDIÓ MERCADO POR NO HABER ACTUALIZADO A TIEMPO SU SISTEMA OPERATIVO

Aún en el mercado se pueden encontrar equipos BlackBerry corriendo la versión 7.x de su sistema operativo. BlackBerry, al igual que Apple, es desarrolladora del hardware y sistema operativo de todas sus plataformas móviles.

Figura 7. Windows Phone 8.1 y **BlackBerry 10** ocupan el tercero y cuarto puesto en el mercado de dispositivos móviles.

Otros sistemas operativos

Como bien dijimos, el mercado principal se centra hoy dentro de estas plataformas móviles, las cuales se disputan en todo momento su hegemonía en el terreno mobile. Pero, aun así, hay otros jugadores que tuvieron su momento de gloria y que todavía hoy siguen conservando un nicho del mercado, o bien buscan ganárselo con el tiempo. Veamos un detalle de cada uno a continuación.

Firefox OS

La **Fundación Mozilla**, dueña de la plataforma open source liderada por **Firefox Browser** y **Thunderbird**, entre otras grandes aplicaciones, decidió incursionar en 2012 en el terreno de un sistema operativo móvil. El impulso que llevó a Mozilla a este terreno fue el creciente interés por parte de los usuarios en la navegación web y en el uso de aplicaciones que se ejecutan dentro de un browser. Teniendo el respaldo de un gran mercado de usuarios y la popularidad lograda por HTML5, CSS3 y JavaScript como base fundamental de tecnología en el mundo web, Mozilla, con el apoyo de **Telefónica de España**, decidió crear un sistema operativo móvil que tomara como filosofía las aplicaciones web basadas en los tres pilares fundamentales de la Web de hoy.

Así nació **Firefox OS**, apuntado en un principio para mercados emergentes. Firefox OS se integra con teléfonos inteligentes de baja y media gama, como un smartphone que funciona basado en HTML5, CSS y JavaScript. Todas las aplicaciones nativas que se instalen en el dispositivo deberán ser desarrolladas con las tres tecnologías pilares de la Web de hoy, y no bajo frameworks propietarios como **XCode**, **Android Studio**, **Visual Studio** o **QNX Momentics IDE**, como así lo requieren sus competidores principales.

La empresa canadiense creadora de BlackBerry aún sigue dominando muchos mercados, principalmente en Latinoamérica, donde su ahora anticuada gama de equipos móviles, con versiones de sistema operativo 6.x y 7.x, todavía sigue deslumbrando a la gente con la comodidad de un teclado QWERTY.

Tizen

Este sistema operativo móvil nace de un proyecto en conjunto de la **Fundación Linux**, **Samsung** e **Intel**, tomando como base el sistema operativo **MeeGo**. Este último fue importante en el ecosistema **Nokia**, que lo dejó abandonado para promover **Symbian** y la plataforma **S40**, mucho antes de que Microsoft adquiriera la firma finlandesa.

Tizen basa su arquitectura principal en HTML5 y otros estándares web y fue pensado no solo para el mundo de los smartphones, sino también para tablets, netbooks, **smart TVs** y otros sistemas de entretenimiento vehiculares.

Actualmente, Tizen se encuentra en su versión 2.2, dando retro-compatibilidad con las aplicaciones MeeGo a través de la API basada en HTML5.

Figura 8. Bajo un look que los empareja bastante, **Firefox OS** y **Tizen OS** buscan ser los protagonistas en la próxima revolución de sistemas operativos móviles.

Ubuntu Phone

Canonical, la firma creadora de **Ubuntu Linux**, el sistema operativo Linux más popular de la última década, también decidió

comenzar a explorar el mundo de las pantallas táctiles. Así fue como desarrolló el sistema operativo **Ubuntu Touch**, que permitió integrar Linux con las pantallas táctiles. Posteriormente, la firma se volcó al desarrollo de un sistema operativo móvil, al cual bautizaron como **Ubuntu Phone**.

Al momento de escribir esta obra, Ubuntu Phone no ha sido lanzado al mercado. Se estima que, para fines de 2014 o inicios de 2015, este sistema operativo comenzará a ser distribuido en diferentes modelos de equipos.

Sí existe una versión instalable de este sistema operativo en algunos terminales Android más populares, como ser **Samsung Galaxy 3** y **4**, o **Nexus 4** y **5**, entre otros. Esto les permite a los desarrolladores explorar el ecosistema Ubuntu Phone y comenzar a diseñar apps para esta plataforma.

FIRE OS CORRE EN DISPOSITIVOS KINDLE Y ESTÁ BASADO EN EL CÓDIGO FUENTE DE ANDROID

Fire OS

Este sistema operativo es una adaptación del código fuente de Android, implementado por la firma Amazon. **Fire OS** corre desde hace algunos años en la mayoría de los dispositivos **Kindle e-readers** y tablets **Amazon Kindle Fire**.

Al estar basado en el código fuente de Android, Fire OS permite crear aplicaciones nativas con las mismas herramientas de desarrollo que se utilizan para Android. Es más, de manera periódica, Fire OS lanza actualizaciones que incluyen las novedades del sistema operativo Android que fueron liberadas para su versión open source.

LOS QUE SE BAJARON DEL CAMINO

Nokia decidió en 2012 retirar del mercado a Symbian y MeeGo para darle total paso a Windows Phone. Sin embargo, ambos sistemas operativos se caracterizaron por dar el puntapié inicial a estas nuevas generaciones que lideran el mercado. Menús basados en iconos, bien estructurados y ágiles en cuanto a manejo táctil o mediante flechas de navegación.

Amazon, por el momento, solo comercializa equipos e-reader o tablets con este sistema operativo. Al momento de escribir esta publicación, Fire OS no está integrado en ningún smartphone.

Figura 9. Ubuntu Phone y **Fire OS**. Este último no pertenece a la gama smartphone, pero se rumorea desde 2012 que **Amazon** ingresará pronto en este mercado.

Nokia

Nokia todavía mantiene su línea de equipos **S40**, basados en el sistema operativo homónimo, bajo la línea **Asha**. **Symbian** fue otro sistema operativo lanzado por Nokia que aún mantenía su ecosistema de aplicaciones propietarias. Pero fue determinada su discontinuidad al momento del anuncio oficial de la compra de esta firma por parte de Microsoft.

Casi todos los sistemas mencionados poseen las mismas características: integran apps propietarias descargables desde sus respectivas tiendas de aplicaciones, poseen GPS, acelerómetro, navegadores web con soporte para HTML5, conectividad a internet mediante 3G y 4G, y utilización de Wi-Fi para el mismo propósito.

Aplicaciones móviles

Si bien el hardware y la variedad de sistemas operativos son las estrellas más importantes del momento en el ecosistema móvil, no podemos pasar por alto que todos estos pioneros no serían nada si no existieran las tiendas de aplicaciones.

Estas tiendas poseen cientos de miles de apps pagas o gratuitas que permiten interactuar de diversas maneras a través del dispositivo móvil, utilizando las bondades del hardware de cada equipo, al que solo usan para ingresar información importante para el usuario, o simplemente para jugar los videojuegos de moda, entre otras alternativas más.

> EL MERCADO DE LAS APLICACIONES NO SE LIMITA SOLO A APLICACIONES INSTALABLES

Pero el mercado de aplicaciones móviles no se limita solo a aplicaciones instalables. Muchas de estas nacieron originalmente como un sitio web para computadoras de escritorio y móviles, y con el tiempo se transformaron en aplicaciones nativas instalables. Otras nacieron como una aplicación web y aún hoy se siguen manteniendo como tales, consumiendo algunas bondades de los browsers móviles que facilitan su uso.

Las apps móviles pueden interactuar con el usuario haciendo uso del espacio de almacenamiento del dispositivo o integrándose con parte del hardware. Podemos mencionar, como ejemplos: una app móvil que nos ayude a desplazarnos por una ciudad y que utilice la función de geolocalización del equipo en conjunto con los mapas que posee el dispositivo; o una app que oficie de agenda telefónica de profesionales específicos para poder comunicarnos con ellos, ya sea mediante un llamado telefónico, un e-mail o un mensaje de texto.

Otro uso muy común es el de las aplicaciones como **Instagram** o las basadas en redes sociales (**Google Plus**, **Facebook** o **Twitter**), que nos facilitan la comunicación con nuestros conocidos a través de publicaciones, o bien compartiendo una foto o video tomada desde la misma app que hace uso de la cámara de los teléfonos.

Como vemos, la integración de las apps con el hardware de los equipos es casi infinita. El único límite aquí establecido es nuestra imaginación como desarrolladores.

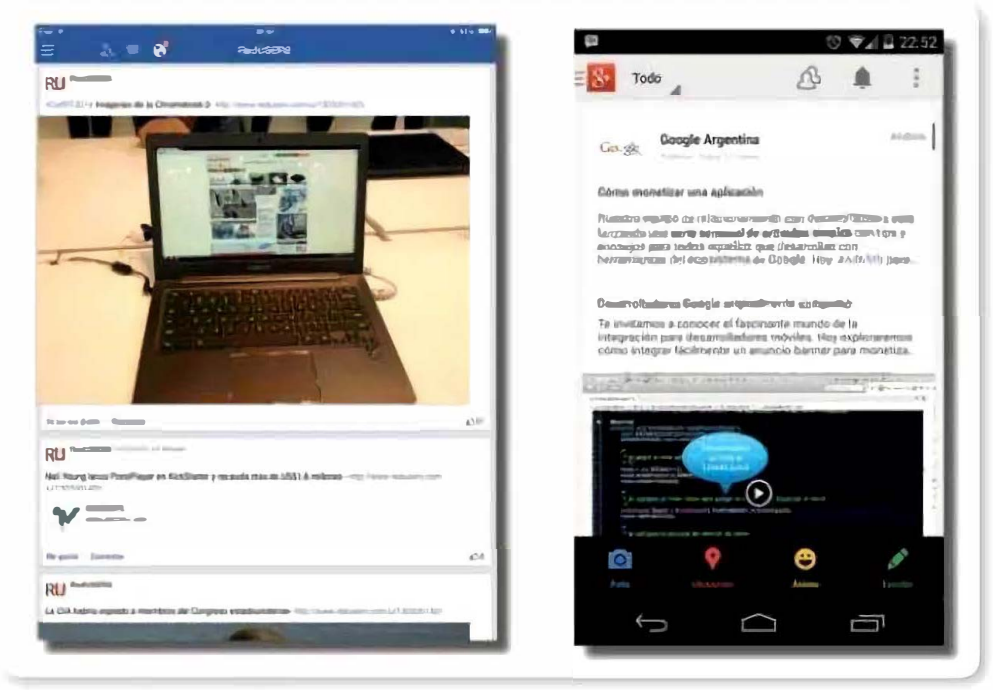


Figura 10. Las **redes sociales** fueron las primeras aplicaciones móviles en llegar a los teléfonos inteligentes.

WebApps

WebApp proviene de la conjunción de las palabras en inglés **Web Application** (aplicación web). Este tipo de aplicaciones son accedidas mediante la Web o una red Intranet. Para acceder a ellas, el requisito esencial es contar con un navegador web que permita ejecutarlas.

De este modo, una WebApp puede categorizarse como un programa informático, con la diferencia de que se ejecuta desde un browser web.

Su estructura está conformada mayormente por: HTML, CSS, JavaScript, y/o algún otro lenguaje de programación que trabaje del lado del servidor (PHP, ASP.Net, Python, Ruby, CGI, Perl, etcétera).

¿UNO, DOS O MÁS DESARROLLOS?

Una buena práctica en el mundo del desarrollo para escritorio y móviles se da de la mano de **Responsive Web Design**. Se trata de un mecanismo que permite crear un solo sitio web que se muestra adaptado a la plataforma desde donde accedemos. Aun así, no siempre es conveniente aplicar esta técnica, debido a una cuestión de performance.

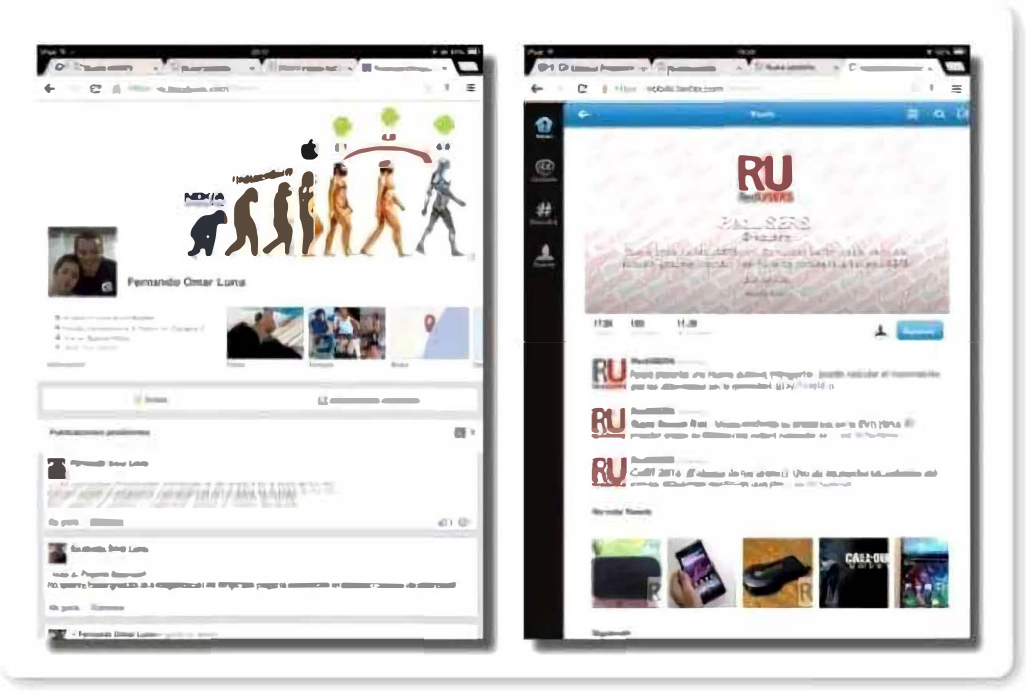


Figura 11. Muchas WebApps han lanzado una versión específica para dispositivos móviles, aun teniendo una aplicación nativa para ellos.

Apps nativas

Una **app nativa** es aquella aplicación que fue programada para ser instalada dentro de un sistema operativo determinado. Para el caso de las aplicaciones móviles, una app nativa es aquella que fue programada bajo el lenguaje o framework recomendado por el fabricante del sistema operativo.

Por ejemplo, para el caso de iOS, se utiliza **Objective C** bajo el framework **XCODE**; para Windows Phone es utilizado el lenguaje **C#** o **Visual Basic** bajo el framework **Visual Studio**; en la plataforma Android se utiliza el lenguaje **JAVA** bajo diversos **IDEs**; y, para la plataforma BlackBerry OS, se utiliza **QNX Momentics IDE**.

Estos frameworks permiten acceder mediante las **API** (*Application Programming Interface*) a prácticamente todas las características del hardware y del sistema operativo móvil.

Apps híbridas

Podemos definir una aplicación híbrida como una aplicación web desarrollada con los estándares HTML, CSS y JavaScript, entre otros, la

cual es empaquetada bajo un conjunto de reglas y parámetros que permite instalarla en un dispositivo como cualquier aplicación nativa.

Luego, al ser ejecutada, la aplicación híbrida utilizará el motor del navegador web, ocultando su menú, su barra de direcciones y de herramientas para poder simular que la WebApp es una aplicación nativa. Un claro ejemplo de app híbrida es el uso de **PhoneGap** como framework que intercede entre el software –desarrollado bajo estándares HTML5, CSS y JavaScript– con una librería JavaScript que agrupa funciones específicas para acceder al hardware de los dispositivos móviles.

> UNA APLICACIÓN HÍBRIDA SIMULA SER UNA APLICACIÓN NATIVA, EN LUGAR DE UNA WEBAPP

PhoneGap ejecuta una sesión reducida del navegador web propio de la plataforma para simular que la app en ejecución es nativa y no una WebApp. Esto es imperceptible para el usuario final, ya que una app desarrollada cumpliendo ciertos estándares, como los que brinda PhoneGap, no presentará ninguna diferencia al usuario final respecto de una app nativa.

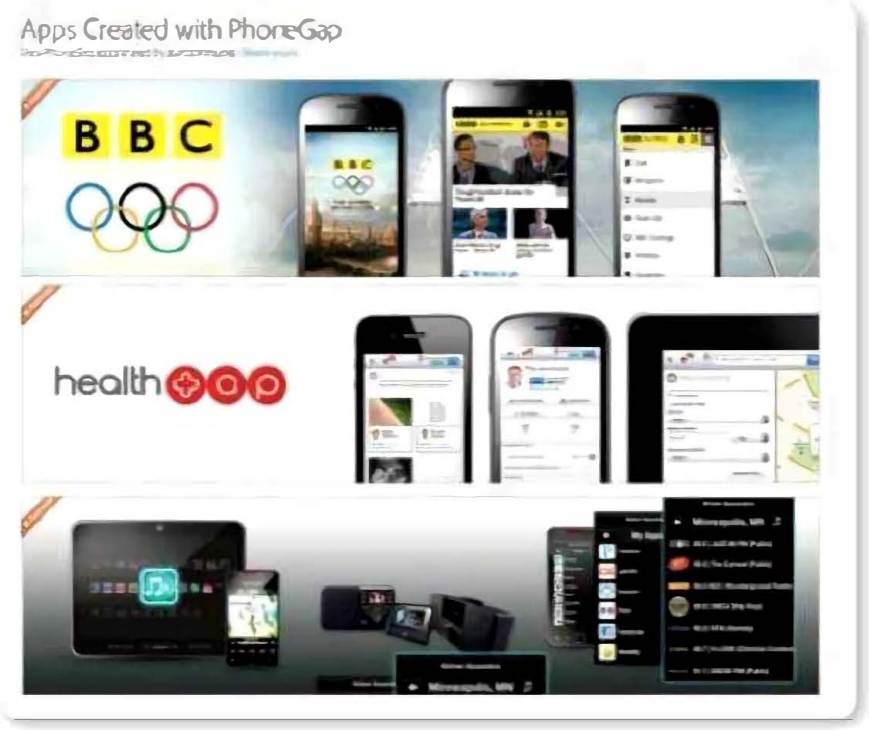


Figura 12. Algunas de las aplicaciones que utilizaron la tecnología **PhoneGap** para crear apps híbridas multiplataforma.

Ventajas y desventajas entre plataformas

La ventaja que presenta una WebApp consiste en unificar, en un único desarrollo englobado bajo un web browser, una app que cumpla con determinadas funcionalidades. Esta es desarrollada una sola vez y puede ser ejecutada indistintamente en un smartphone, tablet o computadora de escritorio.

> UNA APLICACIÓN HÍBRIDA PUEDE SER LLEVADA A VARIAS PLATAFORMAS CON UN ÚNICO DESARROLLO

Como desventaja, podemos destacar que las WebApps utilizan de manera limitada el hardware de los equipos donde se ejecutan. Y, en muchos casos, el acceso al hardware o sistema operativo que funciona bajo una plataforma no siempre funciona en todas las otras.

Las aplicaciones nativas permiten alcanzar un desarrollo veloz en cuanto a respuesta y un acceso a casi todo el hardware de los dispositivos. Como desventaja, podemos destacar que, para llevar una app a las plataformas nativas populares, debemos al menos aprender unos cuatro o cinco lenguajes de programación diferentes. Esto presentará una curva de aprendizaje muy grande, que insume muchas horas, y casi ninguna parte del código local utilizado en un desarrollo podrá aprovecharse en otro.

Las aplicaciones híbridas nos dan la ventaja de utilizar un framework como PhoneGap donde podemos desarrollar una única app que será llevada a, por lo menos, cuatro o cinco plataformas distintas con un único esfuerzo de desarrollo. Como desventaja, no siempre se puede acceder al hardware de manera completa y por igual desde un único desarrollo; por esto, dependiendo la complejidad requerida, puede que entre una plataforma y otra se pierdan características. Además, las apps híbridas presentan tiempos de ejecución inicial más lentos que los de una app nativa.

LOS ÚLTIMOS CAMBIOS EN FACEBOOK

Embrollos de código y compatibilidad HTML5 llevaron a Facebook a dejar de lado las aplicaciones híbridas de la versión móvil de su sitio web, optando por desarrollar desde cero una aplicación nativa para cada una de las plataformas móviles existentes. Los creadores de **Sencha Touch** demostraron, a través de **Fastbook**, que Facebook no estuvo tan acertado en el cambio.

Tecnologías de la Web actual

A lo largo de esta última década, la tecnología que reina la Web cambió drásticamente. A principios de 2000, cuando la banda ancha comenzó a masificarse mundialmente, la tecnología Flash reinó en casi toda la Web gracias a que su interfaz gráfica podía llevar la imaginación de un desarrollador o diseñador web hacia cualquier rincón.

A partir de 2007, cuando Apple lanzó su iPhone y, en su comunicado oficial, anunció que habían decidido dejar la tecnología Flash fuera de iOS, esta tecnología comenzó a decaer a nivel mundial. Como principales problemas, podemos mencionar que las baterías de los dispositivos móviles no son muy duraderas y las conexiones a internet eran lentas (y lo siguen siendo en muchos países), por lo que Flash causaría una demora importante en cargar una web en iOS y consumiría su batería. Un último motivo: los baches de seguridad que siempre tuvo esta tecnología podrían causar serios problemas a Apple.

Android, por su parte, apostó a Flash, que tuvo soporte hasta la versión 3.x de este sistema operativo. A partir de la versión 4.0, Adobe decidió discontinuar el desarrollo de Flash para dispositivos móviles, y Google decidió dejar afuera el soporte Flash en Android 4.0.

Por todo esto, HTML5, CSS3, y JavaScript pasaron a primera plana y comenzaron a reinar fuertemente en la Web. Repasemos, a continuación, estas tres características importantes que hacen a la Web de hoy.

Figura 13. El poder de HTML5, CSS3 y JavaScript llevó los videojuegos a la pantalla de los navegadores web, llenando el vacío que había dejado **Adobe Flash**.

HTML5

El cambio realizado en los browsers fue, durante estos últimos años, significativo, y la mayor parte de este se debió a HTML5. Pero, aun sabiendo esto, cabe que nos preguntemos... ¿qué es, exactamente, HTML5?

Podemos definirlo como una colección de estándares que influyen en el diseño y en el desarrollo de una página web. Anterior a HTML5, el estándar que marcaba la tendencia del mundo de los navegadores era **HTML 4.1**. Estos estándares son supervisados por el **W3C** (*World Wide Web Consortium*).

> HTML5 ES UNA COLECCIÓN DE ESTÁNDARES DE DISEÑO Y DESARROLLO DE UN SITIO WEB

En la creación de este nuevo conjunto de estándares, HTML5 decidió saltar las bases de HTML 4.1 y aplicar un rediseño propio de cómo se debe estructurar un sitio web. Esta decisión definió un conjunto de etiquetas predeterminadas que permiten estructurar una página web de manera rápida, clara y concisa.

Poco a poco, los navegadores web fueron adaptando su motor de renderización para incluir el soporte necesario para HTML5. **Google Chrome** fue uno de los navegadores que pudo apreciar tempranamente este potencial y lo incluyó desde las primeras versiones del browser.

Hoy, la mayoría de los browsers soportan un gran número de etiquetas que conforman el estándar HTML5, pero también encontraremos diferencias entre plataformas, como, por ejemplo, el tipo de formato de reproducción de los tags **<audio>** y **<video>**: dependiendo del motor utilizado por el navegador web, podrá o no reproducirse un códec determinado.

PROBLEMAS EN EL CAMINO

En el diseño correcto de una WebApp debe tenerse en cuenta un factor sumamente importante: el **código muerto**. Muchos sitios conocidos, como **Amazon.com**, presentaron problemas de performance y un gran consumo de batería en móviles por tener mucho código JavaScript en desuso, pero por donde el navegador web debía pasar sí o sí.

La estructura de una página web también fue reemplazada casi por completo con HTML5. Actualmente, crear una página web simple se puede resumir en tan solo un par de tags que, gracias a su nombre, definen bien, en el código HTML, cuál sección corresponde a la página.

```
<head>
  Título de la página web
</head>
<body>
  Aquí incluimos la descripción que queremos que tenga nuestra página web.
</body>
<footer>
  Copyright 2012 - Fernando Luna
</footer>
```

Una simple lectura de este código permite a cualquier persona,, con mínimos conocimientos en el tema, conocer la manera simple que tiene HTML5 de estructurar una página web.

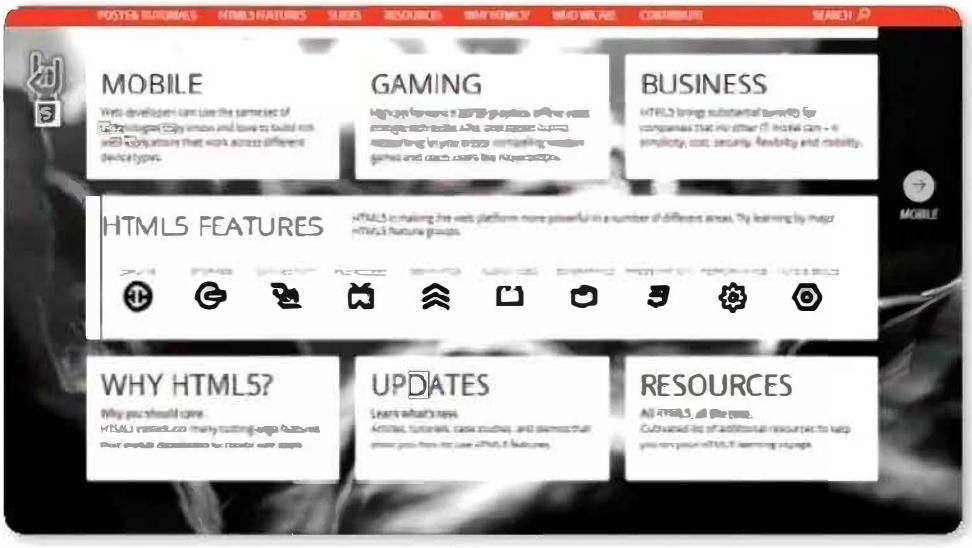


Figura 14. Un sitio web que funciona como fuente de información de la tecnología HTML5 es **www.html5rocks.com**.

CSS

Las siglas **CSS** provienen del inglés *Cascade Style Sheets* (**Hojas de estilo en cascada**). Se utiliza para definir la presentación de un

documento HTML o **XML** estructurado. Actualmente, W3C se ocupa de cuidar y formular la especificación final de las hojas de estilo, para que dicho estándar quede definido como tal y sea implementado luego en los navegadores web.

CSS permite separar la estructura de un documento web de su presentación en pantalla. El atributo **<Style>** permite definir un estilo específico a cada etiqueta que se utiliza dentro de la construcción de una página web. Mediante CSS, entonces, podemos definir el formato, color, fuente, tamaño, estilo, color de fondo, bordes, subrayado, ubicación en pantalla y otros elementos más que componen un texto, imagen o cualquier otro medio utilizado dentro de una página web.

CSS3, su última versión estable, permite aplicar reglas similares a la condición **IF** de cualquier lenguaje de desarrollo, para determinar qué ancho de pantalla estamos utilizando y, en base a esto, aplicar un estilo u otro a las etiquetas de cada página HTML y mostrar u ocultar secciones.

Figura 15. **www.cssbasics.com** nos pone a disposición un libro online con todas las características de CSS (desde cero, hasta transformarnos en ninjas).

JavaScript

JavaScript es un lenguaje de programación interpretado y definido como orientado a objetos. Su principal utilización es del lado cliente, mediante un **SDK** que debe disponer el dispositivo previamente instalado. Su mayor uso se da dentro de los navegadores web.

Mediante este lenguaje de programación podemos realizar un sinfín de cosas sobre los objetos que una página web puede tener. Actualmente todos los navegadores modernos interpretan sin problema el código JavaScript que se integra dentro de las páginas web.

La mayoría de los dispositivos móviles tienen instalado JavaScript en su sistema operativo, lo que permite a cualquier programador web aprovechar muchas funcionalidades resumidas dentro del lenguaje JavaScript, para acceder a ciertas características del dispositivo, potenciando, así, el funcionamiento de una página web.

Figura 16. **www.w3schools.com** es un buen punto de partida para aprender JavaScript desde cero.

Librerías y frameworks móviles

Para comprender mejor los términos del desarrollo web, debemos separar los términos **librería** y **framework**. Una librería es un

La última versión conocida de CSS es la versión 3, en la que se incorporaron ventajas como animación, diseño gráfico y estilos específicos para la estética web, que hicieron que CSS, junto con JavaScript, se transformara en el verdugo del casi olvidado Adobe Flash, pionero de la década del 2000 en el diseño web de sitios animados.

LAS LIBRERÍAS MÁS CONOCIDAS SON JQUERY MOBILE, SENCHA TOUCH Y JQUERY UI

conjunto de tecnologías que puede englobar características de CSS y JavaScript y que nos facilita, de alguna manera, el desarrollo de una solución web para ambas plataformas. Dentro de las librerías más conocidas, podemos mencionar a **jQuery Mobile**, **Sencha Touch** y **jQuery UI**, entre otras.

Estas se ocupan de compactar funcionalidades que requieren mucho tiempo de elaboración por parte de un desarrollador en funciones específicas que nos aportan agilidad al momento de desarrollar y nos permiten despreocuparnos por la estética o solución cuando el proyecto se deba ejecutar en diferentes plataformas.

Por otro lado, el framework nos permite englobar, en un único entorno, todo el conjunto de archivos y APIs que nos permiten desarrollar una solución, estructurados de una manera jerárquica.

Dentro de los frameworks más conocidos, podemos mencionar a **Eclipse** y **Netbeans** como los más populares y gratuitos, y a **Dreamweaver** dentro de los frameworks pagos. Todos estos prestan características similares, aunque con algunos toques personales, y permiten incorporar las librerías mencionadas anteriormente sin mayores problemas.

Figura 17. **jQuery Mobile** y **Sencha Touch** son las dos librerías más utilizadas para realizar WebApps y apps nativas.

Entornos de desarrollo

Los **entornos de desarrollo** (**IDE**, por sus siglas en inglés) permiten incorporar funcionalidades extra de manera nativa para ejecutar los desarrollos en un ambiente de **test**, **simulador** y hasta para compilar los proyectos realizados de manera nativa.

Dentro de los entornos de desarrollo más comunes encontramos a Dreamweaver, Visual Studio, **QNX Momentics IDE**, **Titanium Studio**, entre otros. Los frameworks también permiten realizar compilaciones de proyectos que se desarrollan de manera nativa, pero generalmente requieren de un plugin que permita llevar a cabo esta acción.

Dreamweaver

El IDE elegido para llevar adelante los desarrollos será **Adobe Dreamweaver**. Este fue elegido debido a que, desde el año 2010, integra el desarrollo web mobile de manera nativa. En 2012, la firma Adobe adquirió la mencionada empresa PhoneGap, que permite convertir una web a app para las plataformas móviles más populares, y con esto integró de manera transparente el uso de PhoneGap dentro del IDE Dreamweaver.

Figura 18. Desde **www.adobe.com/products/dreamweaver** podemos descargar la versión de prueba de este popular IDE.

Otra gran ventaja de Dreamweaver es que incluye un entorno **WYSIWYG**, que nos permite ver, en tiempo real, el resultado del código que escribimos, y también testear la funcionalidad del resultado de nuestros desarrollos sin recurrir a **web server** externos.

DREAMWEAVER PERMITE VER EN TIEMPO REAL EL RESULTADO DEL CÓDIGO ESCRITO

Si bien Dreamweaver es pago, se puede descargar una versión de prueba por 30 días, con suscripción previa, en su página oficial: **www.adobe.com/la/products/dreamweaver.html**. La última versión de Dreamweaver al momento de escribir esta obra es Dreamweaver CC. Pero si disponemos de la versión 5.0, 5.5 o 6.0, también podremos llevar a cabo estos desarrollos, y, para la integración con PhoneGap, desde la versión 5.5.

Dado que Dreamweaver funciona solo en las plataformas Mac y Windows, quienes utilicen Linux o alguna otra variante de Unix pueden recurrir a Eclipse o Netbeans como framework de desarrollo.

Estas dos plataformas tienen versiones para todos los sistemas operativos, y Netbeans, a partir de la versión 7.3, integra un web server nativo que permite ejecutar las pruebas en la computadora donde desarrollemos.

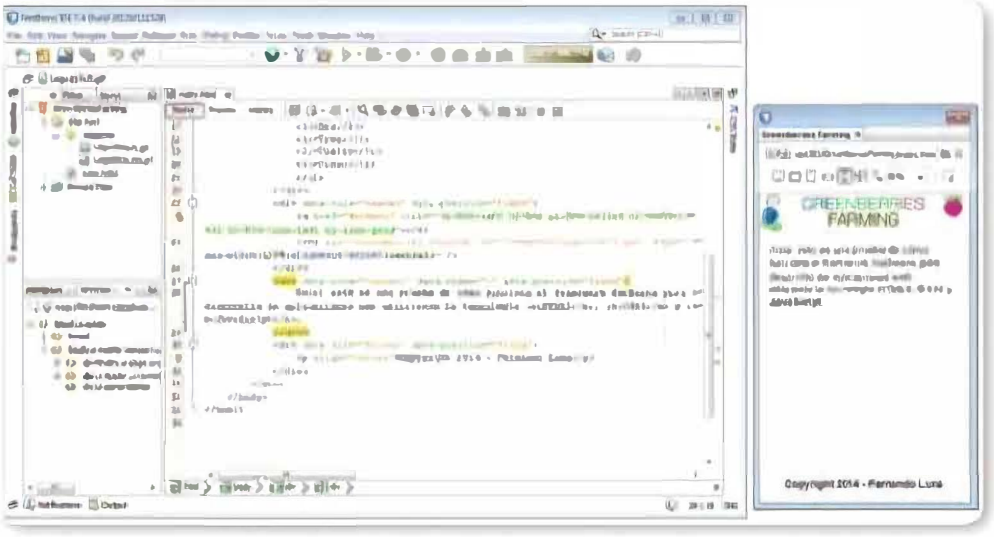


Figura 19. Como alternativa a Dreamweaver, **Netbeans** es una excelente opción, ya que su última versión integra un visor WYSIWYG y un web server nativo.

Y si disponemos de una red Wi-Fi y dispositivos móviles para testear las WebApps, podremos acceder sin problema desde el web browser mobile al web server que Netbeans ejecuta. Puede descargarse desde el siguiente link: **www.netbeans.org/downloads/start.html?platform=windows&lang=en&option=all**.

Si, por el contrario, se desea utilizar Eclipse, este podrá descargarse desde su sitio web oficial: **www.eclipse.org**.

RESUMEN

Este capítulo nos introdujo a las diferentes plataformas móviles y vimos cómo interactúa la nueva Web sobre estos novedosos dispositivos. Conocimos parte de la historia de las plataformas más importantes y del nacimiento de la tecnología HTML5 que, junto con CSS3 y JavaScript, impone las reglas de la nueva Web. También repasamos las diferencias entre WebApp, app nativa y app híbrida, y los entornos de desarrollo y librerías que nos permitirán llevar a cabo nuestro camino hacia la creación de aplicaciones móviles para la mayoría de los dispositivos actuales.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es una **WebApp**?
2. Mencione algunas ventajas que brinda una WebApp.
3. ¿Qué es una **aplicación nativa**?
4. Mencione dos características de las aplicaciones nativas que las coloquen en desventaja con respecto a las WebApps.
5. ¿Qué es una **librería**?
6. ¿**Adobe Flash** consume menos recursos que los que **HTML5** y **CSS** exigen en un browser?
7. ¿Los dispositivos móviles de Apple poseen soporte de Adobe Flash Player?
8. Nombre al menos cuatro plataformas actuales del mundo móvil.
9. ¿Qué es **JavaScript**?
10. Mencione al menos tres **frameworks** de desarrollo móvil.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*02

HTML5

HTML5 es el lenguaje de marcado que domina la Web de hoy y que simplificó las funciones utilizadas en los navegadores, tanto de escritorio como de los dispositivos móviles.
En este capítulo, haremos un repaso por algunos conceptos importantes de la base principal de HTML5 que nos serán de suma utilidad para el resto del libro.

El lenguaje HTML5

HTML5 fue creado con el propósito de brindar soporte al contenido existente en la Web basándose en los estándares impuestos con **HTML 4.x**, pero unificando la estética y funcionalidad de los sitios en todos los navegadores web, lo que se conoce como compatibilidad **cross-browsers**.

Su principal objetivo

Del estándar propuesto por **HTML5** podemos hacer alusión a los nuevos elementos dentro de la sintaxis que difieren en gran parte de la versión anterior de este lenguaje de marcado. HTML5 es utilizado por los diseñadores que crean sitios web y no hay ninguna problemática cuando se requiere combinarlo con una versión anterior.

Navegadores y sistemas operativos

HTML5 propone herramientas avanzadas y mejores experiencias para el usuario final garantizando el correcto funcionamiento de casi todas sus etiquetas en la mayor cantidad de navegadores web.

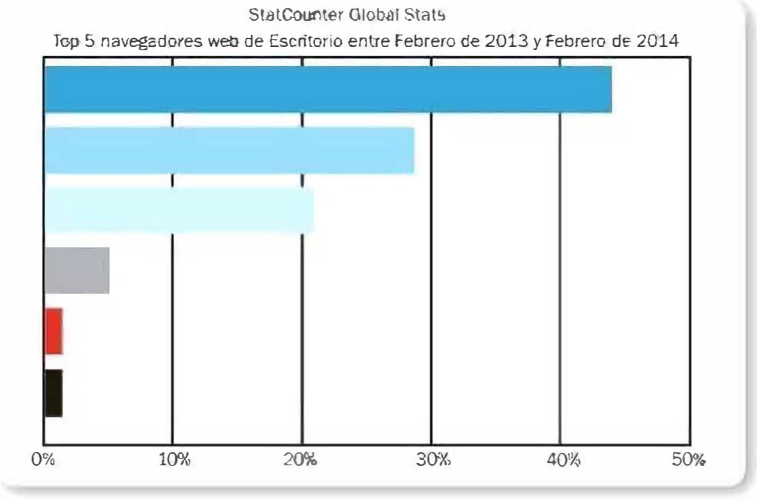


Figura 1. Hasta febrero de 2014, **Google Chrome** es el browser más utilizado, seguido por **Internet Explorer**, **Firefox** y, más lejos, **Opera**.

Hablar de navegadores implica pensar en Chrome, Firefox, Internet Explorer, Opera, Camino, Safari, Android Browser, Galeon, SlimBoat, Apollo, Dolphin, iCab, OmniWeb, Voyager, SpaceTime, QupZilla,

NetSurf, Incognito, Epiphany, myBrowser, RockMelt, y otros tantos más que harían la lista casi interminable.

Tenemos una opción inmensa de navegadores web circulando por internet, y esto se debe a la variedad de plataformas como Windows, Linux, OS-X, iOS, Android, Windows Phone, Nokia, BlackBerry, entre otras tantas. Por suerte, desde hace un tiempo, los desarrolladores comprendieron que lo más importante de un navegador pasa por el **motor de render**, y así fue como decidieron adaptar sus nuevas versiones con los motores de renderizado más populares de la Web.

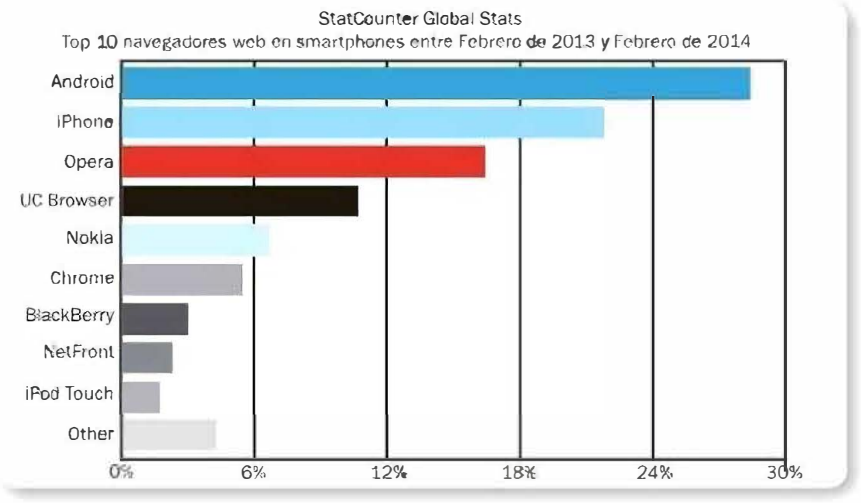


Figura 2. Por la segmentación de **Android** (versión 2.1 a 4.4), **Android browser** es el navegador más utilizado, dejando a **Chrome** relegado al sexto puesto.

Hasta el momento de escribir esta publicación, Android seguía reinando en el mercado de smartphones a nivel mundial, con una porción de más del 80 %. Sin embargo, a pesar de los grandes esfuerzos realizados por Google y de la cantidad de firmas que integran el sistema operativo Android en sus tablets, este sucumbe en un terreno que está siendo ampliamente liderado por iPad.

EL PASADO DE YOUTUBE

YouTube dependió por años, en la reproducción de sus videos, del plugin **Flash Player**. Todas las películas eran convertidas, en el proceso de upload, al formato de video **.FLV**. HTML5 trajo vientos de cambio a este portal, ya que ahora no es necesario depender más de este plugin.

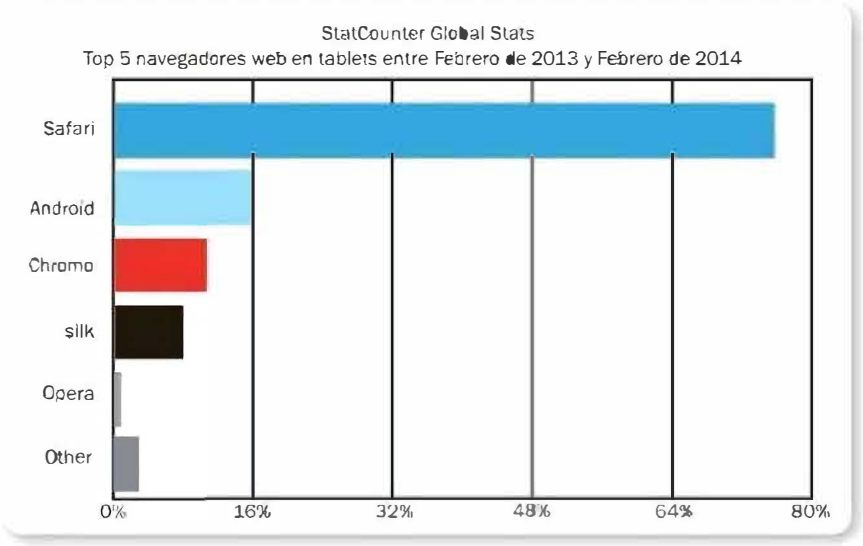


Figura 3. Si bien Android ha conquistado el mercado de los smartphones, aún no se ha podido quedar con el de las tablets.

Motores de renderizado

Entendemos por **motor de renderizado** al software que interpreta el contenido web (basado en HTML, XML, gráficos, CSS3 y JavaScript) y lo traduce a lenguaje gráfico, pintando en pantalla la estética del sitio web o correo electrónico, para que el usuario lo visualice o imprima.

A pesar de que la segmentación se está reduciendo, aún existe una variedad importante de motores de renderizado. Entre ellos están: Gecko, Trident, WebKit, KHTML, Presto, Tasman, gzilla, GtkHTML, Servo y Blink. Los tres primeros son los más utilizados por los browsers más comunes de cada plataforma, mientras que **Blink** es el nuevo motor desarrollado para Google Chrome que reemplaza a **WebKit**, y **Servo** es el nuevo motor desarrollado por Mozilla, que reemplazará a **Gecko** en las plataformas con arquitectura **ARM**.

Un HTML para dominarlos a todos

Hasta ahora sabemos que los sistemas de escritorio están dominados por Windows; los smartphones, por Android; y las tablets, por iPad, aunque en estos dos últimos casos los competidores y ofertas de sistemas operativos abundan. Por el lado de los browsers, debemos hablar de motores de render, ya que estos son los que interpretan al lenguaje de marcado, procesando correctamente las nuevas etiquetas.

La mayoría de los browsers pueden interpretar el lenguaje de marcado HTML5 con casi la totalidad de sus funciones, por lo que los desarrollos basados en esta tecnología nos garantizan que funcionan casi por igual en todas las plataformas.

Un aspecto muy positivo de HTML5 es que no solo se decidió a definir las etiquetas principales del lenguaje HTML, sino que también integró un compendio de nuevas tecnologías que se fueron integrando a los dispositivos. Entre ellas, destacamos: **geolocalización**, **almacenamiento web**, **web sockets** y **web workers**, entre otras bondades del lenguaje.

Todas estas mejoras suponen una actualización que potencia el conjunto de herramientas ya existente y habilita a que diseñadores y programadores web puedan crear páginas, sitios ricos en contenido, sin depender de herramientas o plugins de terceros.

HTML5 garantiza que quien diseña una web basada en este nuevo paradigma logrará obtener una mejor capacidad de respuesta y velocidad. Además, al combinar este lenguaje con CSS3 y JS, cualquier diseñador podrá resolver, de forma nativa, tareas complejas como lo son la edición de imágenes, efectos de sombras, representación de mapas, visualización de videos y reproducción de audio, así como obtener la ubicación física de un usuario, entre otras tareas. Y lo mejor es que hará todo esto sin tener que recurrir a un desarrollo nativo.

Figura 4. Para conocer en detalle lo que nos brinda HTML5, podemos recorrer el sitio **www.mobilehtml5.org**, donde encontraremos su cobertura según navegador y plataforma.

Si bien existen diferencias y no todos los navegadores web soportan todas las características de HTML5, la mayoría presta el soporte necesario para que las WebApps o sitios web basados en este nuevo estándar puedan visualizarse correctamente.

Declaraciones y metatags

Debemos aclarar, antes de iniciar el repaso de la estructura de **declaraciones** y **metatags**, que HTML4 y HTML5 son compatibles recíprocamente. No hay restricciones en cuanto al diseño de una página, y todo el código que escribamos en HTML4 funcionará sin problemas en HTML5. De igual manera, si una página realizada en HTML4 contiene etiquetas específicas de HTML5, estas funcionarán sin ningún inconveniente.

Doctype

Toda página web se inicia con la declaración de las etiquetas **<html>** y se cierra con la misma etiqueta **</html>**. Dentro de estas, se estructura el encabezado de la página, **<head></head>**, y el cuerpo de la página, **<body></body>**.

Antepuesta a la etiqueta HTML que inicia la estructura de estos archivos, debemos agregar la palabra **!doctype**, que significa **tipo de documento**. Esto se utiliza para que el navegador identifique el tipo de archivo que está por abrir. Un navegador web puede abrir tanto páginas web como páginas XML, imágenes y, en las últimas versiones de los browsers más populares, también puede abrir un **archivo PDF** directo, sin necesidad de un plugin o aplicación cliente previamente instalada.

DEFINICIONES DEL LENGUAJE

Las definiciones de HTML5 aún no están cerradas y se estima que para fines de 2014 recién se publicará el estándar definitivo. Aun así, los principales navegadores web ya están brindando el soporte correspondiente.

Dentro del encabezado de página **<head>** se agregan declaraciones y referencias a otros archivos que requiere la página en sí, como:

- Referencias a hojas de estilo CSS.
- Referencia a archivos JavaScript que contienen funciones.
- Referencia a **iconos** o **favicons**.
- Funciones en sí de JavaScript que no se agregan en ningún archivo.
- El título en sí que muestra la página en el navegador web.

```
<!doctype html>
<head>
        <title>Esto es una página web</title>
        <Script>
        Function Saludo() {
                Alert('Hola mundo');
                }
        </Script>
        <link rel="stylesheet" type="text/css" href="estiloredusers.css">
</head>
<body>
<!-- Estructura de la página, menús, textos, imágenes, videos, etc -->
</body>
</html>
```

En el cuerpo de página **<body>** se incorpora todo el contenido que normalmente visualizamos en las páginas web de los sitios que visitamos. Entre ellos:

- El logo de la empresa.
- La barra de navegación o menús.
- Fragmentos de textos e imágenes.
- Video, audio, links hacia otros tipos de archivo.
- Galería de imágenes.

Si aún no hemos instalado ningún editor de HTML (como, por ejemplo, Dreamweaver), podemos abrir **Notepad** o **Notepad++** y copiar el código de estos ejemplos en el orden en que fueron expuestos. Luego guardamos el archivo en formato HTML bajo el nombre **index.html**, y hacemos doble clic sobre él para verlo en el navegador.

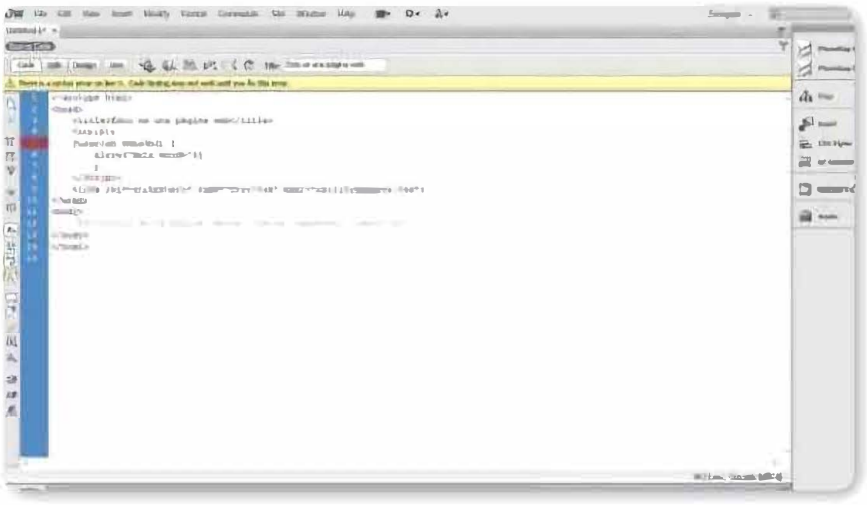


Figura 5. Dreamweaver permite identificar la estructura HTML, indicando con diferentes colores cada sección y notificando si hay un error de tipeo.

Charset

Dentro del **doctype** correspondiente a HTML4, se declara **charset**, que es utilizado para definir el idioma en el cual se realizará la página web (español, inglés, francés, portugués, etcétera). Esto afecta la visualización de los caracteres.

```
... Strict//EN" ...
```

En HTML5 se simplificó esto, generando un **charset internacional**, que permite la correcta visualización de todos los caracteres existentes.

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

EDITORES HTML

Para el objetivo de este libro, utilizaremos **Dreamweaver** como editor HTML predeterminado. Desde **www.adobe.com** podemos descargar la versión de prueba que dura 30 días. Si no, también podemos recurrir a alternativas como **Notepad++**, **Eclipse** o **Netbeans**, que también resaltan la sintaxis del lenguaje.

Si deseamos declarar un charset específico para que la página se vea correctamente en inglés, español, o en el idioma que fuere, debemos buscar la referencia **ISO** específica al charset del idioma necesario. Por ejemplo, en español se utiliza **Charset=ISO-8859-1**.

> PARA DECLARAR UN CHARSET ESPECÍFICO DEBEMOS BUSCAR LA REFERENCIA ISO DEL IDIOMA NECESARIO

Estructura de un documento HTML5

Ya conocemos cómo se estructura un documento HTML, tanto en su encabezado **<head>** como el contenido del cuerpo de la página **<body>**. Nos queda por delante repasar las secciones creadas para HTML5 que representan el contenido de una página web.

Todos los sitios incorporan en su estructura una metodología similar para agrupar las diferentes secciones de un sitio. Dentro de estas, podemos destacar las siguientes:

- Un encabezado (texto y/o logo).
- Una barra de navegación (lateral o superior).
- Un área central donde figura(n) la(s) noticia(s) o información general.
- Una barra lateral con información o links adicionales.
- Un pie de página.

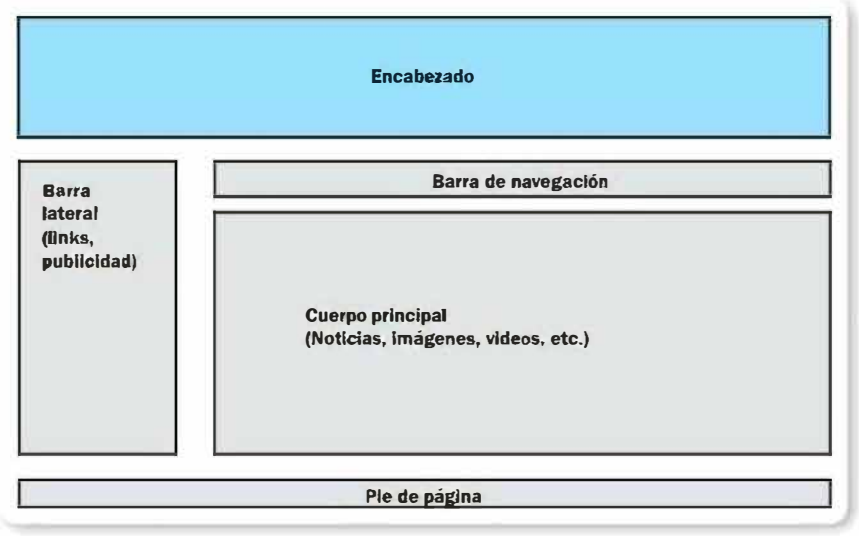


Figura 6. Representación gráfica de las secciones de un sitio promedio. HTML5 buscó nombrar cada sección de una manera mucho más amigable.

HTML5 etiquetó cada sección con un nombre que la referencia amigablemente, para mejorar la tarea del diseñador web, como también la de los motores de indexación. Veamos, en la siguiente tabla, el nombre de cada elemento y su manera de implementación.

NUEVOS ELEMENTOS EN HTML5

ELEMENTO	CÓDIGO DE IMPLEMENTACIÓN	DESCRIPCIÓN
Header	<header id="encabezado">	Cabecera de la página.
Hgroup	<hgroup> Título 1 Subtítulo 2 </hgroup>	Permite agrupar tags del tipo título dentro de Header.
Nav	<nav> <ahref="link1.html">link 1 link 2 ... </nav>	Permite crear una barra de navegación principal que contiene diversas etiquetas.
Section	<section> Título 2 Texto de la sección </section>	Permite definir un área de contenido dentro de un sitio web. Se utiliza habitualmente en los blogs, para agrupar cada noticia o post.
Footer	<footer> Copyright 2014 – Redusers – Fernando Luna </footer>	Representa el pie de página de un sitio web.

EL ECOSISTEMA CHARSET

Dada la diversidad de lenguas existentes en el mundo, es complicado abordar en profundidad todo lo referente al charset. Y no podemos dejarlo de lado porque en el futuro nos pueden requerir el desarrollo de aplicaciones en cualquier idioma. Por ello, debemos apuntar el siguiente link de referencia para sacarnos cualquier duda: **www.desarrolloweb.com/articulos/juego-caracteres-charset-html.html**

ELEMENTO	CÓDIGO DE IMPLEMENTACIÓN	DESCRIPCIÓN
Article	<article> Visual Basic 2010 El libro Manual Visual Basic 2010 permite conocer a fondo la herramienta complementaria de Visual Studio... </article>	Define diversas zonas con determinado contenido dentro de otro elemento. Estos contenidos pueden anidarse para luego ser extraídos fácilmente y distribuidos a través de servicios RSS.
Aside	<aside> Escapadas Bahía Manzano – V. la Angostura Cabañas Huapi – Bariloche ... </aside>	Permite estructurar en un sitio web todo contenido que no tenga relación con los elementos primarios de este, como ser una barra lateral de información.

Tabla 1. Nuevos elementos de estructura/semánticos en HTML5.

Existen muchas más etiquetas que nos permiten crear una página web moderna y que, combinadas con CSS, ayudarán a cualquier desarrollador o diseñador web en la estética y la estructura, pero el objetivo de este libro es abarcar el desarrollo web mobile y cómo se integra este con los dispositivos móviles. Por esta razón, dejaremos el diseño estético de cada aplicación en manos de una librería como jQuery Mobile.

Si desean conocer más a fondo sobre HTML5 podrán hacerlo a través del **manual HTML5** publicado por esta misma editorial.

NUEVOS Y VIEJOS ELEMENTOS HTML5

Debemos tener en cuenta que HTML5 está en constante desarrollo, por lo que los elementos que existen actualmente pueden variar o hasta no ser tenidos en cuenta al momento de dictaminar las especificaciones finales. En el portal **w3c.org** podemos consultar la información precisa y actualizada.

HTML5 para aplicaciones móviles

Los dispositivos móviles modernos adoptaron HTML5 para explotar de manera eficiente el acceso que este lenguaje brinda al hardware y al sistema operativo. Y si pensamos en dispositivos móviles, sabemos en que la mayoría de estos reina la **interfaz de escritura táctil**.

Estas interfaces permiten el ingreso de datos, alfanuméricos o numéricos, a través de un teclado virtual, por lo cual este teclado, a diferencia de los físicos, puede limitar su visualización de teclas en base al campo de texto en el cual se debe ingresar información. Esto permite facilitar la vida del desarrollador o diseñador web, ya que se evita validar, mediante algoritmos complejos, los datos ingresados en cada tipo de campo.

A continuación, repasaremos las propuestas de HTML5 que facilitan el trabajo de cargar información específica en una página web.

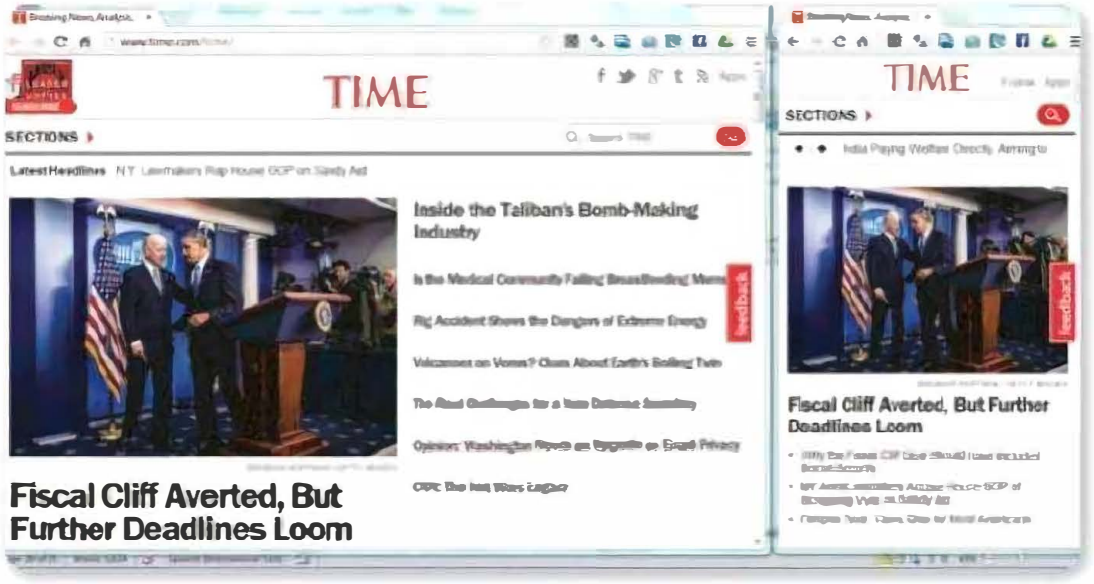


Figura 7. Desde la proliferación de los dispositivos móviles, el **diseño responsivo** tomó un papel muy importante en el terreno web.

Componentes <Input Type>

Las etiquetas **<Input Type>** permiten establecer el tipo de campo que se visualiza en un formulario web. HTML5 propuso cambios a estas

etiquetas para facilitar, en los navegadores de escritorio, la validación de datos ingresados y, en los dispositivos móviles táctiles, el ingreso de información a través del teclado correcto.

Veamos, a continuación, cómo implementar favorablemente la etiqueta **<Input Type>** en el diseño de aplicaciones móviles.

Input Type Color

Este input type permite visualizar la paleta de colores para que seleccionemos uno de ellos. Solamente funciona en los navegadores Chrome y Opera.

```
Por favor, seleccione su color favorito: <input type="color" name="colorfavorito">
```

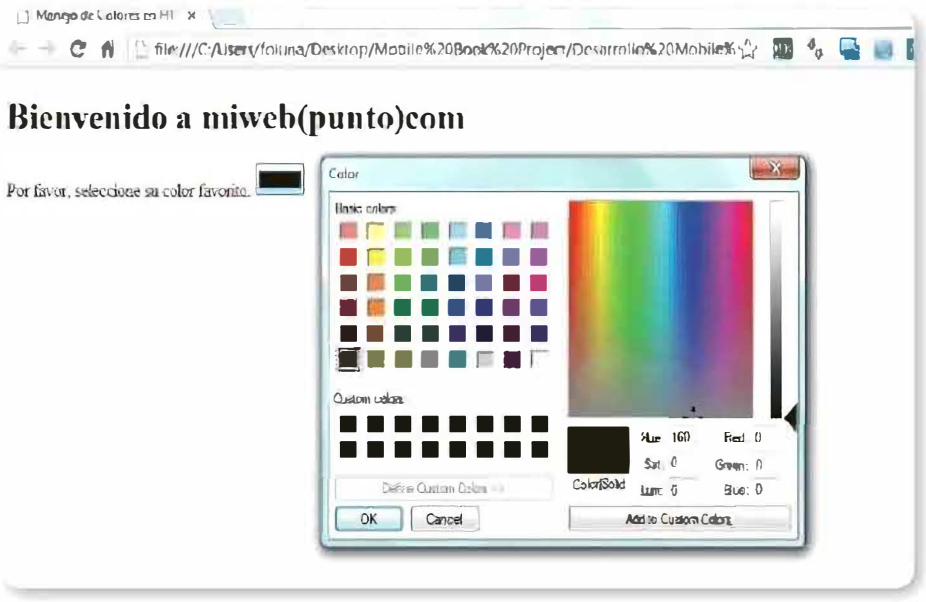


Figura 8. Selección de un color dentro de la paleta de colores disponibles en el sistema operativo.

INPUT TYPES Y MOTORES DE RENDERIZACIÓN

Debido a la fragmentación de motores de renderización, muchas veces encontraremos que las etiquetas **<Input Type>** no se comportan por igual en todas las plataformas. Mayormente esto ocurre en aquellas del tipo **Color**, **Date** y **Range**, entre otras.

Input Type Date

Permite manipular campos que requieran el ingreso de una fecha. Es soportado, por el momento, por los navegadores Chrome, Safari y Opera.

```
<nav>
    Seleccione su fecha de nacimiento: <input type="date"
name="nacimiento">
</nav>
```

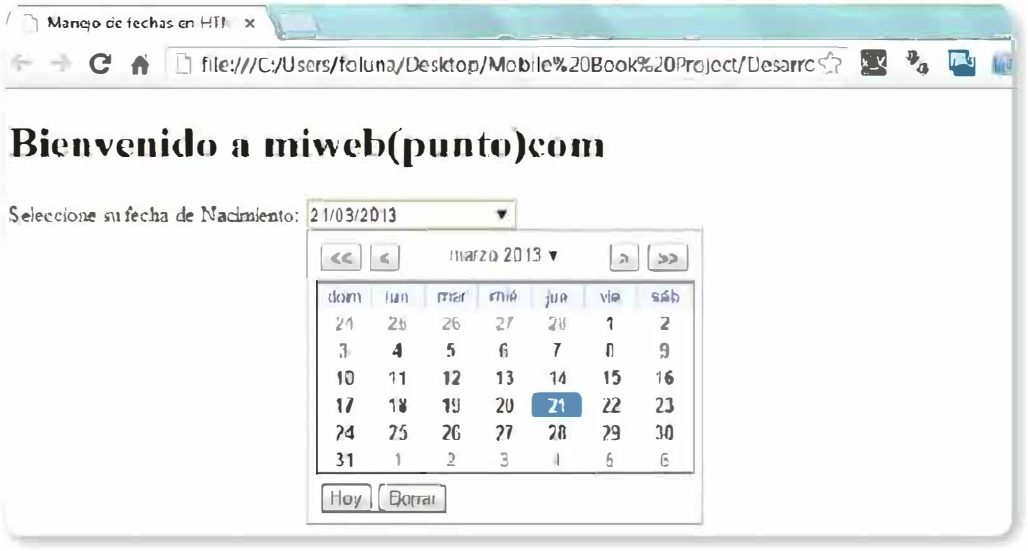


Figura 9. Selección de la fecha desde un combo desplegable. La forma de visualizar el calendario cambiará según el sistema operativo y navegador.

Input Type Datatime y Datetime-local

En ambos casos, nos permite manipular un campo específico donde seleccionamos la fecha y hora. Veamos el ejemplo para **<Input Type Datetime>**:

```
Ingrese fecha y hora de nacimiento: <input type="datetime" name="bdaytime">
```

Y la fecha, hora y GMT correspondiente para **<Input Type datetime-local>**:

```
Ingrese fecha, y GMT hora de su nacimiento: <input type="datetime-local"
name="bdaytime_local">
```

Estos dos elementos solo funcionan correctamente en los navegadores Safari y Opera.

Input Type email

Permite definir un campo donde solo se ingrese una dirección de correo electrónico. Este input type funciona solamente en Opera, Chrome y Firefox.

```
Ingrese su e-mail: <input type="email" name="email">
```

Figura 10. En los dispositivos móviles, el teclado dispondrá de la tecla arroba dentro de las teclas predeterminadas.

Input Type URL

Al igual que la anterior, **<Input Type URL>** permite ingresar una dirección URL en un campo. Al momento, es soportado en Firefox, Chrome y Opera.

```
Ingrese también la URL de su web: <input type="url"
name="web">
```

INPUT TYPE URL PERMITE INGRESAR UNA DIRECCIÓN URL EN UN CAMPO

Figura 11. Para este input type, se incluye en el teclado virtual una tecla de acceso rápido **.COM**, y se visualizan solo los caracteres permitidos en una dirección URL.

Input Type Search

Este input type muestra un campo de búsqueda que permite ingresar una leyenda informativa que desaparece al escribir sobre el campo.

```
<nav>
    Ingrese el texto a buscar: <input type="search" name="websearch"
placeholder="Texto a buscar"> <input type="button" value="Buscar">
</nav>
```

Este atributo funciona bien en Google Chrome y Safari. La leyenda también se verá en otros navegadores como Firefox, aunque en este no se mostrará el botón automático de eliminación de texto.

MARCA DE AGUA

Todos los campos de texto permiten el uso de la propiedad **placeholder="algún texto"**. Esto sirve como una especie de orientación al usuario para que ingrese los datos correctos en el campo, y es muy útil para aplicar en pantallas con mucha información y de dimensiones reducidas.

Figura 12. En Chrome y Safari este input type funciona a la perfección. Es posible lograr lo mismo en otros navegadores, pero ya deberá entrar el código JavaScript como intermediario.

Input Type Range

Permite visualizar un control estilo **Slider** y establecer un valor mínimo, un valor máximo y el intervalo de cambio entre uno y otro valor.

```
Seleccione el valor correcto: <input type="range" name="valores" min="1"
max="15">
```

En el ejemplo, establecemos su valor entre 1 y 15. En cambio, no establecimos el atributo **step**, con lo cual variará uno a uno su valor. Si lo establecemos en **step="3"**, al deslizar el control, este tendrá como posibles valores: 1, 3, 5, 7, 9, 11, 13, 15.

OTROS INPUT TYPES

También existen otros input types, como **<Input Type Week>** e **<Input Type Month>**. Estos permiten ingresar los valores numéricos de las semanas y meses respectivamente. Funcionan correctamente en Google Chrome, Safari y Opera.

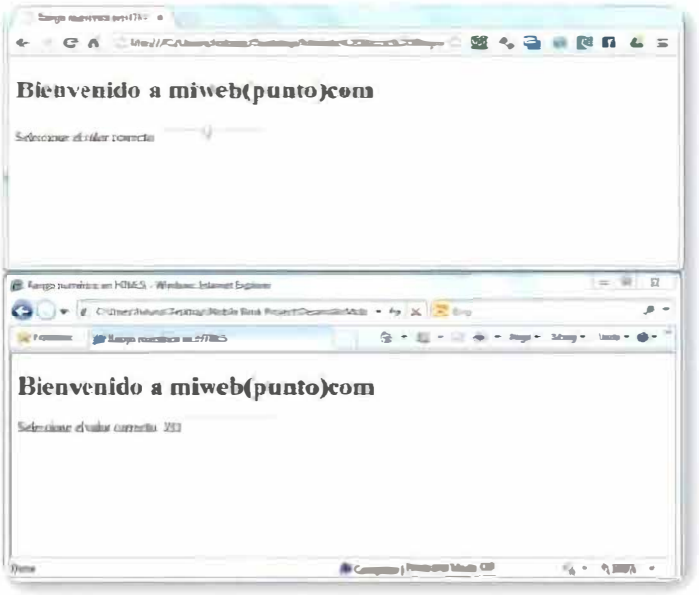


Figura 13. Mientras que Google Chrome visualiza correctamente el control **Slider**, si probamos el sitio en IE, veremos que solo se muestra una caja de texto y en ella podremos ingresar cualquier valor.

Input Type Tel

Este campo permite ingresar un número de teléfono. A simple vista, parece ser un campo común y no tiene ningún efecto en los navegadores de escritorio. Solo veremos su efecto en los navegadores web de los teléfonos móviles, donde, al posicionar el cursor en ese campo, se habilitará el teclado numérico en el dispositivo, para poder ingresar solo números.

```
Ingrese su número de teléfono: <Input Type="Tel" name="teléfono">
```

Input Type Number

Este input type funciona igual que **Range**, pero visualiza un control **Spin** en lugar de un control **Slider**, donde podemos aumentar o disminuir su valor dentro de un rango numérico especificado en los atributos **Min** y **Max**.

```
Ingrese el monto a transferir (en pesos) $:<input type="number"
name="transferencia" min="100" max="500" step="100">
```

También posee el atributo **Step**, para establecer el intervalo entre número y número. Funciona en Chrome, Safari y Opera.

Figura 14. Este control tipo **Spin** permite indicar un rango mínimo y máximo y, también, ingresar el valor de forma manual, siempre y cuando se respeten los rangos establecidos.

Geolocalización

Los navegadores web modernos incluyen una función de **geolocalización**, a través de la cual podemos obtener las coordenadas de la posición del equipo mediante un mapa. Dado que todas las direcciones **IP** de internet tienen una dirección física asignada según su uso, es posible estimar las coordenadas aproximadas de los equipos.

Las computadoras, tablets o teléfonos que acceden a la Web a través de tecnologías 3G o 4G permiten triangular una ubicación mucho más precisa de las coordenadas, lo cual nos dará un resultado más acertado que el que puede brindar una computadora de escritorio conectada a internet por cable módem o ADSL.

A través de una función de JavaScript, podemos obtener los valores de latitud, longitud y altitud aproximados del equipo y, utilizando un servicio como **Google** o **Bing Maps**, representarlas en un mapa.

Para comprender mejor los términos hasta aquí repasados, realizaremos un ejercicio práctico que nos permitirá comprender la

estructura de los documentos HTML5, cómo integrar en ellos las librerías JavaScript, cómo invocar un mapa desde la API de **Google Maps**, y cómo realizar la detección de nuestra ubicación mediante la geolocalización.

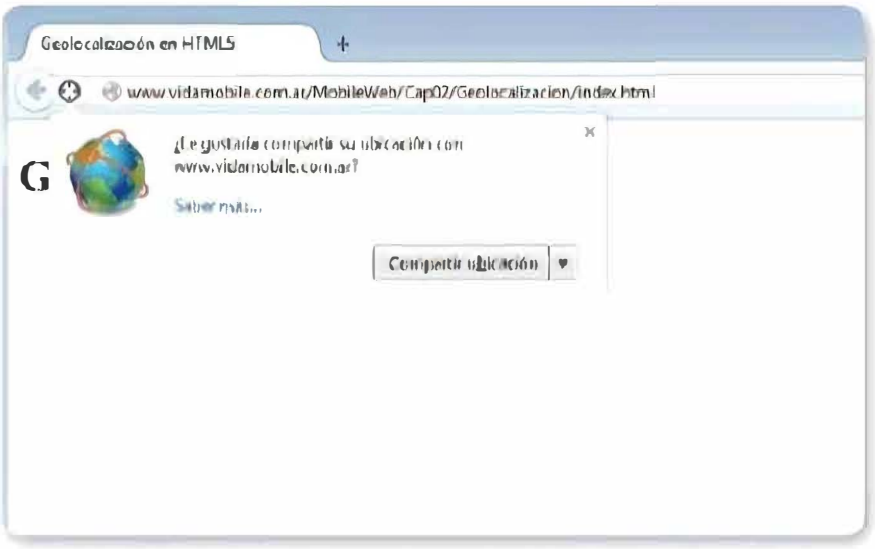


Figura 15. Ya sea desde un navegador web de escritorio o de un dispositivo móvil, la geolocalización puede ser utilizada sin problema alguno.

Elementos para el ejercicio

Para poder realizar el ejercicio debemos preparar los siguientes elementos:

- Un archivo HTML, denominado **index.html**.
- La API de Google Maps para ubicar nuestra posición.
- La invocación a un CDN para uso de la librería jQuery.
- Un sitio web gratuito para alojar nuestras pruebas, o bien un web server local.

Creación de la página HTML

La página HTML que crearemos a continuación contendrá el código necesario (HTML y JavaScript) para poder realizar nuestro ejercicio. Dicha página será alojada para su prueba en una web gratuita de las que existen en internet. Si no queremos subir nuestras pruebas a un sitio accesible por todos, podemos optar por un web server instalable en Windows, Linux o Mac OS, entre las distintas opciones existentes.

El código base de **index.html** es el siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <title>Geolocalización en HTML5</title>
  <meta charset="utf-8">
  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></
script>
</head>

<body>
</body>
</html>
```

Creemos el archivo de nuestro primer proyecto y hagamos una copia para tenerlo listo para los próximos ejercicios a elaborar en los capítulos sucesivos.

Integración de librerías JS

Para poder implementar correctamente nuestro mapa, utilizaremos la API JavaScript de Google Maps que nos permitirá visualizar nuestra posición. Sumada a esta, también declararemos la librería jQuery, que nos simplificará la tarea de escribir el código JS y de referenciar a cada objeto creado en la página. En el **Capítulo 3** conoceremos en detalle esta librería y sus múltiples usos.

Dentro del encabezado del archivo HTML, agregamos las siguientes líneas de código:

```
...
  <header>
    <h1>Geolocalización en HTML5</h1>
  </header>
  <section>
    <article>
      <div id='map_canvas' style='width:100%; height:100%;'></div>
    </article>
    <div id="respuesta">
```

```
    </div>
  </section>
...
```

El tag **<header>** permite crear un título del tipo **<h1>**. A continuación, abriremos un tag **<section>** donde declararemos un nuevo tag **<article>** seguido de un ** ** que contiene una etiqueta del tipo **canvas**. Esta permitirá cargar luego el mapa que visualizará nuestra posición.

Luego agregamos el siguiente código:

```
...
    <script type="text/javascript" src="http://maps.google.com/maps/api/
js?sensor=true"></script>

  </script>
  ...
```

Este código inicia un nuevo script donde incluiremos las funciones correspondientes a geolocalización. Estas funciones solicitarán permiso al usuario para ubicar su posición, validarán los errores que puedan ocurrir en la aceptación o rechazo de la ubicación y dibujarán el correspondiente mapa en la etiqueta **Canvas** para mostrar nuestro lugar.

Incluyamos, a continuación, el siguiente código dentro del tag script:

```
<script type="text/javascript">
    var map;
    var latitud;
    var longitud;
    var precision;

    $(document).ready(function() {
        localizame();
    });
```

Las variables **map**, **latitud**, **longitud** y **precision** almacenarán los datos de nuestra ubicación cuando aceptemos que nos localice la página.
La siguiente función es una invocación a jQuery, que permitirá ejecutar la función **localizame()** cuando la página HTML se haya cargado por completo.

Comencemos a escribir las funciones:

```
function localizame() {
   if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(coordenadas, errores);
   }else{
      alert('Este navegador no soporta geolocalización.');
   }
}
```

Esta función invoca al método **navigator.geolocation.getCurrentPosition()**, que recibe dos parámetros: **coordenadas** y **errores**. Estos parámetros son del tipo función, que, a su vez, contiene otras funciones a invocar.

```
function coordenadas(position) {
   latitud = position.coords.latitude;
   longitud = position.coords.longitude;
   precision = position.coords.accuracy;
   cargarMapa();
    }
```

Esta función setea los valores correspondientes a las variables declaradas en el inicio del script, de acuerdo a los valores recibidos del método **getCurrentPosition**: latitud, longitud y precisión. Por último, ya cargados los valores en las variables, se invoca al método **cargarMapa()**.

```
function errores(err) {
   if (err.code == 0) {
    alert("Error general no identificado.");
   }
   if (err.code == 1) {
    alert("El usuario no aceptó compartir su posición");
   }
   if (err.code == 2) {
    alert("No se puede obtener la posición actual");
   }
   if (err.code == 3) {
```

```
            alert("Tiempo de espera superado");
        }
    }
```

La sentencia **err** nos permite detectar los errores que ocurren en las diversas invocaciones de JavaScript. A través de la propiedad **err.code**, podemos detectar si hubo o no un error. Si esta devuelve **0**, es porque no ha ocurrido ningún error. Si devuelve **1**, es porque el usuario no aceptó compartir su ubicación. El error **2** se debe a que no se pudo determinar la ubicación del usuario, y el error **3**, a que la función **getCurrentPosition** superó el tiempo de espera máximo para devolver la posición del usuario.

```
function cargarMapa() {
    var latlon = new google.maps.LatLng(latitud,longitud);
    var misParametros = {
        zoom: 17,
        center: latlon,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    map = new google.maps.Map($("#map_canvas").get(0), misParametros);

    var coordenadasMarcador = new google.maps.LatLng(latitud,longitud);

    var marcador = new google.maps.Marker({
        position: coordenadasMarcador,
        map: map,
        title: "Obtener mi ubicación actual"
    });
}
```

Por último, nos queda la función principal: **cargarMapa()**. Esta función permitirá establecer la invocación a Google Maps y pasarle todos los parámetros necesarios para dibujar nuestra ubicación en el mapa.

Veamos el código completo de nuestra página HTML:

```
<!DOCTYPE html>
```

```
<html>
<head>
  <title>Geolocalización en HTML5</title>
  <meta charset="utf-8">
  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></
script>
</head>

<body>
  <header>
    <h1>Geolocalización en HTML5</h1>
  </header>
  <section>
    <article>
      <div id='map_canvas' style='width:100%; height:100%;'></div>
    </article>
    <div id="respuesta">

    </div>
  </section>

  <script type="text/javascript" src="http://maps.google.com/maps/api/
js?sensor=true"></script>

  <script type="text/javascript">
    var map;
    var latitud;
    var longitud;
    var precision;

    $(document).ready(function() {
      localizame();
    });

    function localizame() {
      if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(coordenadas, errores);
```

```
 }else{
      alert('Este navegador no soporta geolocalización.');
 }
}

function coordenadas(position) {
 latitud = position.coords.latitude;
 longitud = position.coords.longitude;
 precision = position.coords.accuracy;
 cargarMapa();
  }

function errores(err) {
 if (err.code == 0) {
    alert("Error general no identificado.");
 }
 if (err.code == 1) {
    alert("El usuario no aceptó compartir su posición");
 }
 if (err.code == 2) {
    alert("No se puede obtener la posición actual");
 }
 if (err.code == 3) {
    alert("Tiempo de espera superado");
 }
}

function cargarMapa() {
 var latlon = new google.maps.LatLng(latitud,longitud);
 var misParametros = {
      zoom: 17,
      center: latlon,
      mapTypeId: google.maps.MapTypeId.ROADMAP
 };
 map = new google.maps.Map($("#map_canvas").get(0), misParametros);

 var coordenadasMarcador = new google.maps.LatLng(latitud,longitud);
```

```
            var marcador = new google.maps.Marker({
                position: coordenadasMarcador,
                map: map,
                title: "Obtener mi ubicación actual"
            });
        }
    </script>
</body>
</html>
```

Ya creada nuestra página HTML de geolocalización, podemos subirla al servidor web y probarla. Las pruebas se pueden realizar desde un navegador web de escritorio, pero lo más acertado será probarlo desde un navegador web móvil. En lo posible, desde un smartphone o tablet que soporte conectividad 3G.

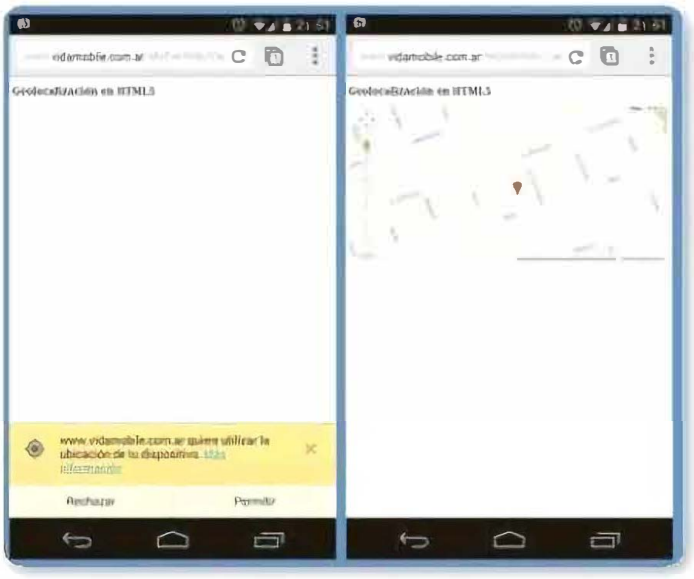


Figura 16. El resultado de nuestro ejercicio muestra cómo el sistema solicita permiso al usuario antes de mostrar su ubicación en Google Maps.

RESUMEN

Este capítulo nos permitió adentrarnos un poco en el nuevo mundo propuesto por HTML5 y conocer la inclusión de varias funciones útiles que nos permiten desarrollar mejores sitios web. A su vez, también realizamos un repaso por las diferencias que podemos encontrar en distintas plataformas y/o distintos navegadores web. Esto constituye un dato muy útil a tener en cuenta al momento de desarrollar una aplicación web en la que buscamos cubrir lo mejor posible todas las plataformas existentes.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué función cumple **HTML** en la Web?
2. ¿Cuál es la versión de HTML utilizada actualmente?
3. Mencione tres características distintivas de **HTML5** respecto de su versión anterior.
4. ¿Cuál es la estructura actual básica que compone un documento HTML?
5. ¿HTML5 funciona por igual en todos los navegadores web existentes?
6. ¿**Input type** sigue siendo un único estándar dentro de HTML5?
7. Mencione al menos dos clases de input types soportados en HTML5.
8. ¿La **geolocalización** determina automáticamente la posición del usuario?
9. ¿Los navegadores web de escritorio soportan geolocalización?
10. ¿Qué hay que tener en cuenta para geolocalizar a alguien con la mayor precisión?

EJERCICIOS PRÁCTICOS

1. Realice los ejercicios invocando a los diferentes **<Input Type>** en páginas HTML distintas.
2. Instale al menos tres navegadores diferentes en su computadora.
3. Visualice las páginas HTML con los input types en cada uno de los navegadores.
4. Pruebe el sistema de geolocalización en un smartphone o tablet.
5. Agregue dos etiquetas al ejercicio de geolocalización que permitan visualizar la latitud y longitud obtenida por la función **localizar()** en la página HTML.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*03

Fundamentos de jQuery Mobile

La estética de una aplicación web móvil es muy importante en cada uno de los proyectos que abordamos. Y lo mejor para reducir los tiempos de desarrollo es recurrir a herramientas que nos ayuden a enfocar nuestro esfuerzo en la funcionalidad del sitio sin dejar de lado la estética. En los próximos capítulos nos centraremos en jQuery Mobile, una de las herramientas que cumple con creces esta función.

¿Qué es jQuery Mobile?

jQuery Mobile es un conjunto de librerías JavaScript y CSS, construido específicamente para dar soporte a múltiples plataformas al momento de diseñar tanto un sitio web como una aplicación móvil. Permite, de manera práctica y sin complicaciones, mantener la estructura de un sitio por igual, al momento de visualizarlo desde una computadora de escritorio, tablet, smartphone o teléfono celular de baja, media o alta gama con soporte para navegación web.

Uno de los pasos más importantes al momento de diseñar una WebApp o app móvil es definir bien la interfaz de usuario que la aplicación tendrá y, por supuesto, cómo se verá en cada una de las plataformas donde podrá ser ejecutada.

Si decidimos hacer una aplicación web que pueda ejecutarse tanto en un navegador de escritorio como en un navegador web móvil, debemos tener en cuenta que su interfaz sea lo más similar posible para que los usuarios concurrentes no deban tener que aprender dos o más veces a desplazarse por nuestro sitio.

Una de las primeras investigaciones que realizan los usuarios de aplicaciones móviles, al momento de decidir cambiar hacia otro sistema operativo móvil, está destinada a saber si las apps de uso cotidiano tienen su versión en la plataforma a la cual piensan migrar.

Desde nuestro punto de vista, al momento de desarrollar apps web móviles, la opción de cambio de usuario no nos debería afectar en lo más mínimo si tenemos las herramientas necesarias para afrontar un desarrollo que funcione por igual en todas las plataformas.

jQuery Mobile fue pensado desde sus inicios para brindar la mayor compatibilidad posible en casi todas las plataformas existentes, sean o no móviles. Para estas últimas, jQuery Mobile enfatizó en cubrir la mayor cantidad de equipos y web browsers que existen en el mercado,

SIMPLEZA SIN TECNICISMOS

jQuery Mobile permite escribir aplicaciones web y móviles de forma básica y sin ningún esfuerzo. El desarrollo de este framework fue pensado para una integración amigable para diseñadores y programadores, sea cual fuere el nivel de sus conocimientos.

lo que garantiza que nuestro sitio se verá –prácticamente– sin cambios tanto en un smartphone de alta gama como en uno de baja gama.

Figura 1. **www.jquerymobile.com** es el punto de partida inicial para trabajar con este complemento pensado íntegramente para aplicaciones móviles.

jQuery Mobile = jQuery, ¿o no?

La respuesta a esta pregunta es sí: jQuery y jQuery Mobile son diferentes. jQuery es una librería que complementa las webs y hace más ameno el desarrollo de un sitio, resumiendo varias funciones kilométricas –en cuanto a líneas de código refiere– en funciones más amigables.

jQuery Mobile, por su parte, está basado en jQuery: debemos utilizar ambas librerías juntas, pero la primera facilita de forma general a los programadores la invocación de cada función, que aparece resumida en forma de estilos aplicables a los elementos que componen una página web. jQuery Mobile busca, de esta manera, agilizar el desarrollo de una página web, cumplir con el requisito de **Responsive Web Design** y evitar que los diseñadores deban escribir código complejo.

> JQUERY MOBILE SE BASA EN JQUERY Y DEBEN USARSE JUNTAS, AUNQUE SON LIBRERÍAS DISTINTAS

¿Utilizar jQuery Mobile de forma local o remota?

Para implementar jQuery Mobile en un sitio web, webApp o aplicación híbrida se requiere incluir, en los archivos HTML que compondrán el sitio, una referencia a este conjunto de librerías.

Veamos, a continuación, un ejemplo de código, donde invocamos a las librerías de jQuery Mobile dentro de una página web llamada **index.html**:

```
<!DOCTYPE html>
<html>
<head>
...
   <link rel="stylesheet" href="jqm/jquery.mobile-1.1.0-rc.1.min.css" />
   <script src=" jqm/jquery-1.7.1.min.js"></script>
   <script src=" jqm/jquery.mobile-1.1.0-rc.1.min.js"></script>
</head>
<body>
...
</body>
</html>
```

Dentro de la etiqueta **<head>** y **</head>** debemos incluir las referencias a las librerías de jQuery Mobile. Estas se componen por:

- El archivo **Jquery.mobile???.css**, que es la hoja de estilos.
- El archivo **Jquery???.min.js**, que es el archivo JavaScript correspondiente a jQuery.
- El archivo **Jquery.mobile???.js**, que corresponde al archivo JavaScript propio de jQuery.

La denominación "**???**" utilizada equivale al número de versión correspondiente de las librerías. jQuery Mobile está en constante desarrollo, por lo que la versión utilizada al momento de escribir este capítulo variará significativamente a la versión existente cuando el libro llegue al mercado.

Aun así, podemos optar por utilizar una librería antigua de jQuery Mobile sin problema alguno. En las últimas versiones lanzadas al mercado, se simplificó la cantidad de temas estéticos que permiten variar la gama de colores de una WebApp entre cinco diferentes

opciones. La versión actual de este framework solo permite utilizar dos temas diferentes y optar por crear un tema específico a gusto del desarr

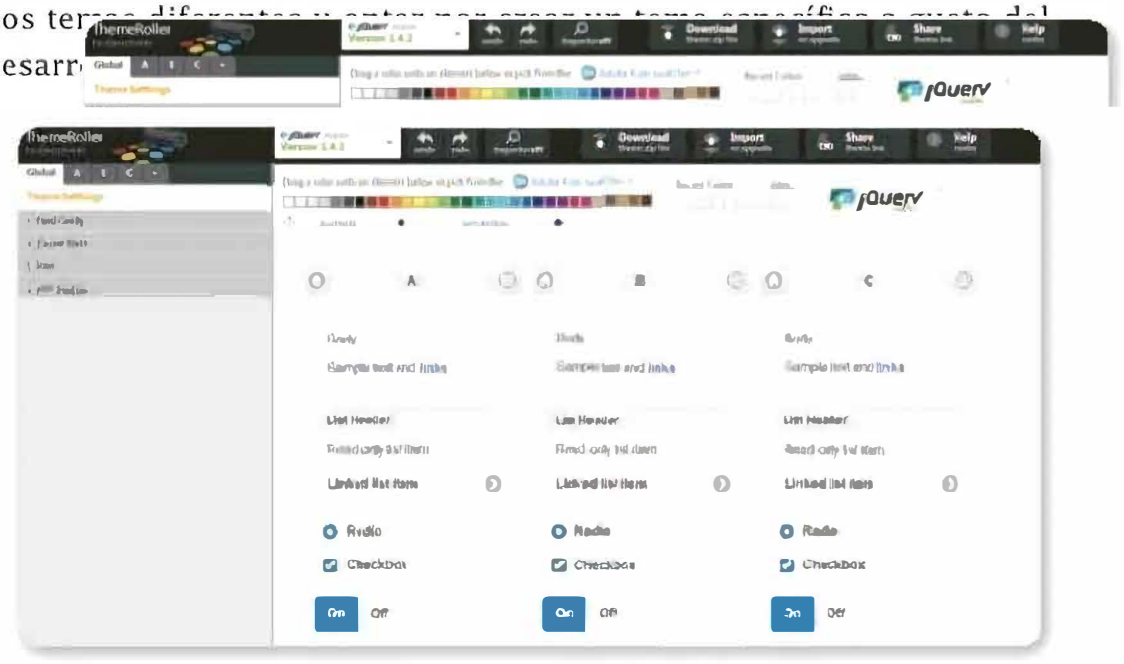


Figura 2. ThemeRoller es la herramienta que provee jQuery Mobile para desarrollar temas personalizados para nuestros proyectos.

Nosotros utilizaremos **jQuery Mobile 1.3.2** para los primeros ejemplos de este libro; más adelante, repasaremos la última versión existente al momento de escribir esta obra.

Instalar JQM de forma local

jQuery Mobile puede descargarse de forma local y configurarse directamente en los servidores donde desplegamos nuestra WebApp, apuntando las librerías a la subcarpeta donde almacenamos todo el framework.

Si deseamos realizar esta tarea, debemos descargar la versión de jQuery Mobile correspondiente a la que utilizaremos en estos ejemplos, desde el siguiente link: **http://jquerymobile.com/resources/download/jquery.mobile-1.3.2.zip**.

A continuación, ubiquemos la **versión 1.3.2** de esta librería y procedamos a su descarga.

> JQUERY MOBILE PUEDE DESCARGARSE DE FORMA LOCAL O USARSE DESDE UN PROVEEDOR REMOTO

Utilizar JQM desde un CDN

También es posible utilizar JQM desde un proveedor remoto de librerías, lo que se conoce como **CDN** (**Content Delivery Network**). Entre los más conocidos encontramos a Google, Microsoft y jQuery. Estos se ocupan de alojar las librerías de jQuery, jQuery Mobile y jQuery UI, entre otras tantas más, y así evitar que nosotros tengamos que ocupar espacio en nuestro servidor web o incrementar sustancialmente el tamaño de instalación de una app nativa o híbrida.

Veamos, a continuación, un ejemplo de cómo llamar a las librerías remotas alojadas en el CDN de jQuery:

```
<!DOCTYPE html>
<html>
<head>
   <title>Título de la página</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">

   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2-rc.1/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.8.2.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
</head>
<body>
...
</body>
</html>
```

Desventajas en el uso de CDN

La desventaja de utilizar un CDN en una app es que esta dependerá –al menos la primera vez que se inicie– de que exista conectividad a internet para poder descargar las librerías remotas al disco local del dispositivo. Si el equipo donde instalamos la app no tiene conectividad, difícilmente se pueda utilizar de manera correcta, y probablemente visualicemos la portada sin aplicar el estilo de jQuery Mobile.

Configuración de una WebApp con jQuery Mobile

Para empezar a utilizar jQuery Mobile, realizaremos, a continuación, un nuevo proyecto. Para estructurarlo correctamente, creamos una carpeta llamada **Ejemplos Jquery Mobile** y, dentro de esta, descomprimimos el contenido del **archivo .ZIP** descargado. Luego renombramos la carpeta de jQuery Mobile como **jqm**.

A continuación, en la carpeta raíz de nuestro ejemplo, creamos una página llamada **index.html** y dentro de ella escribimos el siguiente código:

```
<!DOCTYPE html>
<html>
<head>
   <title>Ejemplos con Jquery Mobile</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="jqm/jquery.mobile.structure-1.3.2.min.css" />
   <link rel="stylesheet" href="jqm/jquery.mobile-1.3.2.min.css" />
   <script src="jqm/jquery.mobile-1.3.2.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
</head>
<body>
<h1>Esto es jQuery Mobile</h1>
<p>Mi primera página web con JQM</p>
</body>
</html>
```

COMPATIBILIDAD CON EL NAVEGADOR

jQuery Mobile nos garantiza, en cada nueva versión, que todos los desarrollos que usan este framework puedan visualizar una interfaz uniforme en cualquier plataforma móvil, dejando que CSS y JS ajusten las funcionalidades necesarias para que la interfaz UX sea transparente al usuario, independientemente del dispositivo donde se ejecute.

Si visualizamos este ejemplo en el navegador, no encontraremos nada distinto en esta página web. Para poder ver un verdadero cambio en el diseño de un sitio web con jQuery Mobile, debemos comenzar a utilizar algunos de sus componentes.

Estructura y widgets

jQuery Mobile basa su estructura en el *Responsive Web Design*, por lo cual aplica, mediante estilos CSS, un diseño específico para las diferentes secciones que componen una página web. Veamos, a continuación, los distintos componentes que podemos integrar en un diseño aplicado gracias a este conjunto de librerías. Estos componentes nos permitirán estructurar toda página de jQuery Mobile.

Page

Page nos permite definir la estructura equivalente a **<body>** en un HTML común. Dentro de **page** agruparemos todas las estructuras de encabezado, pie de página, contenido, barras de herramientas y otras funcionalidades que JQM nos da.

Page no reemplaza a la etiqueta **<body>** dentro de un documento HTML; por el contrario, se deben utilizar los elementos **page** dentro de esta etiqueta. Veamos un ejemplo a continuación:

```
<html>
<head>
...
</head>
<body>
        <div data-role="page" id="index">
                <h1>Esto es jQuery Mobile</h1>
                <p>Mi primera página web con JQM</p>
    </div>
</body>
</html>
```

Al desarrollar este ejemplo, recordemos que dentro del tag **<head>** debemos declarar la ruta de las librerías de JQM. Si lo ejecutamos en un navegador web, no veremos grandes cambios, ya que solo definimos el estilo **page**. Para poder ver cambios en una página web, debemos utilizar mínimamente las estructuras **header**, **content** y **footer**.

DENTRO DEL TAG HEAD DEBEMOS DECLARAR LA RUTA DE LAS LIBRERÍAS DE JQUERY MOBILE

Header

La etiqueta **<header>** no es más que un encabezado que tendrá nuestra página. En ella podemos indicar el título, incorporar un logo y hasta uno o más botones con funciones de menú.

Veamos un ejemplo donde solo incluimos el título:

```
...
<body>
        <div data-role="page" id="index">
                <div data-role="header">
    <h1>Esto es Jquery Mobile</h1>
  </div>
                <p>Mi primera página web con JQM</p>
</div>
</body>
...
```

Content

La etiqueta **<content>** nos permitirá incorporar diversos contenidos en la página creada con jQuery Mobile. Equivale al cuerpo principal de un HTML, y solo se puede utilizar una etiqueta **content** dentro de una etiqueta **page**.

```
...
<body>
        <div data-role="page" id="index">
```

```
                <div data-role="header">
    <h1>Esto es JQuery Mobile</h1>
  </div>
                <div data-role="content">
    <p>Mi primera página web con JQM</p>
  </div>
</div>
</body>
...
```

Footer

La etiqueta **<footer>** nos permite incorporar un pie de página al contenido creado con jQuery Mobile. Podemos especificar, por ejemplo, el copyright de la página web o aplicación, incluir alguna imagen referente o publicidades, o bien, simplemente, estructurar una barra de herramientas.

```
...
<body>
    <div data-role="page" id="index">
                <div data-role="header">
    <h1>Esto es JQuery Mobile</h1>
  </div>
                <div data-role="content">
    <p>Mi primera página web con JQM</p>
  </div>
  <div data-role="footer">
    <p>Copyright 2013: Fernando Luna</p>
  </div>
</div>
</body>
...
```

Para que nuestra página no visualice de manera contigua las tres secciones utilizadas, debemos agregar a la sección **footer** la sentencia **data-position="fixed"**, forzando así a esta sección a ubicarse en el pie de página del browser.

El código de **footer** debe quedar de la siguiente manera:

```
...
    <div data-role="footer" data-position="fixed">
      <h6>Copyright 2013: Fernando Luna</h6>
    </div>
...
```

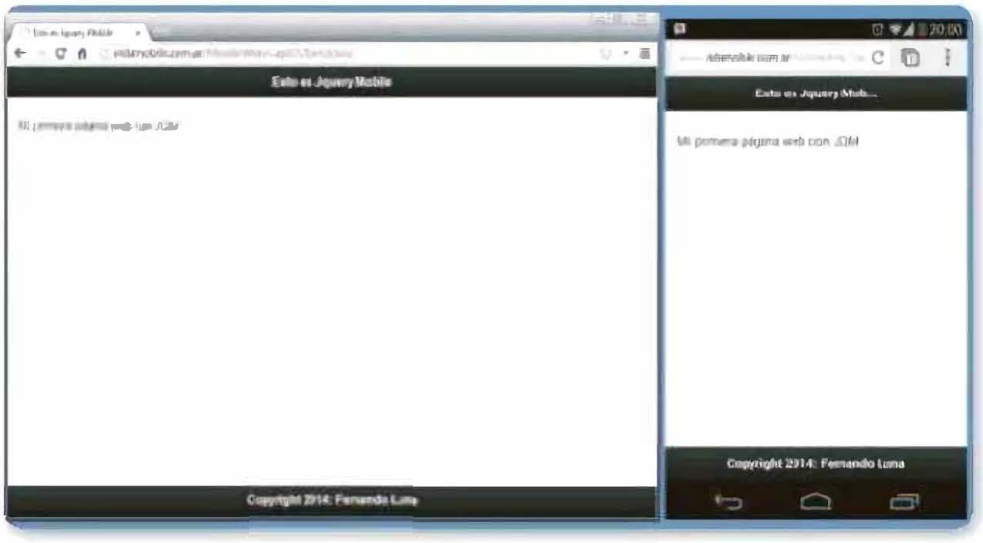


Figura 3. jQuery Mobile representa la estética de una WebApp de la misma forma, tanto en un navegador de escritorio como en un dispositivo móvil.

Navigation Bar

A través de la etiqueta **<navigation>** podemos crear una **barra de herramientas** o **barra de navegación** en nuestra WebApp. Esta etiqueta permite definir, en su interior, botones combinados con texto, icono o ambos.

Esta etiqueta suele ubicarse dentro de las etiquetas **header** o **footer** de la WebApp, para poder tener la pantalla correctamente estructurada. Sin importar su ubicación, cuando creamos una barra de herramientas, esta agrupa los botones en una sola línea, distribuyéndolos de manera equitativa a lo largo de la pantalla.

El máximo de botones permitidos en una sola línea es cinco. De tener que superar esta cantidad, de forma automática los botones se distribuirán en dos líneas.

Veamos un ejemplo de **navigation**:

```
...
    <div data-role="header">
      <H1>Navigation Bar</H1>
      <div data-role="navbar">
        <ul>
          <li><a href="#" data-icon="arrow-l">Atrás</a></li>
          <li><a href="#" data-icon="arrow-r">Adelante</a></li>
          <li><a href="#" data-icon="refresh">Recargar</a></li>
          <li><a href="#" data-icon="star">Favorito</a> </li>
        </ul>
      </div>
    </div>
...
```

Obtendremos, a continuación, una barra de herramientas similar a la de los navegadores web para smartphones. En los siguientes capítulos, veremos más opciones de personalización de esta etiqueta.

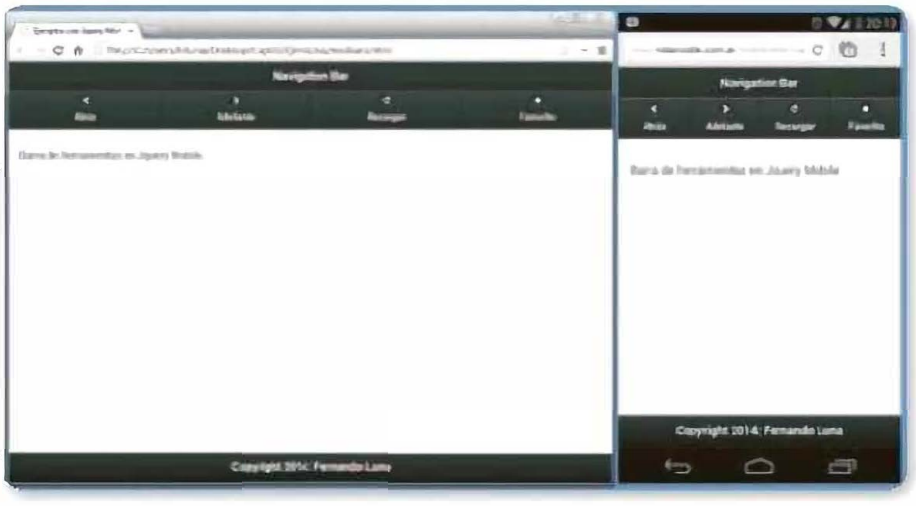


Figura 4. La creación de una barra de herramientas, ya sea en el extremo superior o inferior de la pantalla, es posible gracias a **`Navigation Bar`**.

Transitions

Dentro de jQuery Mobile, los vínculos establecidos hacia otras páginas (o pages) se pueden realizar mediante animaciones

preestablecidas dentro de este framework: a estas se las denomina **transiciones**. Actualmente, existe una serie de transiciones que pueden aprovecharse, que fueron bautizadas como **fade**, **pop**, **flip**, **slide**, **slideUp** y **slideDown**. Se invocan dentro de un link de la siguiente manera:

```
<a href="contacto.html" data-role="button" data-
transition="fade">Contáctenos</a>
```

La transición **slide** genera un efecto de deslizamiento de la página hacia uno de los extremos de la pantalla. La transición **flip** rota la página sobre su eje, permitiendo la entrada de la nueva página cuando finaliza la rotación.

Fade genera un efecto similar al desvanecimiento; **pop**, un efecto emergente desde el centro de la página; y **slideUp** y **slideDown** generan un efecto de desplazamiento vertical hacia arriba y hacia abajo, respectivamente.

> SLIDEUP Y SLIDEDOWN GENERAN EFECTOS DE DESPLAZAMIENTO VERTICAL

No existen restricciones para el uso de las transiciones: desde cualquier widget o texto que disponga un link, estas pueden ser aplicadas sin problema.

La única restricción que puede encontrarse en el uso de las transiciones es que el navegador web del dispositivo móvil no soporte la característica de transición, la cual es propia de los **efectos CSS**. Otra restricción en las transiciones puede encontrarse cuando, por ejemplo, en Android o iOS, el usuario desactiva los efectos de animación de las aplicaciones, menús, etcétera, desde el sistema operativo para ahorrar batería.

FUNCIONAMIENTO DE TRANSICIONES

Al momento de escribir este capítulo, jQuery Mobile informa que aún se encuentran trabajando en una solución para que las transiciones funcionen en todas las plataformas. Dado que se basan en efectos CSS, las transiciones quedan limitadas a la implementación de esta tecnología en todos los navegadores web.

Esta funcionalidad también causará que las transiciones se vean de manera intermitente o que directamente no se reproduzcan en el dispositivo del usuario.

Dialog Page

Esta etiqueta permite que una página definida como **page** con jQuery Mobile o cualquier otra página remota pueda ser visualizada mediante un diálogo. Esto genera una ventana tipo modal, que aparece suspendida por encima de la página desde donde la invocamos.

Para realizar esto, simplemente debemos agregarle un atributo a los hipervínculos creados con JQM.

Veamos a continuación un ejemplo:

```
<a href="gracias.html" data-rel="dialog" data-role="button" data-
theme="b">Enviar</a>
```

Close Dialog

Las páginas visualizadas con el atributo **data-rel="dialog"** crean, de manera predeterminada, un botón superior que permite cerrar la página modal. Este tipo de ventanas generalmente se utiliza para visualizar un formulario, una ventana del estilo **"Acerca de"** y, también, ciertos mensajes de diálogo que interactúan con el usuario.

Si en nuestra WebApp incluimos formularios modales, lo más común es darle al usuario la posibilidad de que interactúe a través de los botones de opciones del cuadro de diálogo, con lo cual el botón **cerrar** que genera la etiqueta **Dialog Page** no nos será útil. Podemos ocultarlo utilizando el siguiente atributo dentro de la invocación a **Dialog Page**:

```
data-close-btn="none"
```

También podemos cambiar la ubicación del botón **cerrar**, el cual originalmente se sitúa en el extremo izquierdo de **Dialog Page**. Simplemente debemos agregar el siguiente atributo:

```
data-close-btn="right"
```

Estilos en Dialog Page

Por defecto, la ventana de diálogo se presenta con bordes redondeados. Este estilo viene aplicado desde la librería de jQuery Mobile. Si deseamos eliminarlo, para que la ventana se presente en su forma natural, debemos utilizar el siguiente código:

```
data-corners="none"
```

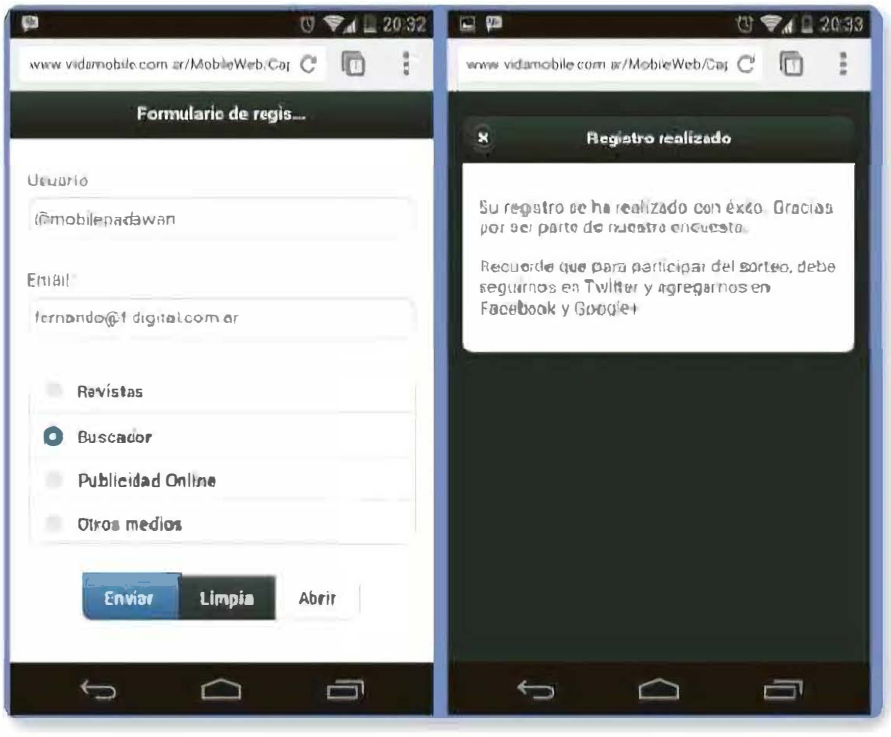


Figura 5. `Dialog Page` nos simplifica la visualización de formularios modal o cuadros de diálogo, basándose en un simple HTML de jQuery Mobile.

Ejercicio práctico

Para poder consolidar los ejemplos hasta aquí repasados, realizaremos a continuación un ejercicio integrador. Este ejercicio nos mostrará cómo integrar las funciones de jQuery Mobile de manera óptima y sin necesidad de llegar a elaborar un desarrollo complejo.

El objetivo es desarrollar un sitio web corporativo. Este presentará una empresa determinada, contará su historia y los productos que

DESARROLLAREMOS CON JQM UTILIZANDO PAGE, HEADER, NAVIGATION BAR, CONTENT Y FOOTER

comercializa. Además, nos permitirá conocer la ubicación de su planta de elaboración y nos ayudará a contactar a la empresa mediante el envío de un mensaje de correo electrónico.

Las funcionalidades básicas que tendrá la página web que elaboremos nos permitirán realizar un desarrollo con jQuery Mobile utilizando los elementos **page**, **header**, **navigation bar**, **content** y **footer**. Podremos también aplicar transiciones entre páginas e incorporar imágenes y texto distribuidos estratégicamente por la página para su correcta visualización en dispositivos móviles.

Por último, realizaremos un formulario de contacto que invocará a la aplicación de correo electrónico instalada en el dispositivo móvil, para que envíe el mensaje de contacto.

Crear la estructura HTML

Descarguemos los archivos necesarios para desarrollar nuestro ejemplo del siguiente link: **https://premium.redusers.com**. Dentro del material, encontraremos un archivo denominado **base.html**, que nos servirá de punto de partida para crear nuestra web.

Crearemos, a continuación, la estructura del sitio web. Deberá tener una carpeta contenedora a la que bautizaremos **greenberries**. Dentro de esta, copiamos el archivo **base.html** y lo renombramos como **index.html**. Luego, creamos la carpeta **imagenes** y, dentro de esta, copiamos los archivos de imágenes que hemos descargado.

Ahora nos queda agregar el siguiente contenido dentro del tag **<body>** y **</body>**:

```
<div data-role="page" id="index" data-theme="d">
  <div data-role="header" data-theme="c" align="center">
      <img src="imagenes/Logo800x150.gif" style="width:80%;max-
width:400px" alt="Greenberries Logo" longdesc="Greenberries Farming Logo">
  </div>
  <div data-role="content" data-theme="d">
      <p>Bienvenido a <b>Green&Berries Farming</b>.</p>
```

```
            <p>Nuestra compa&ntilde;&iacute;a se especializa en la
plantaci&oacute;n, cosecha y distribuci&oacute;n por mayor y menor de frutos rojos
y productos relacionados.</p>
            <p></p>
            <p>A diferencia de otras firmas productoras, Green&Berries posee dos
líneas de productos: Frutos rojos org&aacute;nicos, sembrados y cosechados bajo los
principales est&aacute;ndares de calidad, y Frutos rojos hidrop&oacute;nicos, que
cumplen con su etiqueta org&aacute;nica, pero con la diferencia que estos son semb-
rados y desarrollados &iacute;ntegramente bajo el sistema de hidroponia.</p>
        </div>
        <div data-role="footer" data-theme="c" data-position="fixed">
            <h5> Copyright Vida Mobile</h5>
        </div>
    </div>
```

Iniciamos la estructura de JQM con el tag **<Page>**. Como **id**, lo bautizamos **index**, igual que la página inicial, y le aplicamos el tema **"d"**, propio de JQM. Dentro de este tag, abrimos otro tag **<header>**, donde reemplazamos el título de encabezado por una imagen con el logo de la compañía. Este tag posee como atributos destacables el tema **"c"** y la alineación **"center"**.

El tag **<img>** de logo engloba, dentro de las características más importantes, un ancho que no supera el 80 % del tamaño de la página y un ancho máximo no mayor a 400 pixeles. Como hereda la alineación establecida en **<header>**, por defecto estará centrado.

Cerrado el tag **</header>**, iniciamos un tag **<content>** donde aplicamos un tema **"d"**. Dentro de este, escribimos la presentación del sitio corporativo, enmarcando el texto dentro del tag ** ** y ** ** (paragraph). Podemos destacar párrafos utilizando negrita o cursiva (****, ****), entre otros estilos.

Establecer una barra de herramientas

A continuación, para poder navegar de manera óptima por el sitio web móvil, incorporaremos una barra de herramientas. Esta barra estará dividida en cuatro secciones, a saber:

- Historia
- Productos
- Showroom
- Contacto

Para no sobrecargar el sitio web móvil, descartaremos el uso de la etiqueta **<footer>**. En su lugar, situaremos la barra de herramientas. Al finalizar el div **<content>**, iniciamos un nuevo div del tipo **Navigation Bar**. Para ello, escribimos el siguiente código:

```
<div data-role="navbar" data-position="fixed">
 <ul>
  <li><a href="index.html" data-icon="star" class="ui-disabled">Historia</a></li>
  <li><a href="productos.html" data-icon="check">Productos</a></li>
  <li><a href="showroom.html" data-icon="search">Showroom</a></li>
  <li><a href="contacto.html" data-icon="info">Contacto</a> </li>
 </ul>
</div>
```

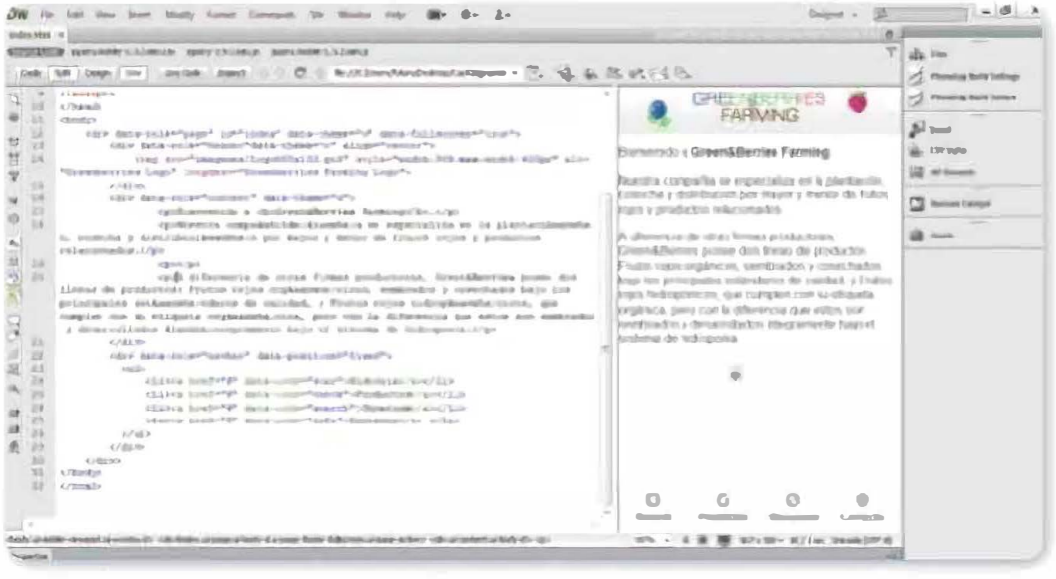


Figura 6. El desarrollo en Dreamweaver nos permite, a través de la opción **Live**, ver en vivo el resultado de nuestro sitio web móvil.

De los cuatro botones que agregamos en la barra de herramientas, podemos notar que **Historia** es diferente a los demás. Este posee el

atributo **class="ui-disabled"**, el cual desactiva el botón simplemente porque el usuario se encuentra en la sección correspondiente a este.

Por cada nueva pantalla que tenga nuestra web mobile, agregaremos esta característica al botón correspondiente a dicha sección.

Figura 7. La portada principal de nuestro proyecto ya ha tomado forma.

Crear la sección Productos

Esta sección mostrará una introducción a los productos que comercializa la empresa y agregará dos imágenes: una por cada categoría de producto. Cuando el usuario haga clic en la categoría de su interés, se abrirá una nueva página modal que visualizará los productos correspondientes. Para ello, utilizaremos **Dialog Page**. Tomamos nuevamente el archivo **base.html**, lo copiamos y renombramos como **productos.html**. Agregamos la barra de herramientas inferior, y seteamos el botón **Productos** con el atributo **ui-disabled**. Agregamos, luego, el código correspondiente al **<body>:**

PARA CREAR LA SECCIÓN DE PRODUCTOS DE LA EMPRESA USAREMOS DIALOG PAGE

```
<div data-role="page" id="productos" data-theme="d">
  <div data-role="header" data-theme="c" align="center">
    <img src="imagenes/Logo800x150.gif" style="width:80%;max-
width:400px" alt="Greenberries Logo" longdesc="Greenberries Farming Logo">
  </div>
  <div data-role="content" data-theme="d">
    <h2>Productos</h2>
    <p>A continuaci&oacute;n puede conocer los productos que comercializa-
mos. Haga clic sobre la categor&iacute;a de su inter&eacute;s.</p>
    <div id="imagenes-links" align="center">
      <a href="organicos.html" data-rel="dialog"><img src="imagenes/
productos-organicos.jpg" Style="width:100%;min-width:150px;max-
width:600px;border:1px" /></a><p></p>
      <a href="hidroponicos.html" data-rel="dialog"><img src="imagenes/
productos-hidroponicos.jpg" Style="width:100%;min-width:150px;max-
width:600px;border:1px" /></a>
    </div>
  <div data-role="navbar" data-position="fixed">
   <ul>
    <li><a href="index.html" data-icon="star">Historia</a></li>
    <li><a href="#" data-icon="check" class="ui-disabled">Productos</a></li>
    <li><a href="showroom.html" data-icon="search">Showroom</a></li>
    <li><a href="contacto.html" data-icon="info">Contacto</a> </li>
   </ul>
```

```
    </div>
</div>
```

En **productos.html** agregamos, en la sección **<content>**, un texto simple seguido de dos imágenes, una por cada categoría de producto que la empresa maneja, que, a su vez, ofician de hipervínculo.

Al hacer clic sobre estas imágenes, se abrirá un pop-up (ventana modal), que visualizará la información correspondiente a este segmento de producto.

Figura 8. Desde la página **productos.html** podemos acceder a las distintas categorías que esta web ofrece.

Veamos ahora al código de la página **organicos.html**, tomando nuevamente como código inicial la página **base.html**:

```
<div data-role="page" id="organicos" data-theme="c">
    <div data-role="header"data-theme="c" align="center">
        <h4>Frutos org&aacute;nicos</h4>
    </div>
    <div data-role="content" data-theme="d" >
```

```
<div id="img-logo" align="center">
  <img src="imagenes/p-organic-200x200.jpg" />
</div>
<p>Conservamos la calidad de nuestros productos trat&aacute;ndolos con
t&eacute;cnicas que garantizan su frescura y sabor, sin que sean alterados por el-
ementos qu&iacute;micos.</p>
<p>Actualmente comercializamos: <strong>Frutillas, Cerezas, Ciruelas,
Ar&aacute;ndanos, Corintos, Frambuesas, Moras coloradas y Moras negras.</
strong><p>
</div>
```

La página de productos orgánicos se visualizará en forma modal:

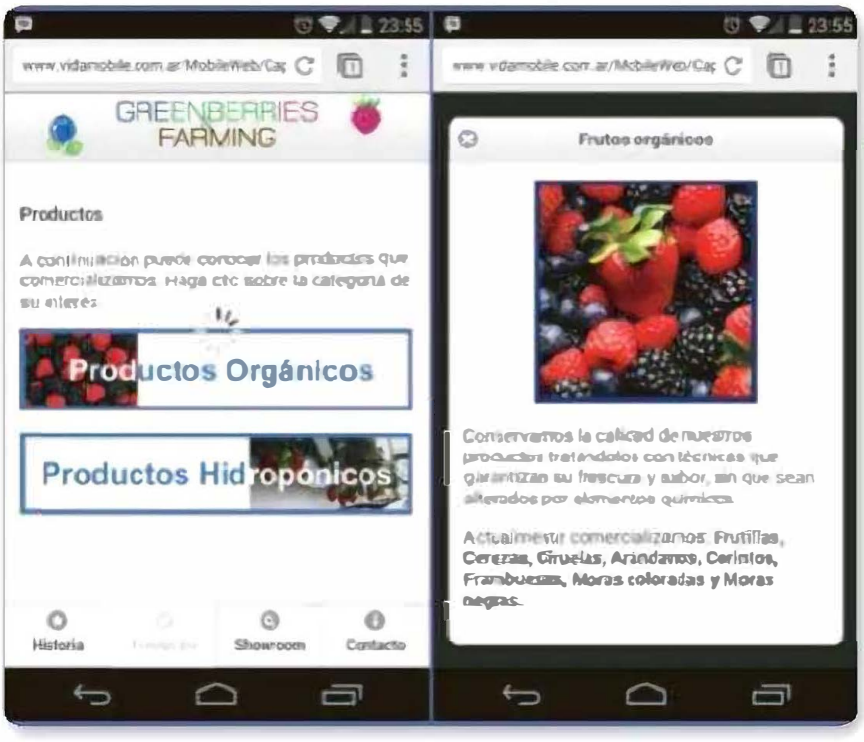


Figura 9. Al hacer un tap sobre el banner **Productos orgánicos** se despliega la página **`organicos.html`** en modo pop-up.

Ahora repetimos este último paso, creando la página **hidroponicos.html**. A continuación, el código de esta nueva página:

```
<div data-role="page" id="hidroponicos" data-theme="c">
  <div data-role="header"data-theme="c" align="center">
```

```
        <h4>Frutos hidrop&oacute;nicos</h4>
      </div>
      <div data-role="content" data-theme="d" >
        <div id="img-logo" align="center">
          <img src="imagenes/p-hydroponic-200x200.jpg" />
        </div>
        <p>Disponemos de la misma variante de frutos rojos org&aacute;nicos desarrollados bajo el sistema de hydroponia, tratado con sustratos org&aacute;nicos creado a base de extractos de desechos de frutas y verduras org&aacute;nicas.</p>
        <p>Estos productos est&aacute;n disponibles todo el a&ntilde;o, ya que son desarrollados bajo un microclima especialmente dise&ntilde;ado para este prop&oacute;sito.</p><p>
    </div>
```

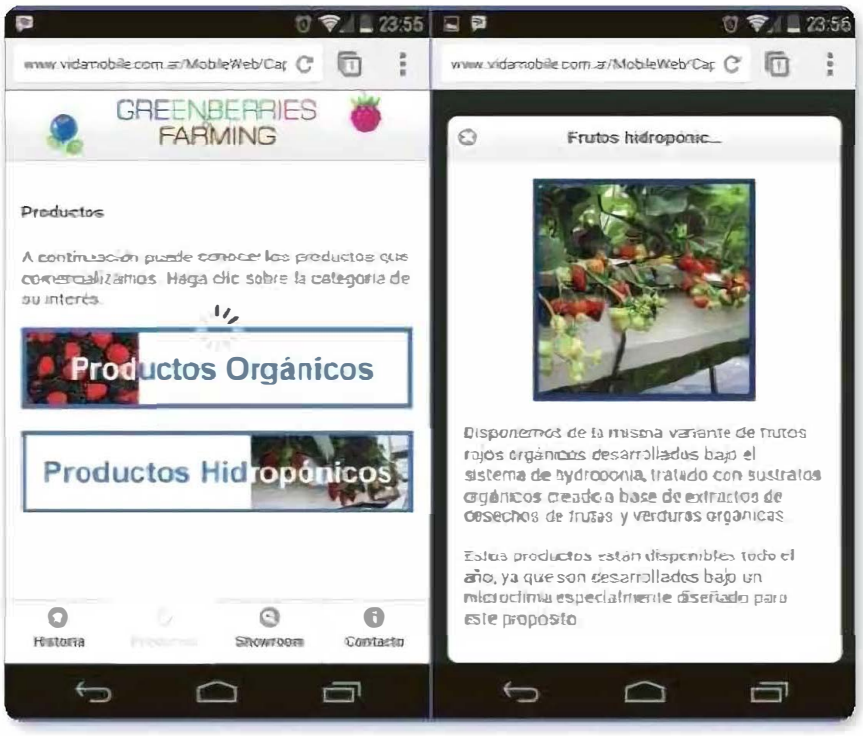


Figura 10. Al hacer un tap sobre el banner **Productos hidropónicos** se despliega la página **`hidroponicos.html`** en modo Popup.

Crear la sección Showroom

Ya entramos en la recta final de nuestro primer sitio web móvil. Crearemos, a continuación, la anteúltima sección, **Showroom**, donde

utilizaremos un mapa estático de Google Maps con una dirección ficticia de la planta de atención al público.

Replicamos los pasos para la creación de un nuevo archivo HTML, al cual llamaremos **showroom.html**. En él, completamos el código correspondiente a jQuery Mobile, con lo siguiente:

```
<div data-role="page" id="productos" data-theme="d">
    <div data-role="header"data-theme="c" align="center">
        <img src="imagenes/Logo800x150.gif" style="width:80%;max-
width:400px" alt="Greenberries Logo" longdesc="Greenberries Farming Logo">
    </div>
    <div data-role="content" data-theme="d">
        <h4>Atenci&oacute;n al p&uacute;blico</h4>
        <div id="mapa" align="center">
            <img border="2" style='border-color:#69C' src="http://maps.
googleapis.com/maps/api/staticmap?center=-34.618747,-58.842229&zoom=16&
size=350x200&markers=size:mid%7Ccolor:blue%7Clabel:G%7C-34.618747,-
58.842229&sensor=true" />
        </div>
    <p>Lo invitamos a conocer nuestro Showroom ubicado en <strong>Autopista
del Oeste, KM. 44.500, La Reja, Buenos Aires</strong>.</p>
    <p>Aqu&iacute; encontrar&aacute; todos nuestros productos envasados y listos
para llevar. Comun&iacute;quese previamente por tel&eacute;fono para coordinar
una cita: <strong>02321-234-9876</strong></p>
    </div>
    <div data-role="navbar" data-position="fixed">
     <ul>
        <li><a href="index.html" data-icon="star">Historia</a></li>
        <li><a href="productos.html" data-icon="check">Productos</a></li>
        <li><a href="#" data-icon="search"  class="ui-disabled">Showroom</
a></li>
        <li><a href="contacto.html" data-icon="info">Contacto</a> </li>
     </ul>
    </div>
</div>
```

En esta nueva página, incluimos el uso de **Static Maps**, los mapas estáticos de **Google Maps**. Lo incluimos dentro de un tag **<img>**,

indicando en el atributo **src=""** la URL correspondiente al mapa. Si analizamos el código de esta URL, podemos encontrar el parámetro **Center=**, seguido de las coordenadas de **latitud** y **longitud** que sitúan el mapa en el punto representado.

Figura 11. Los mapas estáticos de Google Maps nos permiten mostrar una ubicación utilizando este recurso de forma gratuita.

El parámetro **zoom**, tal como su nombre lo indica, aplica la distancia a representar en el mapa. **Size** recibe como parámetro el ancho por alto que tendrá el mapa (está limitado a 600 x 400 pixeles para los mapas estáticos). **Markers** permite establecer uno o más marcadores, a los que les podemos indicar el **color**, una **inicial** que puede contener en el interior del globo y las **coordenadas** donde se ubicarán.

> EL PARÁMETRO ZOOM APLICA LA DISTANCIA A REPRESENTAR EN EL MAPA

Crear la sección Contacto

La última sección que dispondrá nuestro sitio web móvil, es la de **contacto**. En ella,

UTILIZAREMOS EL CORREO ELECTRÓNICO QUE EL USUARIO TIENE CONFIGURADO EN SU DISPOSITIVO

estableceremos un formulario que podrá completar el cliente, para luego enviarlo por e-mail. La idea es utilizar el correo electrónico que el usuario tiene configurado en su dispositivo, independientemente de cuál sea este. Esto nos da la ventaja de que el usuario tendrá una copia del mensaje almacenada en sus elementos enviados, y que nuestra WebApp no requerirá procesar el envío de un e-mail a través de un sistema propio interno.

El formulario de la página **Contacto** constará de los campos **E-mail**, **Asunto**, **Mensaje**. En el campo **E-mail**, el usuario que desea enviar un mensaje de consulta debe ingresar su correo electrónico. Esto nos servirá para que el usuario reciba una copia del mensaje enviado. El e-mail de la empresa irá embebido dentro de la página web y solo será visible cuando se cargue el mensaje en el cliente de correo por defecto.

El campo **Asunto** y el campo **Mensaje** se cargarán de manera automática en los respectivos campos del cliente de correo electrónico.

Creemos, a continuación, un nuevo archivo HTML, llamado **contacto.html**, con el siguiente código:

```
<!DOCTYPE html>
<html>
<head>
   <title>Greenberries Farming</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
   <!—Script para enviar correo electrónico-->
</head>
<body>
   <div data-role="page" id="productos" data-theme="d">
```

```
        <div data-role="header"data-theme="c" align="center">
          <img src="imagenes/Logo800x150.gif" style="width:80%;max-
width:400px" alt="Greenberries Logo" longdesc="Greenberries Farming Logo">
        </div>
        <div data-role="content" data-theme="d">
            <h4>Formulario de contacto</h4>
                                    <form id="email" method="put"
action="enviarEmail()" >
                 <input type="email" id="correo" placeholder="su correo
electr&oacute;nico" />
                <input type="text" id="asunto" name="asunto"
placeholder="Asunto" />
                <div class="ui-field-contain">
                                     <textarea name="mensaje" id="mensaje"
placeholder="Mensaje"></textarea>
                <input type="button" id="enviar" value="Enviar"
onclick="sendMail()"/>
              </form>
                 </div>
      </div>
      <div data-role="navbar" data-position="fixed">
          <ul>
           <li><a href="index.html" data-icon="star">Historia</a></li>
           <li><a href="productos.html" data-icon="check">Productos</a></li>
                <li><a href="#" data-icon="search" >Showroom</a></li>
                <li><a href="contacto.html" data-icon="info"  class="ui-
disabled">Contacto</a> </li>
          </ul>
           </div>
      </div>
   </body>
   </html>
```

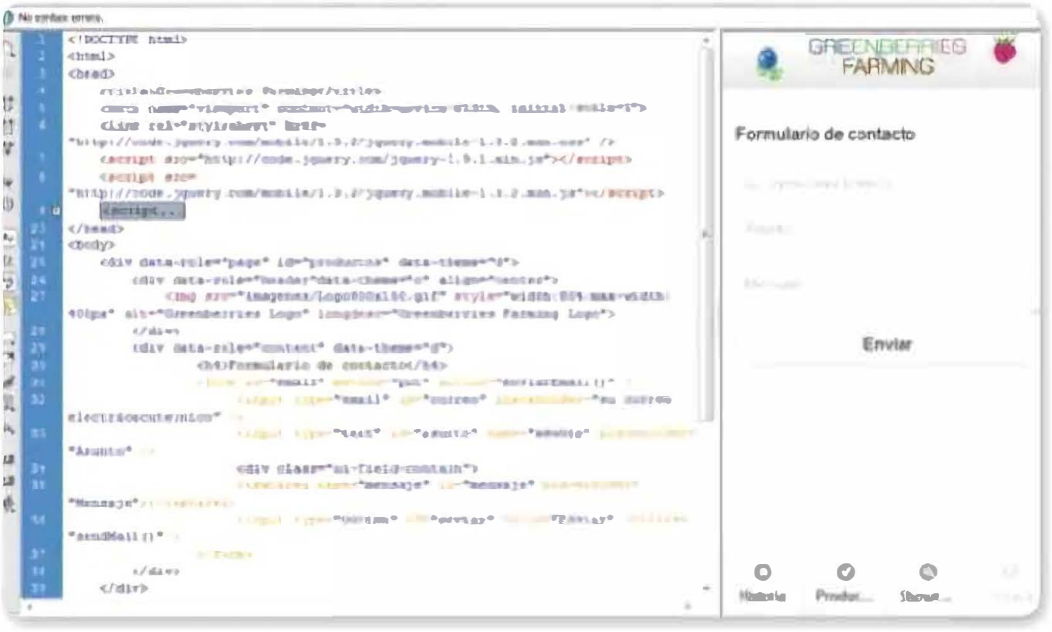


Figura 12. Al igual que con el resto de las páginas, Dreamweaver nos permite obtener una vista previa del formulario de contacto.

Si visualizamos en el navegador la página HTML creada, veremos simplemente un formulario con tres campos. Para el campo **e-mail**, establecemos, en el atributo **placeholder**, el texto **"Su correo electrónico"**. Para los campos **asunto** y **mensaje** establecemos, también en el atributo **placeholder**, los respectivos textos que indican su funcionalidad.

> DADO QUE EN ESTAS PANTALLAS EL ESPACIO NO ABUNDA, DEBEMOS RESERVARLO PARA LOS TEXT INPUT

Dado que estamos elaborando una WebApp ideada específicamente para tablets y smartphones, sabemos bien que el espacio es algo que en las pantallas de estos dispositivos no abunda. Por ello, reemplazamos las etiquetas que identifican a estos campos con el atributo **placeholder**. Con esto nos ahorraremos espacio en pantalla, que quedará disponible para los **Text Input**.

Por último, creamos un botón **Enviar** que invoca la función **sendEmail()**, la cual tendrá la lógica para que se dispare el cliente de correo electrónico que el usuario tiene configurado en su dispositivo. Reemplacemos en el código, a continuación, el texto **<!-- Script para enviar correo electrónico-->** con la siguiente función:

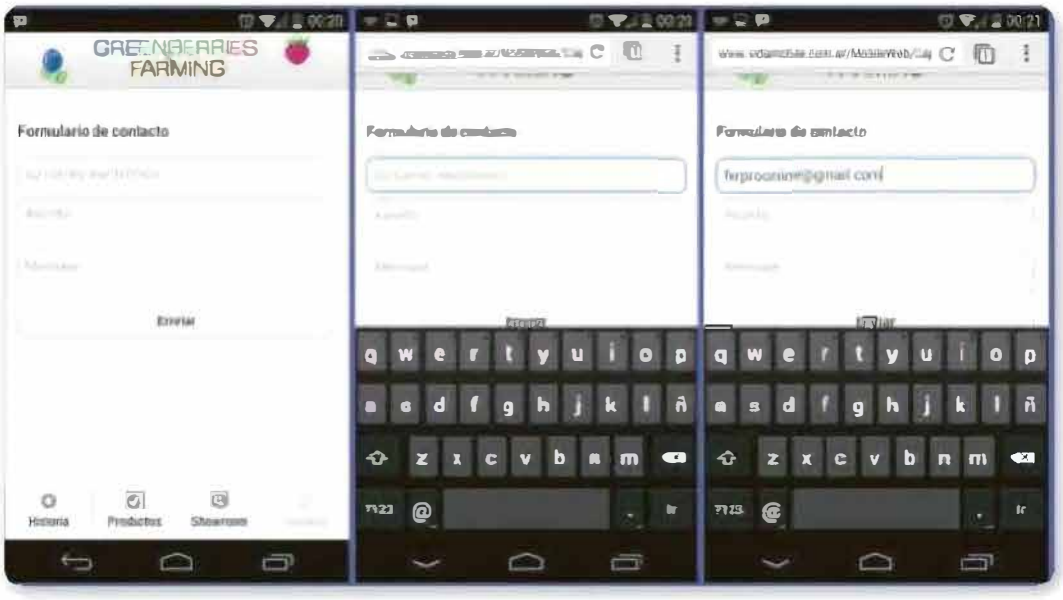


Figura 13. El formulario con el atributo **placeholder** activo y con el foco en el campo e-mail, que activa el teclado correcto. Por último, al escribir la dirección de correo electrónico, se elimina el texto de placeholder.

```
<script type="text/javascript">
        var email = document.getElementById('correo');
        var asunto = document.getElementById('asunto');
        var msj = document.getElementById('mensaje');
        function sendMail() {
                var asunto = document.getElementById("asunto").value;
                var mensaje = document.getElementById("mensaje").
value;
                var link = "mailto:info@greenberriesfarming.com"
                 + "?cc="+email
                 + "&subject="+asunto.replace(" ", "%20")
                 + "&body=" +mensaje.replace(" ","%20")
                window.location.href = link;
                }
</script>
```

Esta función crea una serie de variables en su inicio: **var email**, **var asunto** y **var msj**. Estas variables obtienen, mediante

document.getElementById, el valor de los respectivos campos del formulario. Entre paréntesis, estableceremos qué campos del formulario debe identificar para tomar su valor. Luego armaremos el **string** que ejecutará el **hyperlink mailto:**, existente en HTML desde sus inicios.

AL ARMAR EL STRING UTILIZAMOS LA FUNCIÓN REPLACE(), DE CADA VARIABLE CREADA EN JS

Por último, al armar el string utilizamos la función **replace()** existente, propia de cada variable creada en JavaScript, que reemplaza los espacios que puedan existir en el asunto y cuerpo del mensaje por el char **%20**, equivalente a un espacio en HTML. Así, el string queda confeccionado de la siguiente manera: **mailto:info@greenberriesfarming.com?cc=email&subject=asunto&body=mensaje**.

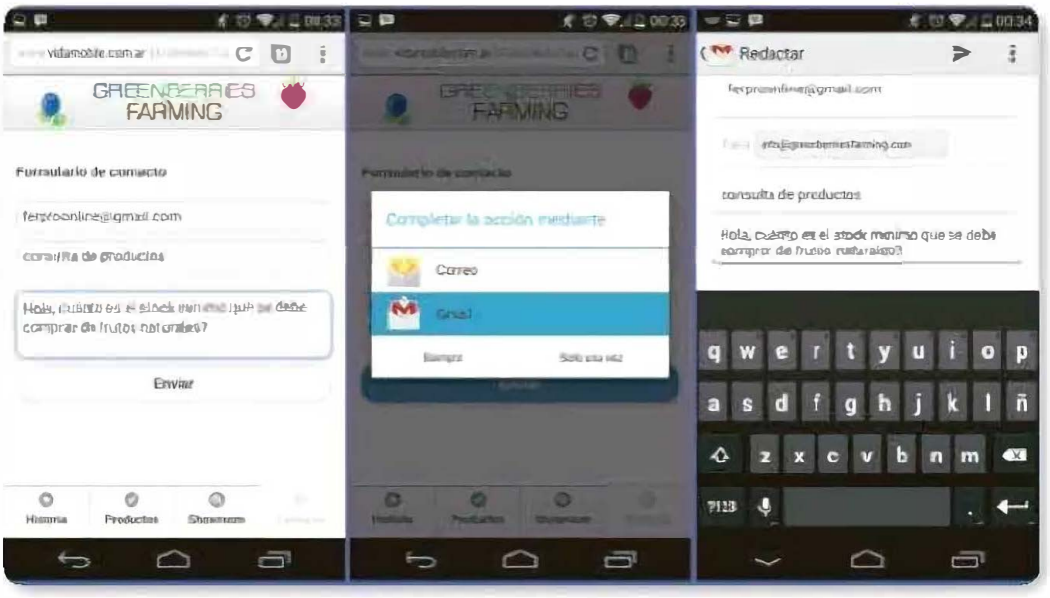


Figura 14. Al presionar el botón **Enviar**, la WebApp ejecutará el cliente de correo por defecto. Si hay más de uno, el usuario deberá elegir. Por último, se cargará el formulario con los campos completos.

Nuestra WebApp se ocupará de invocar el cliente de correo electrónico que el dispositivo tiene configurado. En caso de haber más de uno, el usuario del equipo será quien elegirá a través de cuál se enviará el mensaje.

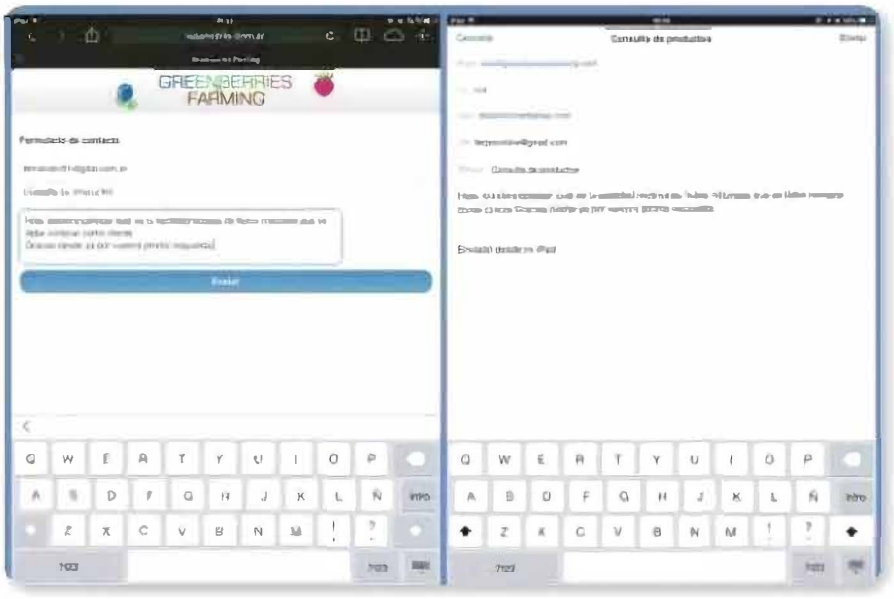


Figura 15. Cualquiera sea el dispositivo, la respuesta de la función **`sendMail()`** se comportará correctamente, cargando los campos automáticamente para luego enviar el mensaje de correo electrónico.

RESUMEN

jQuery Mobile nos abre un abanico de posibilidades para diseñar aplicaciones web móviles como también aplicaciones híbridas. Nos ayuda a mantener la misma estructura en todas las páginas sin importar el navegador web, genera efectos de transición entre páginas e incluso resuelve muchos llamados a funciones de otros lenguajes de manera transparente, gracias a la integración con AJAX que trae. Recomendamos seguir explorando periódicamente la web de jQuery Mobile para enterarnos de las jugosas novedades para implementar en nuestros desarrollos.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es **jQuery Mobile**?
2. ¿Qué librería adicional necesita jQuery Mobile para funcionar?
3. ¿Hay alguna manera de poder utilizar jQuery Mobile sin instalarlo en nuestro servidor?
4. ¿Cómo se compone generalmente una página creada con jQuery Mobile?
5. ¿Cuántos tipos de **input text** podemos identificar?
6. ¿Para qué se utiliza el atributo **data-role="button"**?
7. ¿Qué requiere un **Formulario** para funcionar como tal?
8. ¿Cómo puedo utilizar teclados personalizados en los dispositivos móviles?
9. Indique el tipo de input type que permite ingresar sólo números.
10. ¿Qué uso podemos darle al atributo **data-role="Dialog"**?

EJERCICIOS PRÁCTICOS

1. Desarrolle un sitio web con las siguientes secciones: **Inicio**, **Institucional**, **Servicios**, **Contacto**, donde cada sección sea una página diferente. Luego incluya textos de ejemplo en cada una.
2. Cree un formulario con los tipos de campo **text**, **email**, **password**, **textarea**.
3. Incorpore un botón de enviar y un botón de limpiar campos.
4. Si tiene un sitio web propio, modifique el ejemplo de envío de e-mail con la función de e-mail que le provee su hosting.

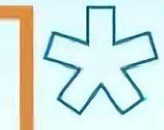
PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*04

Componentes de jQuery Mobile

En el capítulo anterior conocimos las bases necesarias para implementar jQuery Mobile en nuestro desarrollo mobile web. Ahora nos introduciremos aún más en este popular framework para poder sacar provecho al máximo de los componentes prediseñados que JQM nos ofrece, denominados widgets, y así poder explotar su funcionalidad al máximo, estructurando una web móvil de manera tal que nuestros proyectos balanceen armoniosamente su estética y funcionalidad.



Componentes

En jQuery Mobile definimos como componentes a aquellos objetos que conforman el contenido de una página. Este framework nos provee de un set de componentes o **widgets** que podemos utilizar para diversas funcionalidades en esta plataforma.

Entre los widgets más populares podemos destacar los siguientes: **listas**, **viñetas**, **cajas de texto**, **etiquetas**, **botones**, **control calendario** y **slider**. También existen otros componentes que nos permiten darle vida y funcionalidad a una página y que, además, hacen que esta luzca por igual en todas las plataformas donde es visualizada.

Dependiendo de la función o estructura que debamos armar en nuestra web móvil, podremos sacar provecho de cada uno de los componentes de JQM para desarrollar menús funcionales, establecer filtros automáticos en pantallas con muchos datos y aprovechar las diversas variantes que un mismo componente nos ofrece.

A continuación, listaremos los principales componentes que podemos utilizar en jQuery Mobile y veremos un ejemplo de cada uno.

Navigation Bar

En el **Capítulo 3**, repasamos el concepto básico del uso del componente **NavBar**. Vimos cómo crear botones, agregarles un icono o un texto, y cómo estructurar la barra de herramientas en el extremo superior o inferior de la pantalla. Veamos, a continuación, qué otras opciones nos ofrece este maravilloso componente.

Botón activo

En el ejercicio realizado de **Green&Berries Farming** en el capítulo anterior, establecimos que, cada vez que ingresamos a una sección del sitio web móvil, el botón de dicha sección quede inhabilitado: **Disabled**. Esto permite evitar que el usuario pulse nuevamente el botón y que la página se refresque.

En la mayoría de las páginas web de hoy, el **refresco de página** no es un problema cuando navegamos por Wi-Fi o a través de una LAN, pero, cuando el usuario utiliza su dispositivo móvil mediante el pack

de datos 3G o similar, estos datos generalmente tienen un consumo mensual limitado y, por lo tanto, el exceso se le factura aparte al cliente.

Por tal motivo, si podemos evitar el refresco innecesario de una página, no solo ejercemos una buena práctica como programadores, sino que también evitamos la pérdida de datos ingresados en un formulario e impedimos que el usuario consuma innecesariamente de su pack de datos.

> EVITAR EL REFRESCO INNECESARIO DE UN SITIO WEB PREVIENE LA PÉRDIDA DE LOS DATOS INGRESADOS

Otra variante que podemos aplicar en la práctica de navegación mediante un componente **NavBar** es la de marcar el botón correspondiente a la sección en la cual nos encontramos como **botón activo**.

Esto se realiza de la siguiente manera:

```
<div data-role="navbar" data-position="fixed">
    <ul>
     <li><a href="#" data-icon="star" class="ui-btn-active ui-state-
persist">Historia</a></li>
     <li><a href="productos.html" data-icon="check">Productos</a></li>
     <li><a href="#" data-icon="search">Showroom</a></li>
     <li><a href="#" data-icon="info">Contacto</a> </li>
    </ul>
</div>
```

Al cargar la página HTML correspondiente a la dirección que deseamos direccionar, reemplazamos la URL de la página por el carácter **#** y agregamos, en las propiedades del botón, el atributo **class="ui-btn-active ui-state-persist"**, lo que nos permitirá resaltar el botón correspondiente a la sección con otro color.

NAVIGATION BAR Y PANEL

Al momento de realizar una web móvil, podemos establecer como buena práctica la creación de un menú con el widget **Panel**, y de una barra de navegación con el widget **Navigation Bar**, intercalando la visualización del primero si la página web se carga en una tablet, o del segundo, si es vista en un smartphone.

Figura 1. Nuestro proyecto Green&Berries Farming con su Navigation Bar modificada.

Barra de navegación siempre visible

Si probamos nuestro ejemplo anterior desde un teléfono móvil o tablet y pulsamos sobre algún lugar del área de navegación que no posea links, una vez cargada la sección, notaremos que la barra de herramientas se oculta automáticamente. Si volvemos a pulsar en un lugar libre, la barra de herramientas aparecerá otra vez.

Si queremos evitar confusiones a los usuarios del sitio web, podemos hacer que la barra de herramientas quede siempre visible. Para ello, debemos incorporarla a la sección **<Header>** o **<Footer>**, dependiendo de la ubicación que le asignamos en pantalla.

Dado que, en nuestro caso, utilizamos **NavBar** en el pie de página, y no un componente **Footer**, veremos cómo incorporarlo fijando la barra de herramientas allí.

```
<div data-role="footer">
    <div data-role="navbar">
        <ul>
        <li><a href="#" class="ui-btn-active ui-state-persist">Historia</
a></li>
        <li><a href="productos.html">Productos</a></li>
        <li><a href="showroom.html">Showroom</a></li>
        <li><a href="contacto.html">Contacto</a></li>
        </ul>
    </div>
</div>
```

Listas

El componente **List** es un widget que permite formatear una lista desordenada de ítems a través del atributo **data-role="listview"**. El atributo se encarga, a través de la librería JQM, de darle un formato amigable, el cual hará que una simple lista se vea como un listado de ítems profesional.

```
<div data-role="content" data-theme="d">
   <h4 align="center">Top Ten libros de programaci&oacute;n</h4>
   <p></p>
      <ul data-role="listview">
         <li><a href="#">Visual Basic</a></li>
         <li><a href="#">Visual C++</a></li>
         <li><a href="#">Visual C#</a></li>
         <li><a href="#">PHP</a></li>
         <li><a href="#">JavaScript</a></li>
         <li><a href="#">HTML5</a></li>
      </ul>
</div>
```

Reemplazando el carácter # en la sentencia **<a href="#">**, podemos asignarle un **hyperlink** a cada **itemlist** y, así, enviar al usuario hacia otra URL. Si no agregamos nada, el icono **arrow** derecho no aparecerá.

Listas formateadas

A estas simples listas podemos darles un aspecto profesional, aplicándoles un formato para que no muestren solo texto. Veamos las opciones de las que disponemos.

> PARA QUE LAS LISTAS NO MUESTREN SOLO TEXTO, PODEMOS APLICARLES UN FORMATO

Bordes redondeados

A esta lista le podemos aplicar un borde redondeado para que tenga estilo y se despegue del resto del contenido de la página, en caso de que agreguemos más texto e imágenes. Simplemente, debemos agregar el atributo **data-inset="true"**.

```
<ul data-role="listview" data-inset="true">
  <li><a href="#">Visual Basic</a></li>
  <li><a href="#">Visual C++</a></li>
  <li><a href="#">Visual C#</a></li>
  <li><a href="#">PHP</a></li>
  <li><a href="#">JavaScript</a></li>
  <li><a href="#">HTML5</a></li>
</ul>
```

Lista numerada

Los ítems que componen un **Listview** pueden ser numerados de forma consecutiva. Para ello, debemos modificar la apertura de una lista, cambiando su valor original **<ul data-role...>** por **<ol data-role...>**.

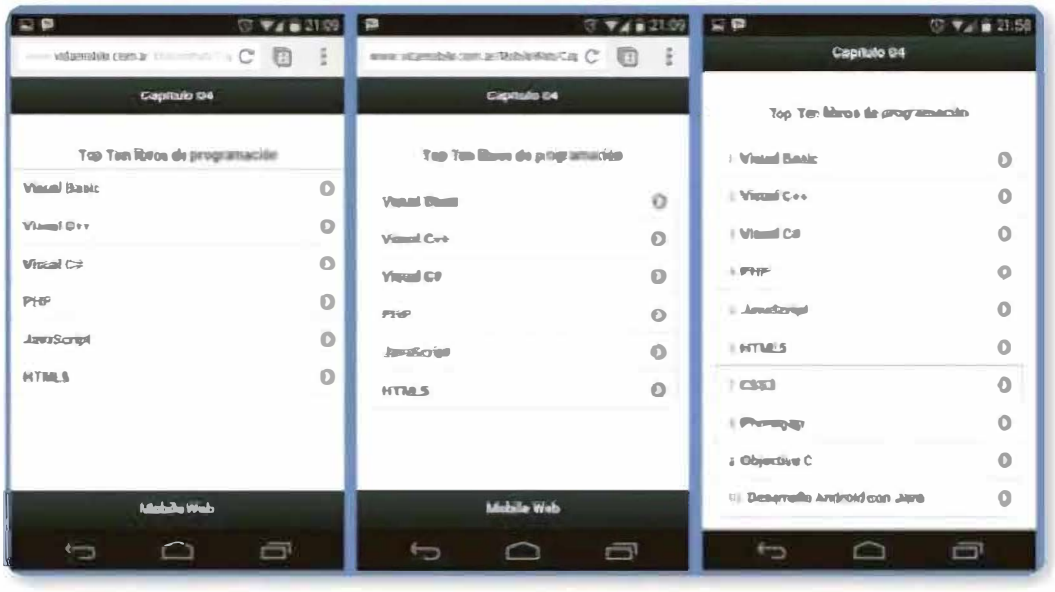


Figura 2. ListView clásico, **ListView formateado** y **lista numerada** son algunos de los estilos que jQuery Mobile nos propone para listar contenido.

A continuación, crearemos un ejemplo de listas numeradas a partir de una serie de libros técnicos:

```
<h3>Top Ten libros de programación</h3>
```

```
<ol data-role="listview">
  <li>Visual Basic</li>
  <li>Visual C++</li>
  <li>Visual C#</li>
  <li>PHP</li>
  <li>JavaScript</li>
  <li>HTML5</li>
  <li>CSS3</li>
  <li>Phonegap</li>
  <li>Objective C</li>
  <li>Desarrollo Android con Java</li>
</ol>
```

Lista dividida

Los **ListItems** pueden agrupar su contenido de una forma específica que podemos dársela nosotros como programadores. Para ello, debemos utilizar **<data-role="listdivider">** en la creación de la lista y especificar el tipo de agrupamiento que vamos a darle.

```
<ul data-role="listview" data-inset="true">
   <li data-role="list-divider">Informática</li>
   <li>Windows 7</li>
   <li>Windows 8</li>
   <li>Ubuntu Linux 14.04</li>
   <li>Mac OS-X</li>
   <li>Microsoft Office 365</li>
   <li data-role="list-divider">Programación</li>
   <li>PHP</li>
   <li>JavaScript</li>
   <li>HTML5</li>
   <li>CSS3</li>
   <li>Phonegap</li>
</ul>
```

Autodividers

Si necesitamos generar un ListView con división automática del contenido para listar un glosario técnico ordenado alfabéticamente y dividido por letra, por ejemplo, podemos utilizar, dentro de **ListView**, el atributo **<data-autodividers="true">**.

```
<ul data-role="listview" data-autodividers="true">
   <li>Append</li>
   <li>As new</li>
   <li>Boolean</li>
   <li>Catch</li>
   <li>Currency</li>
   <li>Decimal</li>
   <li>Dim</li>
   <li>Do Loop</li>
   <li>Double</li>
   <li>else</li>
   <li>else If</li>
   <li>Err</li>
   <li>If</li>
   <li>Integer</li>
   <li>MessageBox</li>
   <li>Private</li>
   ...
</ul>
```

AGRUPAMIENTO DE ÍTEMS

Si el tipo de desarrollo web que realizamos requiere visualizar una lista de ítems extensa, es fundamental establecer un punto de separación entre los ítems, a través del uso de los atributos **Auto Dividers** o **Data Dividers**. De esta forma, la experiencia de usuario al momento de utilizar la WebApp será superior, ya que presentará una mejor visualización de la información mostrada.

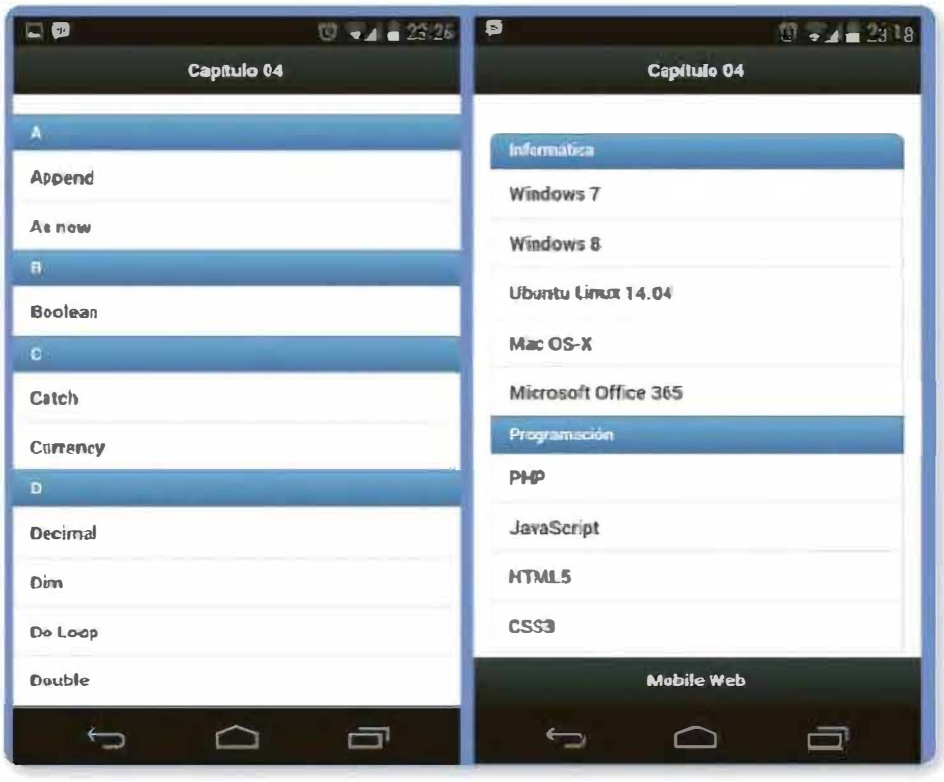


Figura 3. **`Data Dividers`** y **`Auto Dividers`** son dos atributos útiles para el correcto agrupamiento de ítems.

Filtrar ListItems

Puede pasar que utilizando **ListView** se nos presente una lista prolongada de ítems a visualizar y no tengamos manera de agrupar el contenido con **autodividers** o **list-divider**. En este caso, para que el usuario de la aplicación no deba tener que hacer un constante scroll sobre la pantalla, podemos facilitarle la tarea aplicando, en el ListView, el atributo **data-filter="true"**.

Veamos un ejemplo a continuación:

```
<ul data-role="listview" data-filter="true">
  <li>Append</li>
  <li>As new</li>
  <li>Boolean</li>
  <li>Catch</li>
  <li>Currency</li>
  <li>Decimal</li>
```

```
    <li>Dim</li>
    <li>Do Loop</li>
    <li>Double</li>
    <li>else</li>
    <li>else If</li>
    <li>Err</li>
    <li>If</li>
    <li>Integer</li>
    <li>MessageBox</li>
    ...
</ul>
```

El atributo **data-filter** nos permite agregar, en el encabezado del widget ListView, un campo del tipo **<input type search>** que oficiará de campo de búsqueda. Al escribir una o más letras en él, jQuery Mobile aplicará un filtro sobre todos los ListItems coincidentes y sólo mostrará estos en pantalla.

Si queremos eliminar el filtro aplicado, simplemente borramos el texto ingresado o presionamos sobre el botón que aparece sobre el extremo derecho del **input type**, y el texto se eliminará y restablecerá el ListItem con todo su contenido.

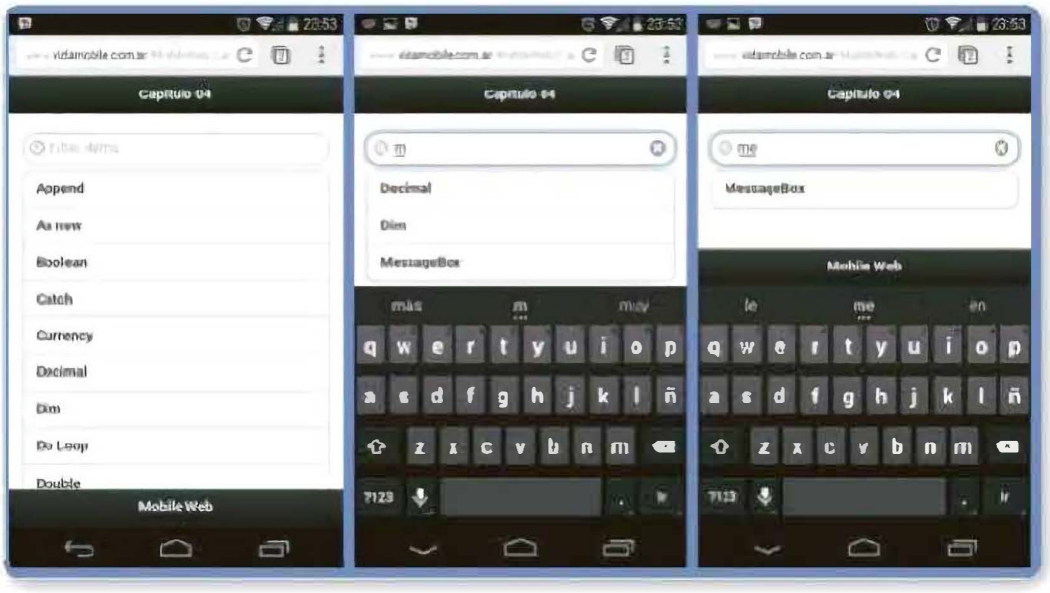


Figura 4. El atributo **data-filter** es una utilidad que simplifica complejas funciones que en otros medios harían lenta la web dinámica.

Buttons

Los botones también son parte de jQuery Mobile. Modificados significativamente si los comparamos con un botón estándar creado con HTML, sirven para un sinfín de situaciones dentro de una WebApp. El widget **Button**, dentro de JQM, se puede utilizar de dos formas diferentes. La primera es invocando una URL determinada o una página web a través de su link:

```
...
<a href="miurl.html" data-role="button">Mi link</a>
...
```

Y la segunda es creando el widget Button, y que este invoque a una función JavaScript:

```
...
<div data-role="button" onclick="miFuncionJS()">Mi link</div>
...
```

Al igual que con el resto de los widgets de jQuery Mobile que repasamos hasta ahora, el objeto button nos permite personalizarlo en base a nuestra necesidad. Veamos, a continuación, qué atributos podemos especificarle.

Deshabilitar botón

Mediante la clase **ui-disabled**, podemos especificar que un widget Button aparezca deshabilitado por defecto. Luego podemos habilitarlo a través de una función o evento creado en JavaScript. Veamos un ejemplo:

```
...
<a href="miurl.html" data-role="button" class="ui-disabled">Mi link</a>
...
```

Si posteriormente deseamos crear una función que indique que, luego de determinada condición, el botón deshabilitado por defecto deba cambiar su estado, tenemos que agregarle un **id** para poder

identificarlo fácilmente desde el código JavaScript. El código quedará de la siguiente manera:

```
...
<a href="miurl.html" data-role="button" id="miButtonLink" class="ui-disabled">Mi link</a>
...
```

MEDIANTE UN ATRIBUTO, PODEMOS MINIMIZAR EL TAMAÑO ESTÁNDAR DE UN BOTÓN

Minimizar el tamaño

En muchos casos, la pantalla de un dispositivo móvil requiere visualizar un importante número de objetos y, dado que estas pantallas son significativamente más pequeñas que las de los monitores, debemos tener atención en la creación de estos objetos. Button nos permite, mediante el atributo **data-mini="true"**, minimizar el tamaño estándar que trae por defecto. Veamos cómo hacerlo en el siguiente código:

```
...
<a href="miurl.html" data-role="button" data-mini="true">Mi link</a>
...
```

Para restaurar su tamaño, simplemente eliminamos este atributo o cambiamos el estado **True** por **False**.

UTILIZACIÓN DE BOTONES EN FORMULARIOS

El uso de botones en un formulario siempre se limita a las funciones de **Enviar**, **Limpiar campos** o **Cancelar**. En este aspecto, deberemos contemplar la agrupación de los botones mediante **data-role="controlgroup"** en las pantallas que son amplias, como las de una tablet. En las pantallas de los teléfonos móviles, es conveniente agruparlos de forma vertical.

Figura 5. Esta página nos muestra el widget **Button** en su formato estándar, deshabilitado y reducido en altura.

Form buttons

Los formularios en HTML son primordiales, en general, para completar datos que luego serán enviados por e-mail o guardados en una base de datos. Si combinamos JQM con un **Form**, debemos tener en cuenta que, al crear el **input type="button"**, este asumirá la estética de los botones de jQuery Mobile.

Si deseamos evitar esto y visualizar un objeto button tal como se ve originalmente en HTML, simplemente debemos agregar el atributo **data-role="none"** en la creación.

```
...
<input type="button" value="Enviar" data-role="none">
...
```

Agregar un icono

Al igual que el widget Navigation Bar, Button soporta la visualización de iconos en su interior. Esto se realiza agregando el atributo **data-icon** cuando creamos el botón. Veamos un ejemplo:

```
...
<a href="favoritos.html" data-role="button" data-icon="star">Favoritos</a>
...
```

Por defecto, el icono **Star**, integrante del set de iconos que viene con la librería JQM, se agregará en el extremo izquierdo del botón. A su vez, podemos combinar y disminuir el tamaño del botón sin que este pierda su icono, combinando ambos atributos en la creación.

```
...
<a href="favoritos.html" data-role="button" data-icon="star" data-
mini="true">Favoritos</a>
...
```

Botones compactos

SI NUESTRA WEBAPP DEBE VISUALIZARSE EN POSICIÓN HORIZONTAL, PODEMOS COMPACTAR EL BOTÓN

El widget **Button** se crea ocupando casi todo el ancho de la pantalla. Así, la visualización en posición vertical no es un problema, dado que dispone de un ancho reducido. Pero si nuestra WebApp debe visualizarse en posición horizontal, estéticamente el botón tendrá un estilo innecesariamente exagerado. Esto podemos solucionarlo con los atributos **mini** e **inline**, los cuales nos permiten crear botones compactos tanto en alto como en ancho, respectivamente.

Veamos un ejemplo:

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

De esta forma, el ancho que tomarán los botones por defecto será el necesario para visualizar el texto que los describe.

Figura 6. Aquí podemos observar el botón normal utilizado en formularios, el botón con icono y el botón con icono compacto.

Posicionamiento del icono

Si no tenemos problemas con la cantidad de componentes en pantalla y deseamos cambiar la ubicación del icono que ilustra el widget **Button**, podemos utilizar el atributo **data-iconpos** en la creación del botón. Veamos un ejemplo:

```
...
<a href="favoritos.html" data-role="button" data-icon="star" data-
iconpos="right">Favoritos</a>
...
<a href="principal.html" data-role="button" data-icon="home" data-
iconpos="left">Principal</a>
...
<a href="productos.html" data-role="button" data-icon="grid" data-
iconpos="top">Productos</a>
...
<a href="contacto.html" data-role="button" data-icon="check" data-
iconpos="bottom">Contacto</a>
...
```

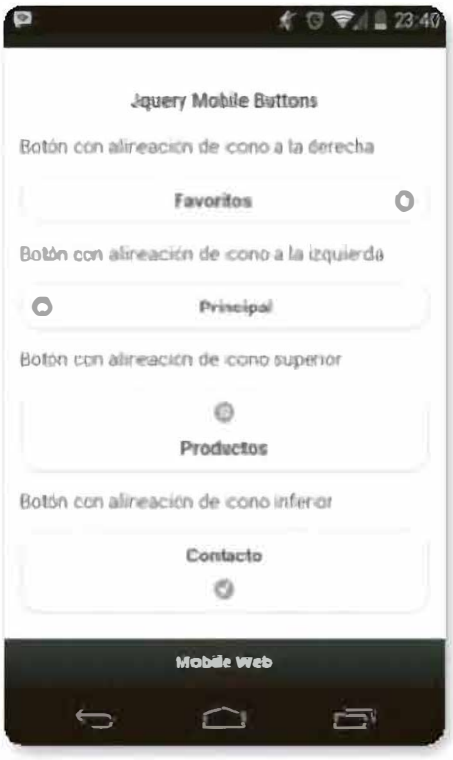


Figura 7. Los widgets **Button** que utilizan iconos pueden ubicarlos en cualquiera de sus lados.

Agrupación de botones

También es posible crear un set de botones agrupados mediante un contenedor que nos permita, tanto vertical como horizontalmente, visualizar un conjunto de botones que conforman un mismo objeto. Veamos un ejemplo:

```
...
<div data-role="controlgroup">
   <a href="acepto.html" data-role="button">Sí</a>
   <a href="noacepto.html" data-role="button">No</a>
   <a href="condicional.html" data-role="button">Tal vez</a>
</div>
...
```

El contenedor **controlgroup** nos permite agrupar un set de controles. Por defecto, su agrupación se realiza de manera vertical, dado que en la mayoría de los dispositivos móviles se encuentra de esta manera. También es posible realizar el agrupamiento de manera horizontal, agregando el atributo **data-type**, como vemos en el siguiente ejemplo:

```
...
<div data-role="controlgroup" data-type="horizontal">
   <a href="acepto.html" data-role="button">Aceptar</a>
   <a href="noacepto.html" data-role="button">Cancelar</a>
   <a href="condicional.html" data-role="button">Ayuda</a>
</div>
...
```

Los atributos presentados hasta aquí permiten reducir el tamaño de un botón, tanto de forma horizontal como vertical. También nos permiten agregar un icono en diferentes posiciones dentro del botón, y hasta agrupar un conjunto de botones gracias al atributo **data-role="controlgroup"**.

Figura 8. Tanto vertical como horizontalmente, agrupar botones nos ayuda a centralizar de forma óptima la pantalla creada.

Text Inputs

En el **Capítulo 3** nos interiorizamos en el uso de Text Input en reemplazo de los campos clásicos utilizados en el viejo HTML. A continuación, repasaremos algunas variantes más de este componente y cómo nos ayudará en el desarrollo de nuestras soluciones.

Vista clásica

Al text input clásico, que se crea simplemente con la sentencia **<input type="text">**, podemos otorgarle un valor agregado, por ejemplo, identificando con un **Label** el tipo de dato que deseamos ingresar. Esto lo hacemos de la siguiente manera:

```
<label for="inputBasico">Ingrese algo:</label>
<input type="text" id="inputBasico" value="" />
```

PODEMOS DARLE MÁS VALOR AL TEXT INPUT, IDENTIFICANDO EL TIPO DE DATO QUE SE INGRESARÁ

Como resultado, obtenemos una etiqueta que describe el valor a ingresar en este campo y, en el segundo renglón, el text input correspondiente. Pero, como bien dijimos en reiteradas oportunidades, en un dispositivo móvil debemos tener cuidado con el espacio a utilizar, dado que las pantallas son reducidas.

De tal manera, podemos recurrir al atributo **data-mini**, el cual nos ayudará a reducir el alto del campo de entrada:

```
<label for="inputBasicoMini">Ingrese algo:</label>
<input type="text" id="inputBasicoMini" value="" data-mini="true" />
```

Field Container

También, si deseamos resumir el **Label** más el **campo** en una sola línea en pantalla, lo podemos realizar utilizando un componente **field container**.

FIELD CONTAINER

El uso de **field container** a través de jQuery Mobile puede aprovecharse siempre en pantallas de amplia visualización, aunque debemos tener en cuenta que, si un formulario contiene demasiados campos, para reducir la invasión de widgets en la pantalla de la WebApp podrá reemplazarse el uso de **Label for** por el atributo **placeholder**.

```
<div data-role="fieldcontain">
   <label for="inputUnaLinea"> Ingrese algo:</label>
   <input type="text" id=" inputUnaLinea " value="" />
</div>
```

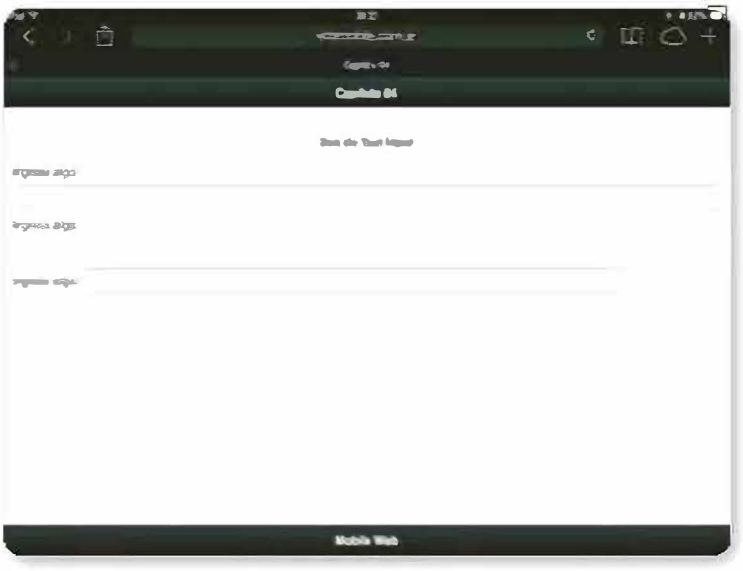


Figura 9. Un **`text input`** normal, uno reducido en altura y el último con su **`Label`** correspondiente en una sola línea.

Password

También podemos crear campos del tipo **contraseña**, indicando en el atributo **Type="password"**.

```
<label for="inputPass"> Ingrese su password:</label>
<input type="password" id="inputPass" value="" />
```

El caracter comodín que visualizará, en lugar de la contraseña, es el utilizado por el sistema operativo donde se ejecuta esta WebApp.

Number

El atributo **type="number"** nos habilita a ingresar solo caracteres utilizando el teclado numérico del dispositivo móvil, aunque debemos aclarar que esto solo funciona en smartphones en modo vertical y no apaisado.

```
<input type="number" id="inputNum" value="" />
```

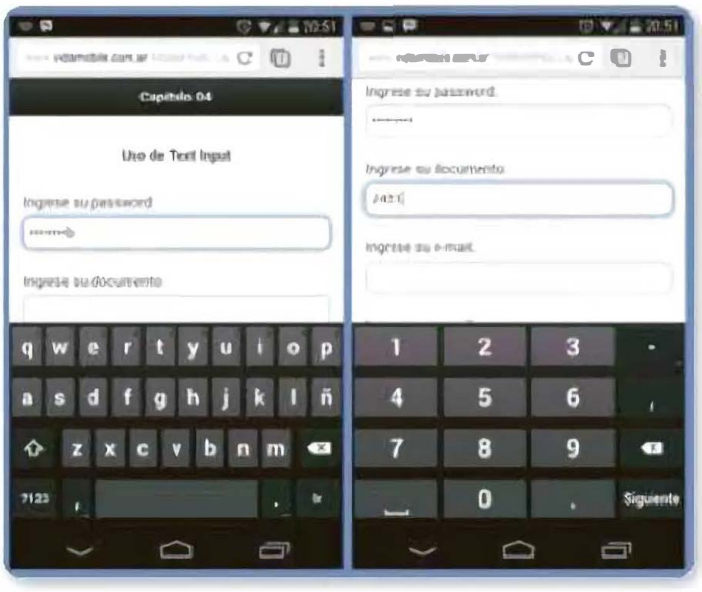


Figura 10. El **`text input password`** y el **`text input number`** cumpliendo las correspondientes funciones asignadas.

E-mail

El atributo **type="email"** activa el formato de teclado para ingresar una dirección de correo electrónico, identificando la tecla @ (arroba) en el teclado común del dispositivo móvil. Funciona en tablets y smartphones.

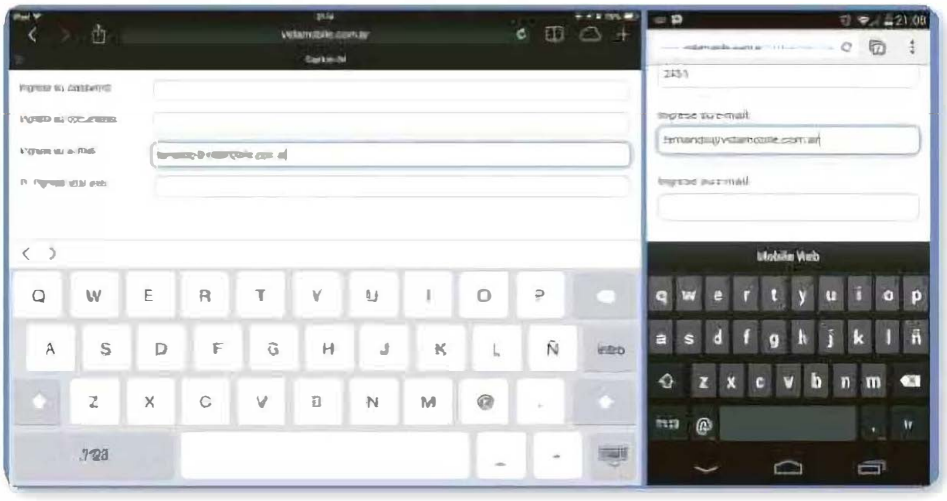


Figura 11. **`Text input email`** nos permite ajustar el teclado de los dispositivos móviles con el carácter arroba disponible.

```
<input type="email" id="inputEmail" value="" />
```

URL

Similar al anterior, este input type nos habilita a escribir una URL. En los dispositivos modernos, dependiendo de la marca, puede habilitar o no teclas especiales para ingresar **www.**, **.com**, y **/**.

En los dispositivos más modernos, dependiendo del sistema operativo, puede incluir solo la tecla especial **/**.

```
<input type="url" id="inputURL" value="" />
```

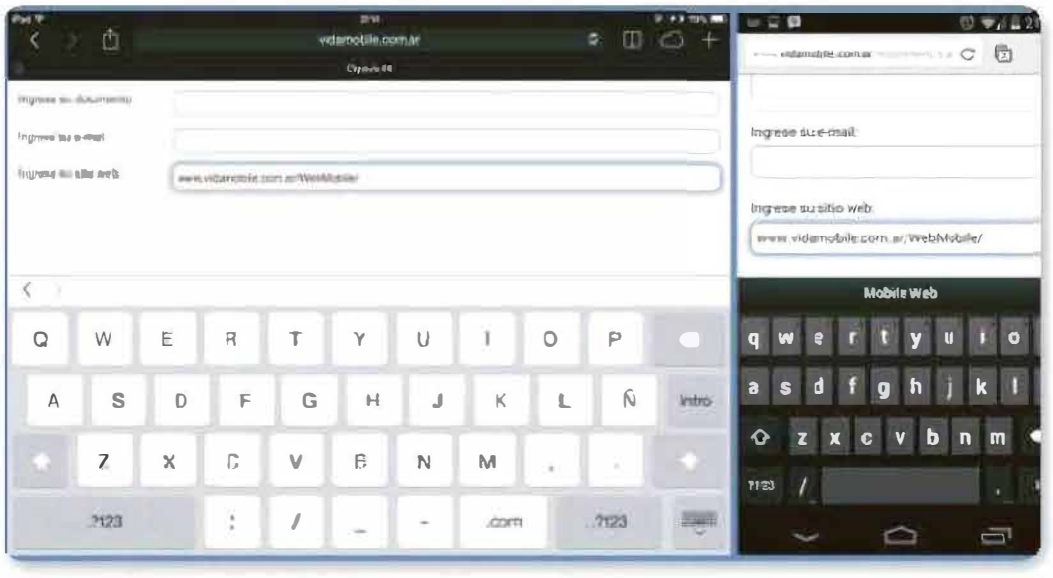


Figura 12. **`Text input` URL** nos habilita las teclas **`.com`** y **`/`** para escribir direcciones web.

Tel

Este input type funciona, al igual que **Number**, permitiendo ingresar un número de teléfono en el campo. Recordemos que la activación del teclado numérico sólo funciona en smartphones en modo vertical y, a diferencia de **Number**, habilita los caracteres * y #.

```
<input type="tel" id="inputTel" value="" />
```

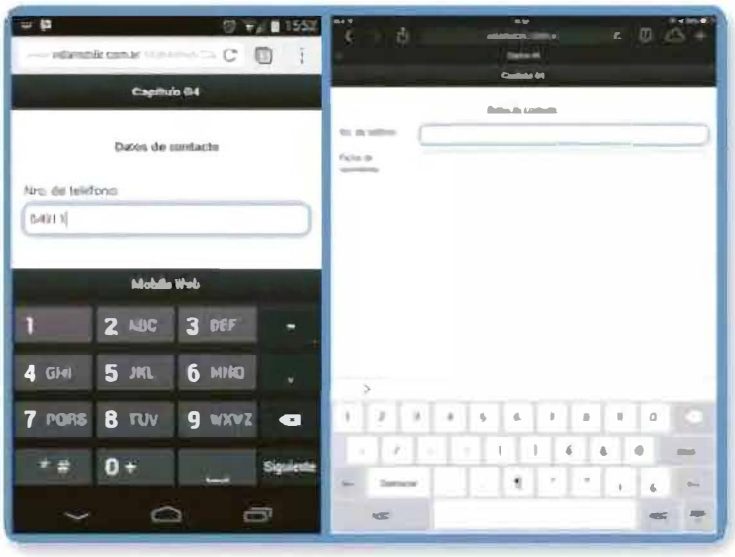


Figura 13. `Text input Tel` nos habilita el teclado numérico en smartphones, y en una tablet prioriza el acceso al pad numérico.

Date

A través de **input type="date"** podremos ingresar una fecha formateada como día/mes/año. Debemos tener en cuenta que este input type se visualiza de forma distinta en los diferentes motores de navegadores web; por lo tanto, en determinadas ocasiones este tipo de campo funcionará sólo como un **TextBox** más.

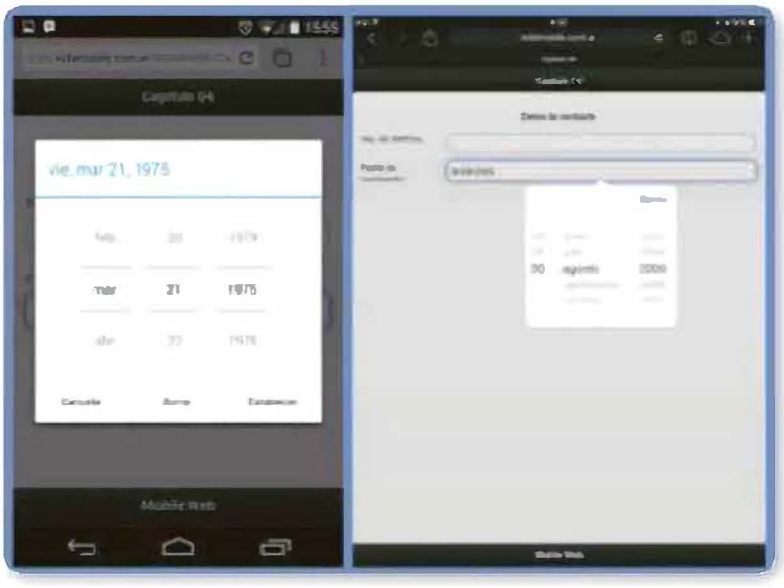


Figura 14. `Text input Date` adopta la versión de visualización de calendario de la plataforma móvil sólo en motores WebKit.

Themes

jQuery Mobile incorpora una serie de **temas** en su plantilla CSS. Si utilizamos una versión antigua, disponible hasta la **1.3.2**, podemos contar con hasta cinco temas diferentes que adornan la estética de cada widget, mientras que en la última versión estable al momento de escribir este capítulo (**1.4.2**), jQuery Mobile limitó a dos temas su estética gráfica.

HASTA LA VERSIÓN 1.3.2, JQUERY MOBILE INCORPORABA CINCO TEMAS DIFERENTES

Igualmente, esto no es un impedimento para que destaquemos nuestro sitio web desarrollado con jQuery Mobile, ya que este incorpora una versión web disponible en **http://themroller.jquerymobile.com** de un generador de temas online.

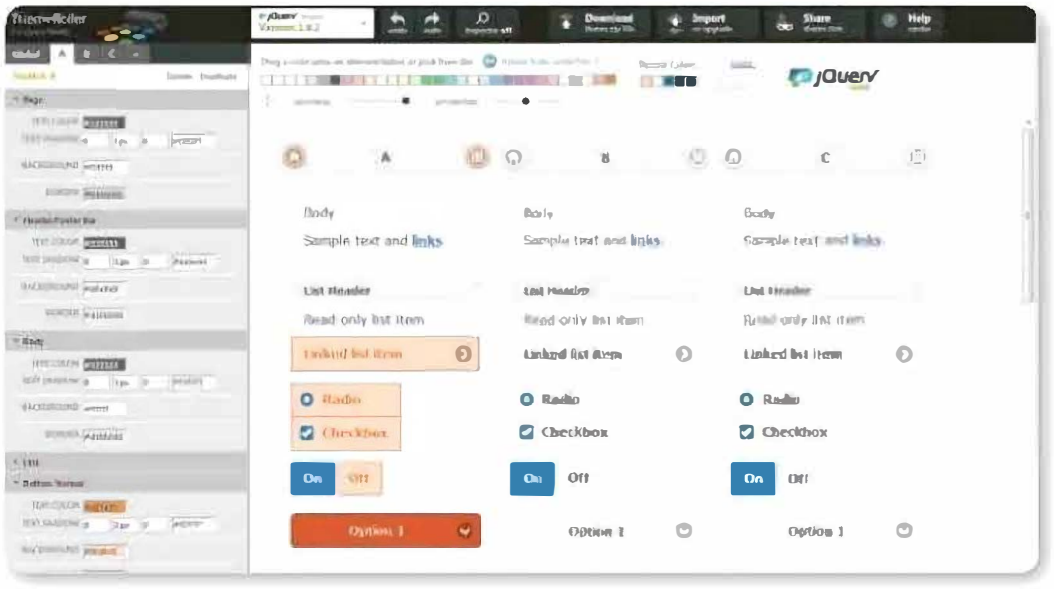


Figura 15. **ThemeRoller** nos permitirá ajustar los colores y estilos de cada widget de jQuery Mobile.

INPUT TYPE DATE

Input Type Date varía su visualización según el navegador web que utilice el usuario de nuestra WebApp; por lo tanto, otra alternativa será utilizar, de forma independiente, un campo para el día, otro para el mes y otro para el año, para que el usuario cargue por separado el valor de la fecha en cada uno.

A través de esta herramienta, podremos crear íntegramente un tema desde cero, basándonos en las paletas de colores que por defecto nos ofrece ThemeRoller o eligiendo específicamente un color para cada uno de los componentes y widgets de esta herramienta.

A medida que elegimos un objeto de la interfaz gráfica de JQM, podremos personalizar el color, la sombra, la fuente, el tamaño de los bordes, la sombra de cada contorno y otras acciones más.

Al finalizar la especificación de nuestro tema, podemos descargar el archivo CSS correspondiente con los cambios que realizamos para así reemplazarlo en la ruta de nuestras páginas web.

Ejercicio integrador

Nuevamente nos encontramos con un ejercicio integrador que nos ayudará a repasar algunos conceptos hasta ahora vistos y a conocer mejor el resto de las bondades que JQM pone a nuestra disposición.

Volvemos a utilizar el ejercicio realizado en el **Capítulo 3**, que adaptaremos ligeramente para que pueda visualizarse de manera diferenciada en tablets. Descarguemos, entonces, el código de ejemplo de este ejercicio desde **premium.redusers.com** para poder modificar su estructura.

LA EXPERIENCIA DE NAVEGACIÓN CAMBIA ROTUNDAMENTE ENTRE DISPOSITIVOS DE CINCO Y SEIS PULGADAS

Adaptación a las pantallas de tablets

Como bien sabemos, las pantallas de tablets (ya sean en formato vertical u horizontal) nos brindan una experiencia de navegación mucho más cómoda y agradable que la que otorga un teléfono móvil.

Si disponemos de un dispositivo de hasta cinco pulgadas, la usabilidad de una WebApp será normal y se igualará a cualquier otra app nativa instalada en el teléfono. Ahora, si disponemos de un dispositivo denominado **Phablet**, o de una tablet por arriba de las

seis pulgadas, la experiencia de navegación cambiará rotundamente, ya que sus pantallas poseen mucho más espacio para distribuir componentes y visualizar texto, imágenes y video.

Por ello, tomando la idea de que nuestra página web puede visualizarse en un smartphone o una tablet, simularemos la adaptación de nuestro proyecto a las pantallas más grandes, pudiendo, a su vez, aprovechar características de JQM que hasta ahora no hemos explorado.

Cambiar NavBar por el widget Panel

Hasta ahora aprovechamos una barra de herramientas que nos permitió desplazarnos por el sitio web corporativo que desarrollamos en el capítulo anterior. Ahora, simulando que su contenido debe ser explotado en una tablet, lo modificaremos para poder aprovechar las características que nos brindan las pantallas de mayor amplitud.

En principio, reemplazaremos **NavBar** por un panel lateral que aparecerá al momento que lo invoquemos. Luego eliminaremos la barra de navegación inferior y ubicaremos, en su lugar, **div data-role="footer"**.

Luego abrimos una copia de nuestro proyecto anterior y editamos el archivo **index.html**. Entre ** ** y ** ** escribimos el siguiente código:

```
<div data-role="page" id="productos" data-theme="d">
  <div data-role="panel" id="panelMenu" data-display="reveal" data-
position="left" data-theme="a">
    <div id="tituloMenu" align="center">
      <p>Men&uacute;</p>
      <p> </p>
    </div>
    <ul data-role="listview" data-theme="a">
      <li><a href="index.html">Historia</a></li>
      <li><a href="#" class="ui-btn-active ui-state-persist">Productos</a></li>
      <li><a href="showroom.html">Showroom</a></li>
      <li><a href="contacto.html">Contacto</a> </li>
```

```
        </ul>
    </div>
        <div data-role="header" data-theme="c" align="center">
...
```

Creamos un **div data-role="panel"** al cual identificamos con un nombre a través de su **id="panelMenu"**. Luego le indicamos al panel, a través del atributo **data-position="left"**, que debe estar alineado a la izquierda de la pantalla y, mediante el efecto **data-display="reveal"**, que se debe visualizar.

A continuación, creamos el menú para cada una de las secciones que nuestro sitio web posee. Para ello, declaramos un nuevo **div** cuyo nombre es **id="tituloMenu"** con alineación en el centro. En su interior, creamos un párrafo titulado **Menú**. Por último, cerramos el **div**. Seguido a este, creamos el menú de navegación para el sitio mediante el componente **ListView**, aplicándole un tema **data-theme="a"**.

Nos queda, a continuación, agregar el sistema de lista, con el cual armamos cada botón de Navigation Bar. Para esto, copiamos el código completo de **ListView** ubicado dentro del **div** inferior y lo pegamos seguido al tag **Menú** que creamos. Quitamos de cada **ListItem** el atributo **data-icon**, dado que no será utilizado. JQM asigna el icono **arrow-r** a cada **ListItem** para indicar cuando este tiene un hipervínculo asociado.

Figura 16. El resultado de nuestro objeto **Panel**, que reemplazará al menú original de nuestro proyecto.

Una vez que obtenemos el resultado de la **Figura 16,** debemos repetir esto por cada uno de los archivos que componen nuestro sitio: **productos.html, showroom.html** y **contacto.html**, utilizando los atributos establecidos en cada hipervínculo de cada una de las páginas.

Reemplazar Navigation Bar por Footer

Para poder estilizar el sitio web móvil ahora que no tenemos un pie de página como menú de navegación, reemplazaremos este por **data-role="footer"**. Eliminamos el **div** Navigation Bar completo y lo reemplazamos por el siguiente código:

```
<div data-role="footer" data-position="fixed" data-theme="c">
    <h4>Copyright 2014 - Fernando Luna</h4>
</div>
```

Solo nos queda un cambio más por hacer: habilitar un botón superior que nos permita desplegar el nuevo menú de navegación. Como estamos acostumbrados a ver en las versiones móviles de las redes sociales más populares (**Facebook** y **Google Plus**, entre otras), el botón de despliegue del menú lateral suele ubicarse en el extremo superior izquierdo de la pantalla.

Para ello, debemos aplicar un pequeño cambio en el **Div header** de nuestro proyecto. Analicemos el código de **header** a continuación:

```
<div data-role="header"data-theme="c" align="center">
    <div id="imagenLogo" align="center">
        <img src="imagenes/Logo800x150.gif" style="width:80%;max-
width:400px" alt="Greenberries Logo" longdesc="Greenberries Farming Logo">
    </div>
</div>
```

Dentro del **data-role="header"**, creamos un **div** con el nombre **id="imagenLogo"** y alineación centrada. Para crear un botón que permita desplegar el widget **Panel**, debemos agregar antes de **id="imagenLogo"** un hipervínculo formateado con los siguientes atributos:

- **data-icon="bars"**
- **data-corners="false"**
- **data-iconpos="notext"**

Los atributos recién listados nos permiten establecer un icono del tipo barra, del estilo del menú utilizado en la mayoría de las apps actuales, sin bordes redondeados. Al no utilizar el atributo **text** en este botón, lo desactivaremos mediante la instrucción **data-iconpos="notext"**, para que la imagen tome la posición central del **div**.

El código final de **header** debe ser igual al siguiente:

```
<div data-role="header"data-theme="c">
    <a href="#panelMenu" data-icon="bars" data-corners="false" data-
iconpos="notext"></a>
    <div id="imagenLogo" align="center">
      <img src="imagenes/Logo800x150.gif" style="width:80%;max-
width:400px" alt="Greenberries Logo" longdesc="Greenberries Farming Logo">
    </div>
</div>
```

El hipervínculo **href="#panelMenu"** hará que, al presionar este objeto, se invoque el panel correspondiente, visualizando el menú lateral que hemos creado. Por último, apliquemos nuevamente la modificación del **header** en el resto de las páginas y, de esta manera, ya podremos visualizar el panel lateral en cada una de las secciones que tiene nuestro sitio.

A continuación, el código completo de cada página creada:

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Greenberries Farming</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
```

```
mobile-1.3.2.min.css" />
        <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
    </script>
    </head>
    <body>
        <div data-role="page" id="index" data-theme="d">
            <div data-role="panel" id="panelMenu" data-display="reveal" data-
position="left" data-theme="a">
                <div id="tituloMenu" align="center">
                    <p>Men&uacute;</p>
                    <p> </p>
                </div>
                                <ul data-role="listview" data-theme="a">
                    <li><a href="#" class="ui-btn-active ui-state-persist">Historia</
a></li>
                    <li><a href="productos.html">Productos</a></li>
                    <li><a href="showroom.html">Showroom</a></li>
                    <li><a href="contacto.html">Contacto</a> </li>
                </ul>
            </div>
            <div data-role="header" data-theme="c">
                    <a href="#panelMenu" data-icon="bars" data-corners="false"
data-iconpos="notext"></a>
                <div id="imagenLogo" align="center">
                    <img src="imagenes/Logo800x150.gif" style="width:80%;max-
width:400px" alt="Greenberries Logo" longdesc="Greenberries Farming Logo">
                </div>
            </div>
            <div data-role="content" data-theme="d">
                <p>Bienvenido a <b>Green&Berries Farming</b>.</p>
                <p>Nuestra compa&ntilde;&iacute;a se especializa en la
plantaci&oacute;n, cosecha y distribuci&oacute;n por mayor y menor de frutos rojos
```

```
y productos relacionados.</p>
                <p></p>
                <p>A diferencia de otras firmas productoras, Green&Berries po-
see dos l&iacute;neas de productos: Frutos rojos org&aacute;nicos, sembrados
y cosechados bajo los principales est&aacute;ndares de calidad, y Frutos rojos
hidrop&oacute;nicos, que cumplen con su etiqueta org&aacute;nica, pero con la dife-
rencia que estos son sembrados y desarrollados &iacute;ntegramente bajo el sistema
de hidroponia.</p>
        </div>
        <div data-role="footer" data-position="fixed" data-theme="c">
                                <h4>Copyright 2014 - Fernando Luna</h4>
        </div>
    </div>
</body>
</html>
```

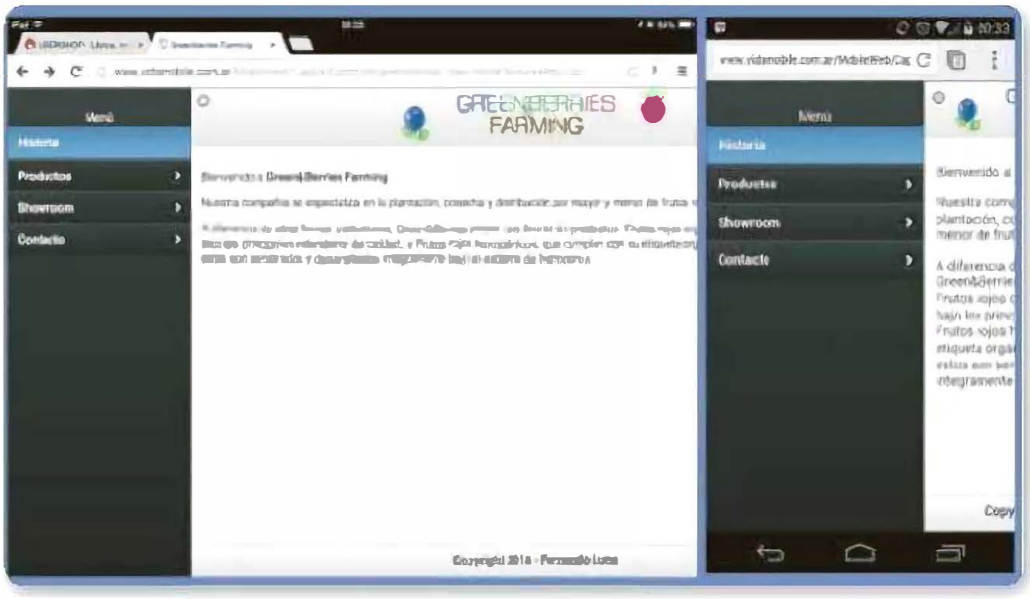


Figura 17. El panel en acción, tanto en Android como en iPad.

productos.html

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Greenberries Farming</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
  <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
</script>
</head>
<body>
  <div data-role="page" id="productos" data-theme="d">
    <div data-role="panel" id="panelMenu" data-display="reveal" data-
position="left" data-theme="a">
      <div id="tituloMenu" align="center">
        <p>Men&uacute;</p>
        <p> </p>
      </div>
      <ul data-role="listview" data-theme="a">
        <li><a href="index.html">Historia</a></li>
        <li><a href="#" class="ui-btn-active ui-state-persist">Productos</
a></li>
        <li><a href="showroom.html">Showroom</a></li>
        <li><a href="contacto.html">Contacto</a> </li>
      </ul>
    </div>
    <div data-role="header"data-theme="c" align="center">
      <a href="#panelMenu" data-icon="bars" data-corners="false" data-
iconpos="notext"></a>
      <div id="imagenLogo" align="center">
        <img src="imagenes/Logo800x150.gif" style="width:80%;max-
width:400px" alt="Greenberries Logo" longdesc="Greenberries Farming Logo">
      </div>
    </div>
```

```
        <div data-role="content" data-theme="d">
            <h4>Productos</h4>
                <p>A continuaci&oacute;n puede conocer los productos que comer-
cializamos. Haga clic sobre la categor&iacute;a de su inter&eacute;s.</p>
                <div id="imagenes-links" align="center">
                <a href="organicos.html" data-rel="dialog"><img src="imagenes/
productos-organicos.jpg" Style="width:100%;min-width:150px;max-
width:600px;border:1px" /></a><p></p>
                <a href="hidroponicos.html" data-rel="dialog"><img
src="imagenes/productos-hidroponicos.jpg" Style="width:100%;min-
width:150px;max-width:600px;border:1px" /></a>
                </div>
        </div>
        <div data-role="footer" data-position="fixed" data-theme="c">
            <h4>Copyright 2014 - Fernando Luna</h4>
        </div>
    </div>
</body>
</html>
```

Figura 18. Todas las secciones se comportan por igual luego de utilizar el atributo **`ui-button-active`**.

showroom.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Greenberries Farming</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
</script>
</head>
<body>
   <div data-role="page" id="productos" data-theme="d">
      <div data-role="panel" id="panelMenu" data-display="reveal" data-
position="left" data-theme="a">
         <div id="tituloMenu" align="center">
            <p>Men&uacute;</p>
            <p> </p>
         </div>
         <ul data-role="listview" data-theme="a">
            <li><a href="index.html">Historia</a></li>
            <li><a href="productos.html">Productos</a></li>
            <li><a href="#" class="ui-btn-active ui-state-persist">Showroom</
a></li>
            <li><a href="contacto.html">Contacto</a> </li>
         </ul>
      </div>
      <div data-role="header"data-theme="c" align="center">
         <a href="#panelMenu" data-icon="bars" data-corners="false" data-
iconpos="notext"></a>
         <div id="imagenLogo" align="center">
            <img src="imagenes/Logo800x150.gif" style="width:80%;max-
width:400px" alt="Greenberries Logo" longdesc="Greenberries Farming Logo">
```

```
            </div>
        </div>
        <div data-role="content" data-theme="d">
            <h4>Atenci&oacute;n al p&uacute;blico</h4>
            <div id="mapa" align="center">
              <img border="2" style='border-color:#69C' src="http://maps.
googleapis.com/maps/api/staticmap?center=-34.618747,-58.842229&zoom=16&
size=350x200&markers=size:mid%7Ccolor:blue%7Clabel:G%7C-34.618747,-
58.842229&sensor=true" />
            </div>
        <p>Lo invitamos a conocer nuestro Showroom ubicado en
<strong>Autopista del Oeste, KM. 44.500, La Reja, Buenos Aires</strong>.</p>
        <p>Aqu&iacute; encontrar&aacute; todos nuestros productos envasados y
listos para llevar. Comun&iacute;quese previamente por tel&eacute;fono para coordi-
nar una cita: <strong>02321-234-9876</strong></p>
        </div>
        <div data-role="footer" data-position="fixed" data-theme="c">
            <h4>Copyright 2014 - Fernando Luna</h4>
        </div>
    </div>
</body>
</html>
```

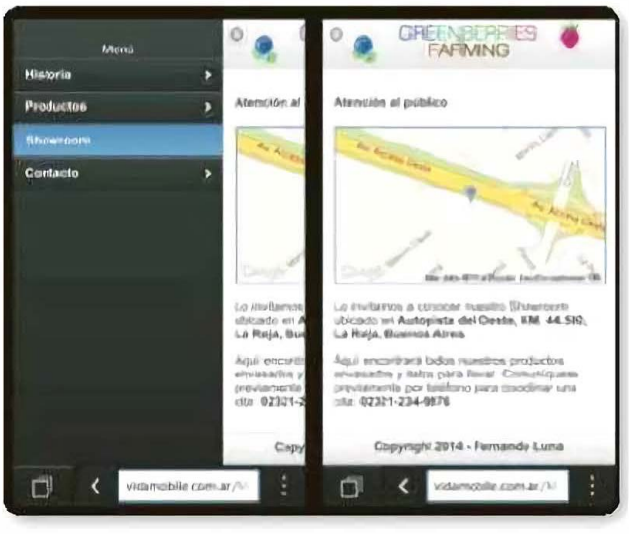


Figura 19. El menú desplegado en la sección Showroom de nuestro proyecto corriendo en el simulador BlackBerry 10.

contacto.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Greenberries Farming</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
      <script type="text/javascript">
            var email = document.getElementById('correo');
            var asunto = document.getElementById('asunto');
            var msj = document.getElementById('mensaje');
            function sendMail() {
                  var asunto = document.getElementById("asunto").value;
                  var mensaje = document.getElementById("mensaje").
value;
                  var link = "mailto:info@greenberriesfarming.com"
                   + "?cc="+email
                   + "&subject="+asunto.replace(" ", "%20")
                   + "&body=" +mensaje.replace(" ","%20")
                  window.location.href = link;
                  }
   </script>
</head>
<body>
   <div data-role="page" id="productos" data-theme="d">
      <div data-role="panel" id="panelMenu" data-display="reveal" data-
position="left" data-theme="a">
         <div id="tituloMenu" align="center">
            <p>Men&uacute;</p>
            <p> </p>
         </div>
```

```
            <ul data-role="listview" data-theme="a">
                <li><a href="index.html">Historia</a></li>
                <li><a href="productos.html">Productos</a></li>
                <li><a href="showroom.html">Showroom</a></li>
                <li><a href="contacto.html" class="ui-btn-active ui-state-
persist">Contacto</a> </li>
            </ul>
        </div>
        <div data-role="header"data-theme="c" align="center">
            <a href="#panelMenu" data-icon="bars" data-corners="false" data-
iconpos="notext"></a>
            <div id="imagenLogo" align="center">
                <img src="imagenes/Logo800x150.gif" style="width:80%;max-
width:400px" alt="Greenberries Logo" longdesc="Greenberries Farming Logo">
            </div>
        </div>
        <div data-role="content" data-theme="d">
            <h4>Formulario de contacto</h4>
                                <form id="email" method="put"
action="enviarEmail()" >
                <input type="email" id="correo" placeholder="su correo
electr&oacute;nico" />
                <input type="text" id="asunto" name="asunto"
placeholder="Asunto" />
                <div class="ui-field-contain">
                                <textarea name="mensaje" id="mensaje"
placeholder="Mensaje" rows="10" cols="50" ></textarea>
                <input type="button" id="enviar" value="Enviar"
onclick="sendMail()" data-inline="true"/>
            </form>
                </div>
    </div>
    <div data-role="footer" data-position="fixed" data-theme="c">
        <h4>Copyright 2014 - Fernando Luna</h4>
    </div>
</body>
</html>
```

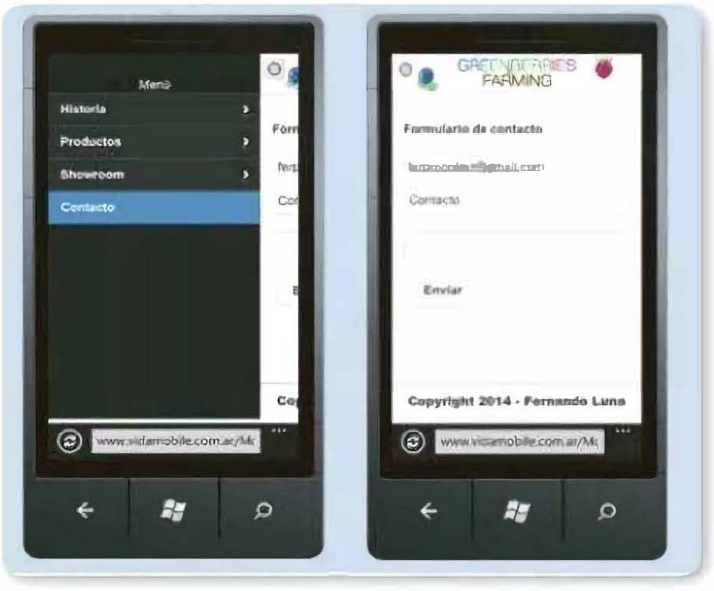


Figura 20. La última sección de nuestro proyecto, representada en el simulador de Windows Phone.

RESUMEN

Hasta aquí hemos conocido los componentes de jQuery Mobile que nos permiten desplegar un desarrollo web móvil con muy poco esfuerzo y, a su vez, obtener resultados reales en tiempo récord. Pudimos también verificar cómo nuestra app se comporta en diferentes plataformas móviles –como, iOS, Android, Windows Phone y BlackBerry 10–, lo que nos ayudó a ver el resultado certero que jQuery Mobile propone desde su framework.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cómo puedo visualizar un link activo en jQuery Mobile?
2. ¿Cómo quito los bordes redondeados de un botón?
3. ¿Qué es una lista autodividida?
4. ¿Qué widget nos permite aplicar un filtro de búsqueda?
5. ¿Cómo deshabilito un widget **Button**?
6. Mencione al menos un campo que permita visualizar el teclado numérico.
7. ¿Para qué sirve el **input type URL**?
8. ¿Qué nos permite realizar el objeto **Panel**?
9. ¿En qué sector se ubica el **data-role="Panel"**?
10. ¿Cómo puedo personalizar los colores de un sitio construido con JQM?

EJERCICIOS PRÁCTICOS

1. Explore **ThemeRoller** y cree un tema personalizado.
2. Investigue cómo descargar un tema desde ThemeRoller y cómo aplicarlo a nuestro proyecto.
3. Investigue qué otras opciones de visualización de un widget **Panel** podemos utilizar.
4. Investigue qué otras opciones de **input type** existen.
5. Busque un simulador móvil disponible en internet, descárguelo e instálelo.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*05

Interacción con el hardware de comunicaciones

Este capítulo nos trasladará al terreno de las comunicaciones a través de los dispositivos móviles: videoconferencias, llamadas vía Skype, llamados telefónicos y mensajes de texto. Estas son algunas de las opciones que aprenderemos a integrar en nuestras aplicaciones web móviles: conoceremos cómo invocarlas y cómo se comporta cada una de ellas cuando el dispositivo no posee alguna característica.

La Web y el hardware de los dispositivos móviles

Con el avance tecnológico de las comunicaciones y la masificación a nivel mundial de las computadoras, internet, los teléfonos móviles y las tablets, entre otros dispositivos, hoy contamos con un sinnúmero de opciones para comunicarnos con nuestros pares.

Y lo mejor de todo es que, gracias a la integración de los servicios de datos en los teléfonos móviles y tablets, podemos hacer uso de casi todas estas formas de comunicación desde nuestro dispositivo móvil. Tampoco podemos dejar de reconocer que hubo una fuerte penetración de la red internet en los smartphones y tablets, lo cual hizo evolucionar favorablemente los estándares web, para que estos últimos se integraran con las diversas opciones de comunicación con las que contamos hoy.

Cuando hablamos de **sistemas de comunicación**, debemos separar el concepto en tres metodologías distintas:

- **Sistema de comunicación por voz**.
- **Sistema de comunicación por texto escrito**.
- **Sistema de videocomunicación**.

A su vez, estos sistemas de comunicación se despliegan a través de dos **canales** diferentes:

- La red de telefonía móvil.
- La red de datos móviles.

Y si representamos estos sistemas de comunicación mediante el sistema de pertenencia utilizado en la **Teoría de conjuntos**, notaremos que todos ellos conjugan en ambas plataformas y ya ninguno es exclusivo de la red de telefonía móvil.

Esto ocurre gracias a sistemas como **Skype**, que permiten, con un costo mínimo, establecer una comunicación con un teléfono fijo o móvil a través de la red **voIP**, la cual integra los sistemas de comunicación telefónica clásicos a través de la red de datos de internet.

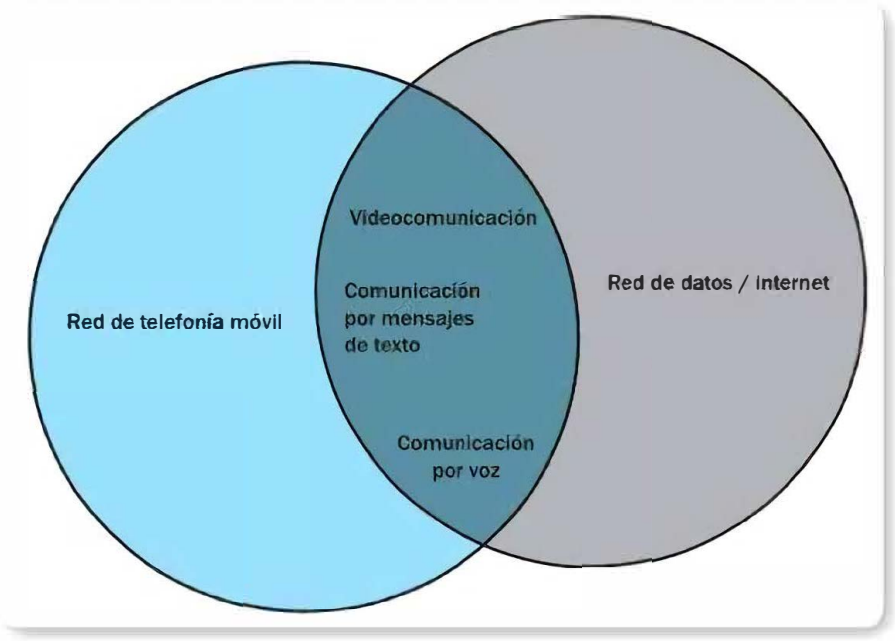


Figura 1. Este pequeño diagrama, basado en la Teoría de conjuntos, nos permite representar el canal de transporte para la diversidad de comunicaciones actuales.

La red de telefonía clásica ya no es un requisito indispensable para comunicarse con otras personas. En un futuro cercano, todos los sistemas de comunicaciones que utilizan una red de telefonía (móvil o fija) terminarán migrando hacia la red de datos, aprovechando las grandes ventajas que brinda la telefonía VoIP.

Si bien esto parece demasiado futurista por el momento, no podemos pasar por alto que algunos fabricantes de equipos telefónicos inalámbricos disponen ya, en el mercado, de modelos de aparatos que funcionan con pantallas táctiles y corren, por ejemplo, el sistema operativo Android, con acceso a la tienda de aplicaciones Google Play incluido.

Este tipo de equipos nos permite contar con un sistema como Skype, instalado en el aparato telefónico, a través del cual podemos aprovechar, por una tarifa mínima, los sistemas de comunicación telefónica a cualquier parte del mundo, y hasta contar con un número de teléfono internacional propio. Solo necesitamos tener acceso a internet para que esto funcione.

> LA RED DE TELEFONÍA CLÁSICA YA NO ES UN REQUISITO INDISPENSABLE PARA HABLAR CON OTROS

Los sistemas web evolucionaron a través de los navegadores móviles y –volviendo al terreno de la programación– desde hace unos años nos permiten contar con un conjunto de protocolos y etiquetas para interactuar de forma transparente con el hardware de los dispositivos móviles. Veamos, a continuación, cómo sacar provecho de estos mecanismos desde nuestra web móvil.

Hipervínculos en jQuery Mobile

En los primeros capítulos de esta obra conocimos la etiqueta **link**, la cual nos permite establecer un hipervínculo hacia otra página web o enviar un correo electrónico. En los navegadores web móviles se extendió esta funcionalidad, al agregar la capacidad de invocar a eventos de llamada telefónica, SMS, Skype, entre otros. El uso de **link** se realiza a través del tag **<a href="url">**. En jQuery Mobile podemos utilizarlo de diversas formas, como, por ejemplo, a través de un botón:

```
<a href="www.misitio.com" data-role="button" id="miBoton">Ir al sitio web</a>
```

También podemos utilizarlo con un simple texto que contenga una URL:

```
<a href="www.misitio.com" id="miLink">Ir al sitio web</a>
```

O a través del uso de listas:

```
<ul data-role="listview" id="miLista">
   <il><a href="www.misitio.com">Ir al sitio web</a></il>
   <il><a href="www.otrositio.com">Ir a otro sitio web</a></il>
</ul>
```

URI SCHEMES

Se denomina **URI schemes** a las especificaciones de caracteres reservados que permiten establecer un hipervínculo a una función especifica dentro de un sistema operativo o hacia alguna aplicación en particular. Son muy utilizadas en internet, y cada vez más empresas de software suman estas características a la Web actual.

En estos ejemplos, vemos el uso común que suele darse a **<a href>** para establecer un hipervínculo, tanto para una web visualizada en una computadora de escritorio como también para una web móvil. Ahora exploraremos las alternativas de hipervínculos para establecer comunicaciones desde una web móvil, utilizando el hardware de nuestro teléfono.

Interacción con el smartphone o tablet

Casi todos los navegadores web móviles modernos le dan soporte al estándar de comunicaciones a través de hipervínculos que podemos ejecutar desde un smartphone.

Desde las primeras versiones de estos equipos inteligentes –más precisamente, con el nacimiento de **iPhone**–, el navegador web **Safari** estableció un protocolo automático que procesaba la página web que cargaba y generaba, a continuación, un hipervínculo sobre los números telefónicos que en esta aparecían.

Con el tiempo, el resto de los navegadores web móviles adoptaron este estándar, con lo cual hoy es posible invocar los métodos de comunicación comunes en un teléfono inteligente desde una página web.

Realizar una llamada

Para establecer un hipervínculo a una llamada desde una página web, podemos hacer uso de la etiqueta **Tel**. Para ello, debemos escribir la siguiente sentencia:

```
<a href="tel:54911-1234-5678">Llámame</a>
```

La página que posee un link de este tipo habilita a que el usuario que lo presione cargue el número telefónico especificado en el hipervínculo para poder establecer una llamada telefónica.

Pero esto no funciona de manera automática en los equipos. Cuando el usuario ejecute el hipervínculo, el smartphone cargará el número telefónico en el **Pad** del teléfono, o le preguntará al usuario si desea establecer una comunicación telefónica con dicho número, dependiendo del sistema operativo móvil en el cual sea realizada esta acción.

Esto ocurre por una cuestión de seguridad, para que el usuario identifique que la acción iniciará una llamada telefónica, y no que esto ocurra de manera automática.

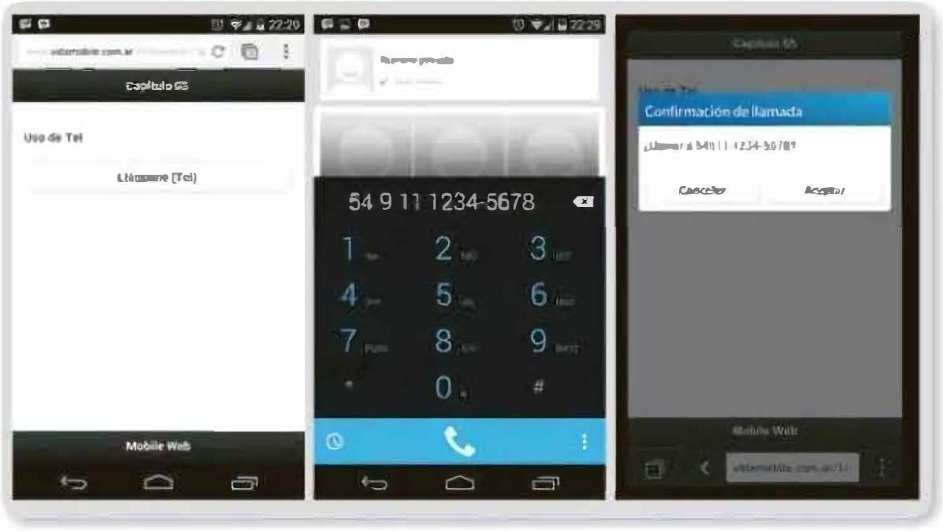


Figura 2. Aquí podemos ver cómo se comporta cada plataforma al invocar el hipervínculo **Tel**.

Formato numérico

El número de teléfono que agregamos en el hipervínculo puede estar escrito de manera estándar, si es que la llamada a realizar se encuentra en una página que solo visualizan usuarios de la misma región y localidad.

Para las páginas que son utilizadas a nivel internacional, se debe establecer el código de país seguido del código de área. Esto permitirá realizar la llamada desde un país o localidad diferente de donde se encuentre ubicado físicamente el usuario.

USO DE NÚMEROS DE TELÉFONO EN IPHONE

Los sitios web que cargan páginas donde figura un número telefónico utilizando el navegador web Safari para iPhone formatean automáticamente el número para que podamos llamar con solo presionar sobre este. Para deshacer esta acción, debemos incluir en el header de la página la etiqueta **<meta name="format-detection" content="telephone=no">.**

En el ejemplo realizado estamos incluyendo un número de teléfono móvil cuyo código de país corresponde a Argentina y su código de localidad, a Buenos Aires.

CÓDIGOS TELEFÓNICOS PARA LA CIUDAD DE BUENOS AIRES

FORMATO NUMÉRICO	VALOR	REGIÓN
Código de país	54	Argentina
Identificador de teléfono móvil	9	Argentina
Código de área	11	Buenos Aires (AMBA)

Tabla 1. Es importante tener la tabla de códigos telefónicos internacionales para desarrollar un sistema de comunicación a teléfonos móviles.

Button Call en jQuery Mobile

En el diseño de una aplicación web móvil, es conveniente agrupar los números de teléfono a los cuales deseamos habilitar la función de llamada dentro de un botón con un icono distintivo. Veamos cómo realizaríamos esto con el widget **Button** de jQuery Mobile:

```
<a href="tel:+54911-1234-5678" data-role="button" id="miBoton"><img
src="imágenes/phone-icon.png" />+54911-1234-5678</a>
```

El uso del guion "-" como separador en el número de teléfono no es necesario. Se utiliza en aplicaciones o en textos para estructurar la lectura del número telefónico; por lo tanto, no tiene injerencia en el comportamiento de la acción a realizar.

Skype Call

Cuando se popularizó la plataforma Skype, la firma entonces dueña de este producto instauró, en el mercado web, los denominados **Skype Buttons**. Estos botones se podían integrar en páginas web para que el usuario que tuviese Skype en su computadora pudiera contactar a una empresa que brinda atención a través de esta vía con solo presionar este botón.

Junto con la instalación de Skype en los equipos locales, se instalaba también una librería que modificaba el comportamiento del navegador para que este, al presionar el botón Skype, iniciara esta aplicación y llamara directamente al usuario que estaba identificado en el código de dicho botón.

Con la proliferación de las plataformas móviles, Skype lanzó, a través del apartado **Developers** de su portal, el **URI scheme** correspondiente para que esto pudiera implementarse en todo aquel navegador móvil que quisiera integrar el llamado a través de Skype.

Gracias a esta opción –que fue muy bien adoptada por los principales navegadores web móviles–, si deseamos invocar una llamada a un equipo con Skype desde una web móvil, debemos crear el siguiente hipervínculo:

```
<a href="skype:usuariodeskype?call" data-role="button"
id="miBotonSkype"><img src="imagenes/skype-icon.png" width="18px"
height="18px"/> Ll&aacute;mame por Skype</a>
```

Al presionar el botón donde figure un **nombre de usuario Skype** válido, se ejecutará la plataforma de mensajería y se iniciará la comunicación con el destinatario especificado **usuariodeskype**.

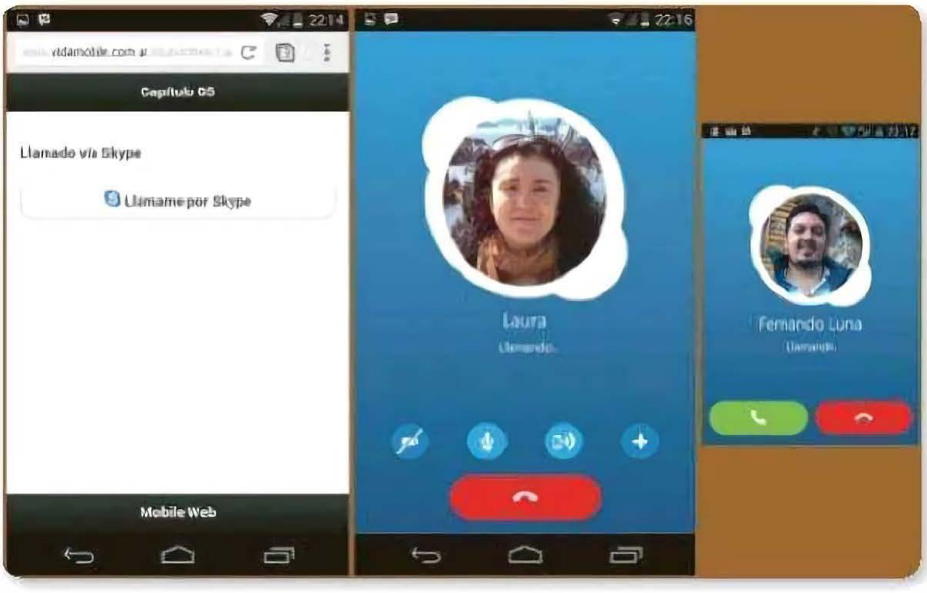


Figura 3. Con solo presionar el hipervínculo se iniciará automáticamente la llamada a través de Skype.

También se puede reemplazar el nombre de usuario Skype por un número de teléfono válido. Esta última acción funcionará solamente si tenemos crédito cargado para realizar llamadas telefónicas a través de esta plataforma.

Por supuesto que es también un requisito tener Skype instalado en el dispositivo móvil y una cuenta activa para que todo esto ocurra.

Al invocar una llamada a un usuario Skype, automáticamente se iniciará la aplicación, la cual tratará de establecer una comunicación por voz, utilizando esta plataforma como medio.

También es posible realizar otras acciones a través de este mismo URI scheme. Si, en lugar de una llamada, preferimos utilizar un **sistema de chat**, simplemente debemos reemplazar el parámetro pasado después del signo de pregunta por el valor **chat**.

```
<a href="skype:TheSkypeUsername?chat" ...>
```

En la implementación de este URI scheme desde las plataformas móviles no se contempló el inicio de una sesión de Skype directamente a través de videoconferencia, ya que los equipos móviles habitualmente utilizan un pack de datos restringido a un consumo determinado por mes, y por ello se busca evitar el gasto por parte del cliente que inicia la comunicación.

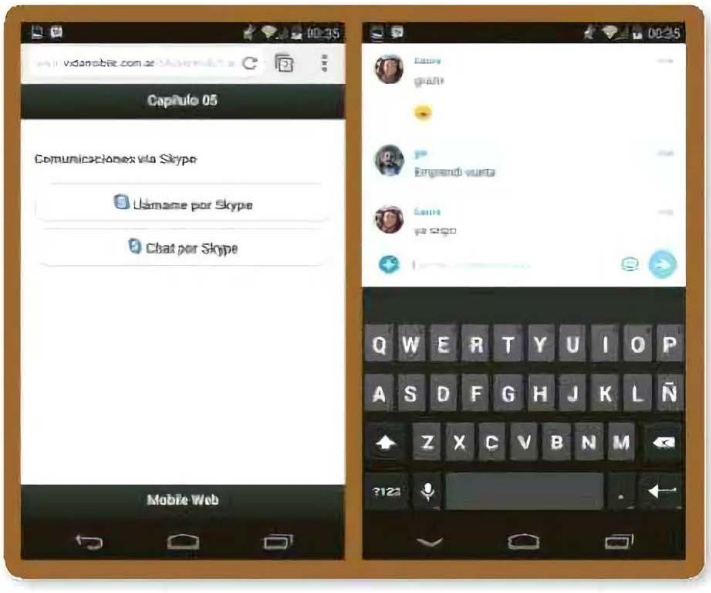


Figura 4. Al igual que con el llamado telefónico vía Skype, también podremos iniciar una nueva conversación de chat.

En caso de que se requiera sí o sí iniciar una **videoconferencia**, se puede activar la videocámara luego de iniciar el llamado.

Videoconferencia en iOS con FaceTime

Si bien disponemos de una versión de Skype para iOS, los fieles usuarios de la marca Apple disponen de su propio sistema de videocomunicación, conocido como **FaceTime**. Si debemos realizar un desarrollo específico para esta plataforma y tenemos que integrar FaceTime como software de comunicaciones base, también disponemos de URI scheme.

```
<a href="Facetime:+5491123456789" data-role="button" data-icon="skype"
id="miBotonFTime">Llámame por Facetime</a>
```

Este hipervínculo establecerá una comunicación telefónica, con la opción de activar el video, utilizando la plataforma FaceTime. Ambos interlocutores deberán tener esta plataforma configurada en sus equipos iPhone: esta es la única opción para establecer un llamado mediante un número telefónico.

Figura 5. En la plataforma iOS, también es posible invocar las conferencias vía FaceTime desde una WebApp.

Otra alternativa para utilizar videoconferencia mediante FaceTime es a través del **ID de Apple** (dirección de correo electrónico), en lugar del número telefónico. Esto nos permitirá establecer una videollamada con equipos iPhone, iPod, iPad y Mac OS-X.

```
<a href="Facetime:myAppleID@myemail.com" data-role="button" data-
icon="skype" id="miBotonFTime">Llámame por Facetime</a>
```

Enviar SMS desde una WebApp

Desde los smartphones también es posible enviar mensajes de texto utilizando una WebApp como lanzador. Debemos tener en cuenta que, al igual que un llamado telefónico desde una página web, en los mensajes de texto será el usuario quien presione el botón **Enviar**.

La WebApp solo puede iniciar la ventana de un nuevo mensaje de texto, cargar el número telefónico del destinatario y el cuerpo del mensaje que se desea enviar. Esto se realiza de la siguiente manera:

```
<a href="sms:+5491123456789" data-role="button" data-icon="skype"
id="miBotonSMS">Enviar SMS</a>
```

Esta acción simplemente abrirá un nuevo mensaje de texto y cargará el número de teléfono del destinatario en nuestro equipo. Si queremos iniciar un SMS con un texto predefinido, debemos realizar lo siguiente:

```
<a href="sms:+5491123456789?body=Hola" data-role="button" data-
icon="skype" id="miBotonSMSconTexto">Enviar SMS con Texto</a>
```

Y si deseamos enviar un SMS a múltiples destinatarios, simplemente debemos separar los números de teléfono con una coma. Veamos un ejemplo a continuación:

```
<a href="sms:+5491123456789, +5491198765432?body=Hola%20
a%20los%20dos." data-role="button" data-icon="skype"
id="miBotonSMSconTexto">Enviar SMS con Texto</a>
```

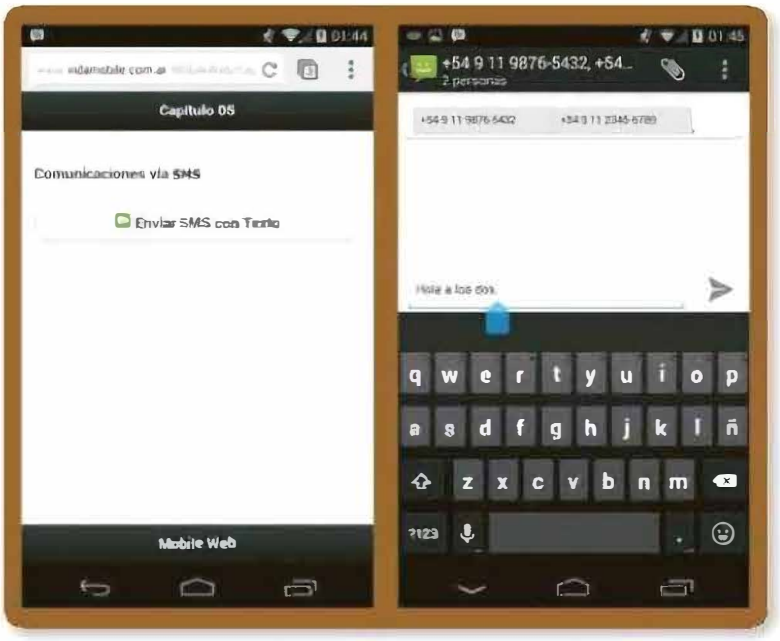


Figura 6. En este ejemplo podemos ver cómo, desde una WebApp, iniciamos un nuevo SMS con texto predefinido y dos teléfonos como destinatarios.

Mensajes a través de BBM

Durante el año 2014, BlackBerry (ex RIM) trasladó su clásico mensajero instantáneo **BlackBerry Messenger** (BBM) hacia la plataforma iOS y Android, con lo cual los antiguos usuarios de los dispositivos BlackBerry que cambiaron a otras plataformas pudieron seguir utilizando el consagrado mensajero BBM con su **PIN** original.

Y, por supuesto, al igual que el resto de los sistemas de mensajería, BBM lanzó, desde hace unos años, su propio URI scheme para poder iniciar desde una WebApp un chat o llamada de voz a través de este sistema de mensajería.

Veamos a continuación cómo iniciar un chat con un contacto de BBM desde un botón de jQuery Mobile:

WIDGETS EN JQUERY MOBILE

Los widgets incluidos en jQuery Mobile tienen una infinidad de funciones, atributos y eventos que pueden ser invocados tanto desde el HTML como a través de JavaScript. Es conveniente conocer toda su capacidad recurriendo a la documentación oficial: **http://api.jquerymobile.com/category/widgets**.

```
<a href="bbm://29949AD3?chat" data-role="button" id="btnBBM"><img
src="imagenes/bbm-icon.png" width="18px" height="18px"/> Blackberry Messen-
ger</a>
```

Limitación de BBM

BlackBerry Messenger permite establecer una comunicación con otro PIN solo cuando ambos interlocutores se aceptaron previamente con los usuarios. Por ello, cuando invoquemos un chat hacia un PIN de BlackBerry que no figura como contacto, se abrirá directamente la ventana de invitación de BBM.

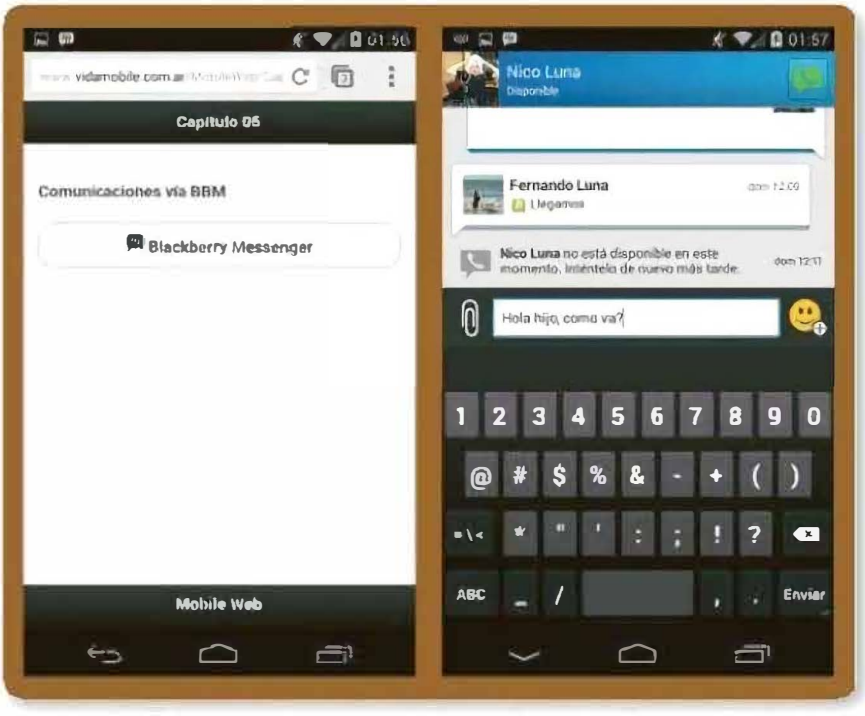


Figura 7. BlackBerry Messenger también puede ser invocado mediante URI scheme.

APIS DE COMUNICACIÓN

A lo largo del crecimiento de las plataformas móviles, más y más sitios web, apps nativas y plataformas de servicio web incluirán URI schemes dentro de la funcionalidad mobile. Es bueno recorrer los buscadores web de forma periódica para conocer a fondo los avances en este terreno.

Comunicación a través de redes sociales

En base al impacto de las redes sociales y al uso cotidiano que todos les damos, sumado a que casi todas las empresas consiguieron nuevos canales de comunicación directa con sus clientes a través de estas, las plataformas más populares llevaron su propio URI scheme a los navegadores de escritorio y móviles.

Perfil y mensajes en Twitter

Desde los dispositivos móviles podemos utilizar el URI scheme de Twitter para poder visualizar, con un simple tap, el perfil de una empresa o personalidad. Veamos un ejemplo de cómo invocar un perfil de Twitter desde una WebApp:

```
<a href="twitter://user?screen_name=mobilepadawan" data-role="button"
id="btnTwitter">@mobilepadawan en Twitter</a>
```

También contamos con la posibilidad de iniciar un mensaje con un texto predefinido, de la misma manera que lo hacemos con un e-mail o SMS:

```
<a
ref="twitter://post?message=Genial%20el%20sitio%20Vida%20Mobile%20
http://www.vidamobile.com.ar/.%20Powered%20by%20&#64mobilepadawan"
data-role="button" id="btnPostTwitter">Postear URL en Twitter</a>
```

Al igual que en los envíos de largos textos por e-mail a través de un hipervínculo o el envío de un SMS, debemos tener en cuenta que

TWITTER URI SCHEME

Al igual que el resto de los URI schemes vistos, el de Twitter dispone de una amplia cantidad de funciones que nos pueden llegar a ser útiles para desarrollar tanto webs móviles como apps nativas. Podemos acceder a estas funciones a través de la siguiente URL: **https://dev.twitter.com**.

necesitamos reemplazar algunos caracteres por sus equivalentes de la tabla de códigos HTML.

En este último ejemplo, utilizamos nuevamente la combinación **%20** para reemplazar los espacios y la combinación **&#64** para hacer la arroba correspondiente al nombre de usuario en Twitter.

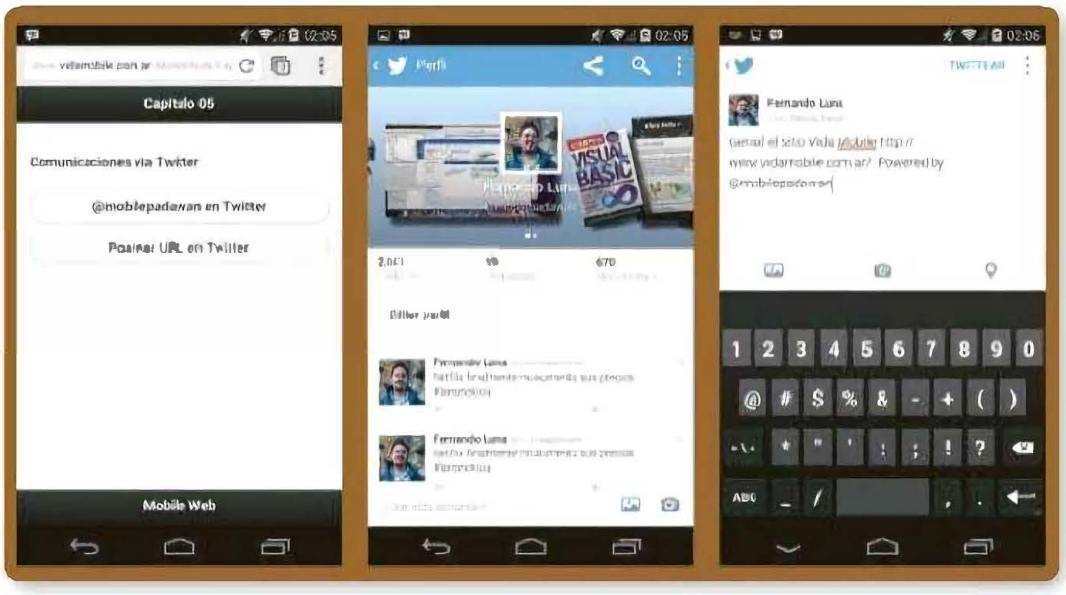


Figura 8. Twitter dispone de varias opciones de URI schemes para invocar, desde una WebApp o app nativa, diversas funciones de la red de microblogging.

Perfil y mensajes en Facebook

Al igual que en Twitter, también podemos invocar el perfil de Facebook de una persona o empresa desde un hipervínculo establecido en una WebApp. Veamos, a continuación, cómo visualizar un perfil de esta red social:

```
<a href="fb://profile/1363874497" data-role="button" id="btnFBProfile"><img
src="imagenes/fb-icon.png" width="18px" height="18px"/> Perfil de Facebook</a>
```

A diferencia de Twitter, para visualizar un perfil de Facebook debemos invocarlo a través de su código de identificación (**Facebook ID**). En sus inicios, esta red social manejaba ID de usuarios en lugar de un nombre personal o nickname. Con el tiempo instauró el nickname, pero esto no llegó a aplicarse en el URI scheme hasta el momento.

El ID de Facebook de cada persona o empresa se puede identificar en la URL de una foto publicada en el perfil. Abrimos un álbum fotográfico que esté publicado en el perfil que deseamos obtener y en la URL encontraremos el ID entre el último punto (.) y el final de la URL, que generalmente termina con **&type=3**.

También podemos invocar la comunicación a través del **sistema de mensajería de Facebook**. Podemos realizarlo de la siguiente manera:

```
<a href="fb://messaging/compose/1363874497" data-role="button"
id="btnFBProfile"><img src="imagenes/fb-icon.png" width="18px"
height="18px"/> Chat por Facebook</a>
```

Esto abrirá directamente el sistema de mensajería de Facebook en la pantalla del perfil especificado, para que el usuario escriba directamente el mensaje que desea enviar.

Alcance de los URI schemes

La mayoría de los URI schemes que nacieron estos últimos años lo hicieron para desplegar su potencial en las plataformas móviles. Hasta el momento, hemos repasado las siguientes:

- Llamado telefónico.
- Llamado por Skype.
- Videoconferencia con FaceTime.
- Enviar SMS.
- Enviar mensajes por BBM.
- Perfil y mensajes por Twitter.
- Perfil y mensajes por Facebook.

Estas funcionalidades fueron pensadas íntegramente para dispositivos móviles. De todas ellas, solo aquellos equipos de escritorio (Windows, Linux, Mac) que tengan un cliente Skype instalado podrán hacer uso del hipervínculo **Tel://** y también del hipervínculo **Skype://**.

El resto de los URI schemes funcionarán solo en las plataformas móviles. Por supuesto que, para que funcionen de manera óptima, deberá estar instalado en el equipo cada uno de los sistemas de mensajería a utilizar como pasarela de comunicación.

¿Y Facebook Messenger?

En el caso puntual de los URI schemes desarrollados para Facebook, cuando deseemos iniciar un chat utilizando esta tecnología, el dispositivo móvil determinará si el equipo tiene o no instalado **Facebook Messenger**. Si solo tiene la app de Facebook, se iniciará el chat a través de esta; si existe Facebook Messenger en el dispositivo, se abrirá y se iniciará la comunicación por chat desde esta plataforma.

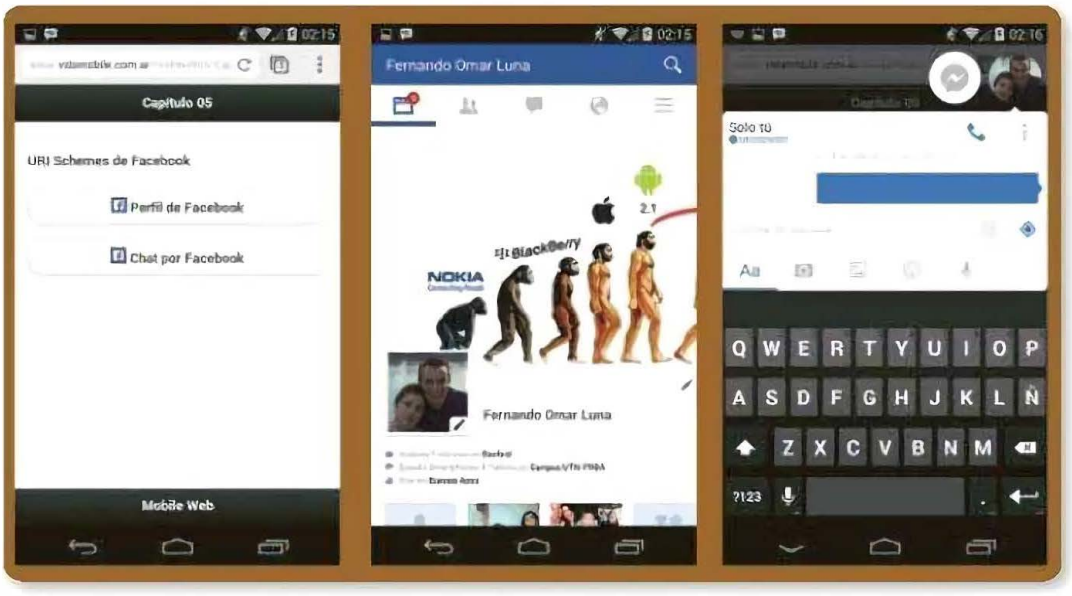


Figura 9. Facebook también dispone de una variada cantidad de opciones para utilizar funcionalidades de su plataforma vía URI schemes.

Mensajes con URI scheme de Twitter

A continuación, desarrollaremos un pequeño ejercicio para probar el concepto de URI scheme en la plataforma de Twitter. Para ello, iniciamos un nuevo proyecto en nuestro editor de código, creando una nueva página HTML a la que llamaremos **comentariosen140.html**.

API DE COMUNICACIÓN FACEBOOK

Al momento de utilizar el URI scheme para enviar mensajes a través de Facebook, este funciona de manera transparente para el usuario, priorizando el uso de Facebook Messenger en los equipos móviles. En caso de que esta app no se encuentre instalada, se utilizará la app de Facebook en su lugar.

DEBEMOS CONSIDERAR QUE TWITTER SOLO PERMITE PUBLICAR MENSAJES DE 140 CARACTERES

El proyecto consiste en simular, desde una página que dispone de un componente **Text Area**, cómo podemos invitar al usuario visitante a escribir su mensaje en esta sección, para luego redireccionarlo hacia la app de Twitter que tiene instalada en su dispositivo y, desde allí, publicarlo en su timeline.

Pero esto no solo involucra al llamado del URI scheme, sino que, para este desarrollo, debemos tener en cuenta que Twitter permite publicar mensajes de 140 caracteres solamente. Por tal razón, deberemos advertir al usuario si se pasa de esta cantidad y, si lo hace, no permitirle publicar su mensaje.

Vamos al código: lo primero que haremos es montar la estructura básica de una nueva página HTML.

```
<!DOCTYPE html>
<html>
<head>
   <title>Comentarios en 140</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
     <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>

</head>
<body>
   <div data-role="page" id="listas" data-theme="d">
     <div data-role="header" data-theme="c">
            <h4>Cap&iacute;tulo 05</h4>
     </div>
     <div data-role="content">
            <h4>Comentarios en 140</h4>
                  <p>D&eacute;janos tus comentarios por Twitter:</p>
```

```
            <div id="comentarios">
                <textarea id="txtComentarios" placeholder="Ingresa tu comen-
tario"></textarea>
            </div>
                <p id="resto">Restantes: 140</p>
                    <div data-role="button" id="btnEnviar" data-
inline="true" data-theme="a">Enviar</div>
            <input type="reset" id="btnLimpiar" data-inline="true"
value="Limpiar" />
        </div>
                <div data-role="footer" data-position="fixed" data-theme="c">
                <h4>Mobile Web</h4>
        </div>
    </div>
  </body>
  </html>
```

Esta estructura es simple. Tenemos una página realizada en jQuery Mobile con un título, un campo de texto amplio llamado **txtComentarios**, una leyenda que dice cuántos caracteres restantes nos quedan en el campo (englobada dentro de un **div** a la que llamamos **resto**), un botón **Enviar** y un botón **Limpiar.**

Esta página no tiene ninguna funcionalidad, por lo tanto, comenzaremos a agregársela. En el **<header>**, a continuación de la última sentencia **<script>**, abriremos un nuevo tag **<script></script>** para escribir algunas funciones en lenguaje JavaScript.

Vamos por la primera de ellas:

```
function limpiarCampos() {
    document.getElementById('txtComentarios').value = '';
    document.getElementById('resto').innerHTML = "Restantes: 140";
    }
```

La función **limpiarCampos()** nos permite borrar el contenido del Text Area **txtComentarios** y resetear el contenido del **div 'resto'.**

A continuación, agregamos una nueva función denominada **cuentaCaracteres()**, más compleja y extensa, que nos permitirá verificar cuántos caracteres está escribiendo el usuario y reflejar el número restante en el **div 'resto'**. El texto del **div** cambiará a color **rojo** para advertir que el usuario se ha pasado de la cantidad máxima permitida y, cuando vuelva a los parámetros normales, el mismo texto cambiará nuevamente al color **negro**.

```
function cuentaCaracteres(){
        var intCuenta = 0;
        var intTotal = 140;
        var strResto = document.getElementById('resto');
        var strComent = document.getElementById('txtComentarios');
        intCuenta = strComent.value.length;
        intResultado = intTotal - intCuenta;
                strResto.innerHTML = "Restantes: " + intResultado;
                if (intResultado < 0) {
                        strResto.style.color = '#FF0000';
                        }
                else {
                        strResto.style.color = '#000000';
                        }
                }
```

La variable **intCuenta** posee un valor inicial en **0**, la variable **intTotal** un valor inicial de **140**. **intCuenta** se irá incrementando a medida que agregamos caracteres en **txtComentarios**. Luego, **intResultado** tendrá el valor de **intTotal** menos **intCuenta**, y lo mostrará en el **div 'Resto'**.

LLAMADOS MEDIANTE WTAI

Al igual que las diversas funciones para realizar llamados, podemos aprovechar la funcionalidad **WTAI://**, que nos permite generar llamados que incluyan las codificaciones de tonos conocidas como **DTMF**. Esto proveerá soporte para comunicarnos a números telefónicos en los que el contestador automático nos atiende en una primera instancia.

Al final de la función, preguntamos a través de **if()** si el valor de **intResultado** es menor a cero. De ser así, mostrará el contenido del **div 'Resto'** en color rojo; de lo contrario, lo mostrará en color negro.

Vamos ahora por la última función, la cual nos permitirá (o no) iniciar la app de Twitter y cargar el texto ingresado en **txtComentarios** para publicarlo en nuestro timeline.

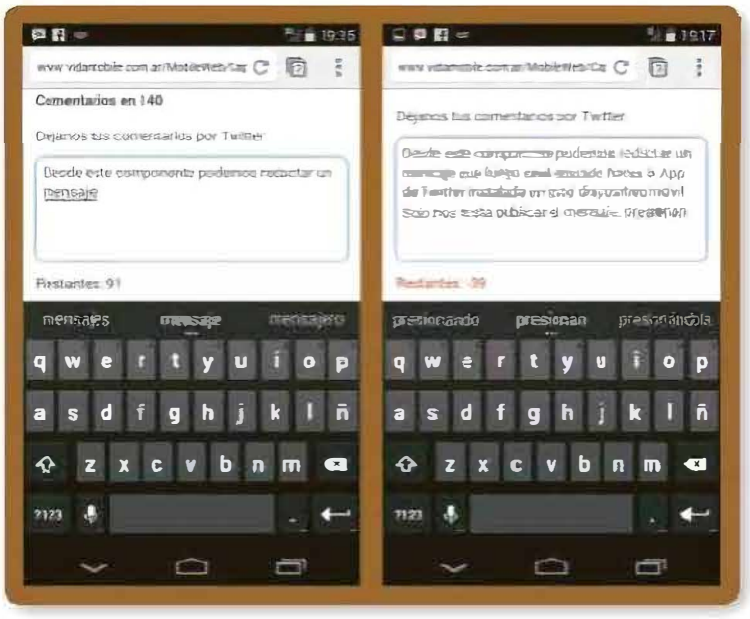


Figura 10. Al iniciar el tipeo de nuestro mensaje, automáticamente se inicia el contador de caracteres y, si nos pasamos, cambia el color del texto a rojo.

```
function enviarTwit() {
        var intCuenta = document.getElementById('txtComentarios').value.length;
        var comentarios = document.getElementById('txtComentarios').value;
        if (intCuenta > 140) {
                window.alert('Mensaje extenso.');
                }
        else {
                var msj = comentarios.replace(" ", "%20");
                location.href="twitter://post?message=" + msj;
                return;
                }
}
```

LA VARIABLE COMENTARIOS TOMA EL VALOR DE LO INGRESADO EN TXTCOMENTARIOS

En la función **enviarTwit()** encontramos la variable **intCuenta** que captura la cantidad de caracteres que posee **txtComentarios**. Luego, la variable **comentarios** toma el valor de los caracteres ingresados en **txtComentarios**. Si **intCuenta** supera los 140 caracteres, alertará al usuario de que el mensaje es extenso; si no, utilizará la función **replace()** para cambiar los espacios por el carácter equivalente y, por último, ejecutará el URI scheme para cargar el mensaje en Twitter.

Ya tenemos todas nuestras funciones JavaScript escritas. Ahora, vamos a ensamblarlas dentro de los componentes del HTML para que funcionen.

Ubicamos el botón **Limpiar** y le agregamos, en el evento **onClick**, la función **limpiarCampos()**.

```
<input type="reset" id="btnLimpiar" data-inline="true" value="Limpiar"
onClick="limpiarCampos()" />
```

Ahora, ubicamos el componente **Text Area**, al cual le agregamos la función **cuentaCaracteres()** en dos eventos: **onKeyUp()** y **onKeyDown()**.

```
<textarea id="txtComentarios" placeholder="Ingresa tu comentario"
onKeyDown="cuentaCaracteres()" onKeyUp="cuentaCaracteres()">
</textarea>
```

Por último, agregamos la función **enviarTwit()** al evento **onClick()** del botón **Enviar**. A diferencia del resto de las funciones incorporadas en los otros componentes, a esta función le antepondremos **return**.

```
<div data-role="button" id="btnEnviar" data-inline="true" data-theme="a"
onClick="return enviarTwit()">Enviar</div>
```

Dado que su acción invocará una aplicación externa, a través de la sentencia **return** permitiremos que la WebApp retome el control cuando salgamos de la aplicación invocada (en este caso, Twitter).

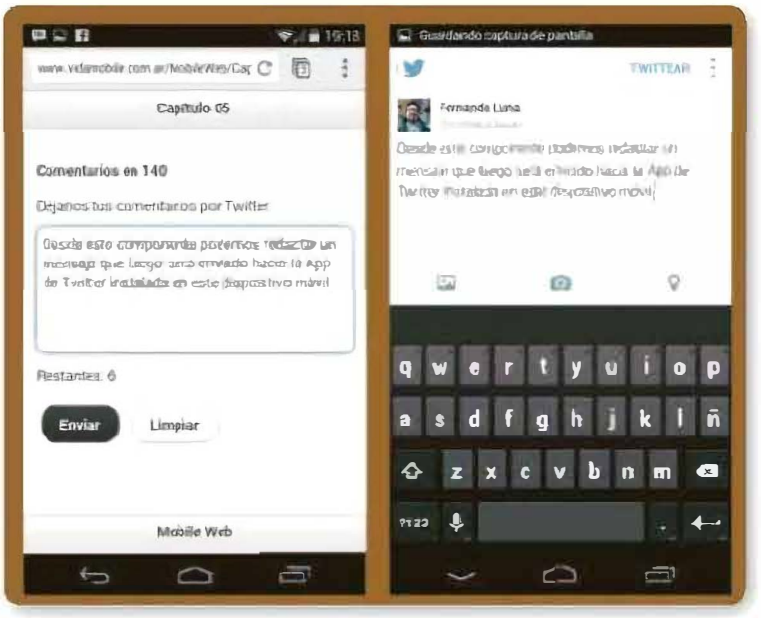


Figura 11. Una vez escrito el mensaje, solo nos resta presionar el botón **Enviar** para finalizar su publicación desde la app nativa de Twitter.

Comportamientos de los eventos según el dispositivo

Como hemos visto hasta ahora, el uso de URI schemes nos permite ejecutar diversas maneras de comunicación desde los dispositivos móviles. Pero vale aclarar nuevamente que, dependiendo del dispositivo móvil, algunas de estas tareas no se podrán llevar a cabo de la manera habitual.

Veamos, a continuación, cómo responde cada uno de los dispositivos móviles a los eventos a través de URI schemes.

Respuesta de eventos en tablets

En las tablets que existen actualmente se pueden realizar casi todos los eventos que repasamos hasta el momento. Sin embargo, debemos tener en cuenta que algunos de ellos no pueden llevarse a cabo porque la app o función con la cual nuestra web móvil debe interactuar no está disponible en el equipo.

Por ejemplo: las tablets, hasta el momento, no permiten descargar e instalar la aplicación BBM; por lo tanto, este evento no podrá llevarse a cabo desde una web móvil. Respecto a los eventos a realizar a través del URI scheme de Skype, dependerá de que esta app esté o no instalada en el equipo.

Figura 12. Invocación del URI scheme de BBM desde un iPad. En este caso, el control de error es realizado por el sistema operativo.

Lo mismo ocurrirá con los SMS: si la tablet no posee un medio de comunicación que permita el uso de SMS, la invocación al URI scheme podrá fallar y producir un error en la web. De igual modo, la última versión de la app **Hangouts**, que reemplaza al antiguo **GTalk**, integra una función que permite, en algunos dispositivos Android, utilizarla para el envío y recepción de SMS.

> SOLO SE PUEDE INVOCAR EL LLAMADO A FACETIME DESDE UNA WEB MÓVIL CARGADA EN IOS

Por el lado de FaceTime, esta plataforma solo está disponible para iOS, por lo tanto, cualquier botón que invoque el llamado a esta aplicación desde una web móvil cargada en Android, BlackBerry o Windows Phone no tendrá funcionalidad alguna. Por el lado de los URI schemes de Twitter y Facebook, estos funcionarán sin problema alguno desde cualquier dispositivo móvil, tablet o smartphone y prácticamente desde cualquier sistema operativo móvil, siempre y cuando la app esté instalada en el dispositivo.

En cuanto a la invocación de un llamado telefónico estándar, las tablets pueden responder de dos maneras distintas: si el sistema operativo es iOS, podrán desviar el llamado telefónico a través de la app FaceTime o incluso a través de Skype, si este está instalado.

Figura 13. El iPad, al no poder realizar un llamado, ofrece copiar el número, guardarlo como contacto o enviar un mensaje vía **iMessage**.

Si no tenemos opción de ninguna plataforma de comunicación en la tablet, solo nos mostrará la opción de agregar el número de teléfono a un contacto nuevo o existente de nuestra libreta de direcciones.

Respuesta de eventos en smartphones

Casi todos los eventos de los URI schemes vistos funcionarán de manera óptima en un smartphone. Es probable que, si diseñamos una página web que sea utilizada por un segmento corporativo, las apps de Facebook y Twitter no estén instaladas en los dispositivos móviles.

Por lo tanto, debemos verificar si es conveniente que el usuario valide (por ejemplo, a través de su perfil) si utiliza apps de redes sociales o no, para así poder controlar qué funciones sociales nuestra WebApp puede activar y cuáles no. Otra alternativa es establecer un control de errores cuando nuestra página web ejecute una acción que da error, ya que esto ocurre comúnmente con la invocación de URI schemes específicos.

Una solución viable

Una de las soluciones a implementar en el desarrollo de un sistema web móvil con la invocación a URI schemes es contemplar el tipo de plataforma donde la web se está ejecutando y, a través de un listado de equivalencias, visualizar u ocultar los botones de selección.

Para el control de visualización de un hipervínculo a llamados o mensajes de chat de BBM, podemos controlar el tipo de sistema operativo donde se cargó la página. Si fue cargada en un iPad, en un iPod o, tal vez, en Windows Phone, sabremos que BBM no está disponible para estas plataformas; por lo tanto, podremos evitar mostrar el hipervínculo.

Existen alternativas en la Web, tanto gratuitas como pagas, que nos permitirán detectar el tipo de dispositivo donde la página web se carga y, en base a una serie de reglas automáticas, mostrar u ocultar los botones que permiten establecer una comunicación.

Invocar llamados y mensajes de texto

A continuación, realizaremos un ejercicio donde delinearemos un perfil de usuario y agregaremos en él los botones necesarios para iniciar una comunicación a través de distintos medios. Para poder llevar a cabo el ejercicio, deberemos descargar el archivo **Cap05. DoctorAssistance.rar** y descomprimir su contenido en una carpeta. Luego, desde Dreamweaver (o nuestro editor favorito) completaremos este ejercicio con las páginas faltantes.

LLAMADOS DE AUDIO Y VIDEO

Son pocos los países que han implementado un sistema de comunicación de audio y video en los teléfonos móviles; para estos, podemos utilizar el hipervínculo **<href="av:+numerodetelefono">**. Para implementar esta funcionalidad, el dispositivo no solo debe estar activo en los proveedores de telefonía móvil, sino también contar con una velocidad 3G (como mínimo) en la red de datos.

Nuestro nuevo proyecto, **Doctor Assistance**, brinda prestaciones de salud con diversos profesionales médicos, a quienes, a través de esta WebApp, podremos contactar de manera directa.

La página **index.html** tiene una introducción ficticia de la compañía, y la página **prestaciones.html** muestra diferentes especialidades clínicas que el centro médico atiende. En **prestaciones.html** podemos ver el uso de un nuevo widget de jQuery Mobile que simula un acordeón con contenido diverso.

Nuestro objetivo será incorporar un hipervínculo debajo de cada prestación, mediante el cual, al hacer clic, se desplegará un listado con médicos donde veremos al profesional y las diversas maneras en que podremos comunicarnos con él.

Para ello, primero debemos crear un **ListView** con la información resumida de cada profesional. Luego, al presionar sobre este, podremos acceder al detalle de cada médico y ponernos en comunicación con él a través de diferentes vías.

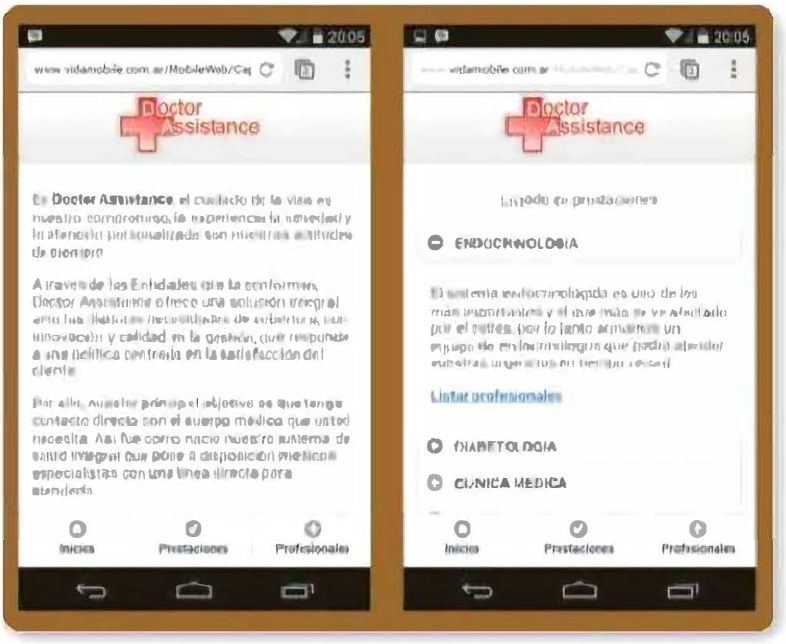


Figura 14. Esta es la WebApp que completaremos, integrándole la funcionalidad de comunicaciones vista a lo largo de este capítulo.

Extender el uso de ListView

Para diseñar la página donde listaremos a los profesionales, recurriremos al uso del widget **ListView**, conocido en el **Capítulo 4**, con

una pequeña diferencia en su desarrollo. Incluiremos una variante de **Listview** que nos permitirá incluir una imagen y contenidos más extensos.

Modifiquemos entonces el archivo **endocrino.html** ubicado dentro de la carpeta del proyecto descargado. Al abrirlo, solo encontraremos una página vacía sin contenido en el apartado **content**. Agreguemos lo siguiente:

```
<p align="center"><strong>Endocrinología</strong></p>
  <div class="content-primary">
    <ul data-role="listview" data-corners="true">
      <li><a href="drnico.html" data-rel="Dialog">
        <img src="imagenes/drnico.png"/>
        <h3>Dr. Nicol&aacute;s Moreno</h3>
        <p>Especialista en endocrinología</p>
      </a></li>
    </ul>
    <ul data-role="listview" data-corners="true">
      <li><a href="drjuly.html" data-rel="Dialog">
        <img src="imagenes/drjuly.png"/>
        <h3>Dr. Juli&aacute;n Conte</h3>
        <p>Especialista en endocrinología infantil</p>
      </a></li>
    </ul>
</div>
```

A través de esta porción de código, agregamos dentro del cuerpo de la página un **ListView** del tipo **Thumbnail View**. La estructura es similar a la del **ListView** visto en el capítulo anterior, con al agregado de que su interior posee un formato diverso. En él, encontraremos una imagen, un título y una breve descripción.

La incorporación del hipervínculo se suma a la de una imagen, utilizando el tag **<img>**. Además, un tag **<h3>** para el título y un **<p>** para la descripción.

En el hipervínculo de ambos **ListItems** se observa el atributo **data-rel="Dialog"**, el cual desplegará el detalle del profesional junto con sus vías de comunicación en una ventana emergente.

Veamos el código de la página **drnico.html**:

```
<!DOCTYPE html>
<html>
<head>
  <title>Doctor Assistance - Profesionales Endocrinología</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
  <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
</head>
<body>
  <div data-role="page" id="contenido" data-theme="c">
    <div data-role="header" >
                    <p align="center">
      <strong>Dr. Nicol&aacute;s Moreno - Endocrin&oacute;logo</strong>
      </p>
    </div>
    <div data-role="content" id="contenido">
      <div id="imagen" align="center"><img src="imagenes/drnico.png" /></
div>
      <a href="tel:+5491187654321" data-role="button" data-
corners="false"><img src="imagenes/phone-icon.png" width="18px"
height="18px" /> Tel&eacute;fono móvil</a>
      <a href="sms:+5491187654321" data-role="button" data-
corners="false"><img src="imagenes/sms-icon.png" width="18px" height="18px"
/> SMS al móvil</a>

    </div>
  </div>
</body>
</html>
```

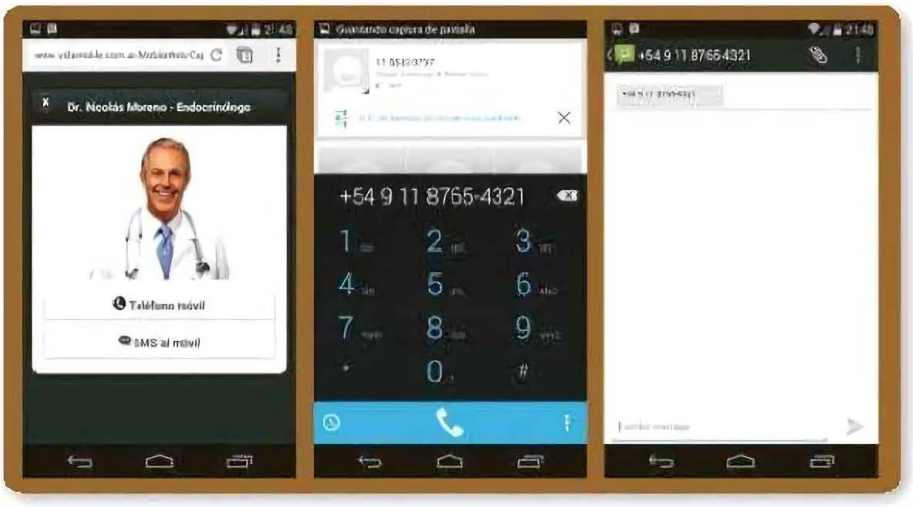


Figura 15. El resultado de la visualización del primer profesional y la acción de llamar por teléfono o enviar un SMS.

Y ahora vamos por el código del otro profesional, el cual se creará a través de la página **drjuly.html**:

```
<!DOCTYPE html>
<html>
<head>
   <title>Doctor Assistance - Profesionales Endocrinología</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
</head>
<body>
   <div data-role="page" id="contenido" data-theme="c">
     <div data-role="header" >
                        <p align="center">
       <strong>Dr. Juli&aacute;n Conte - Endocrin&oacute;logo infantil</
strong>
       </p>
     </div>
     <div data-role="content" id="contenido">
       <div id="imagen" align="center"><img src="imagenes/drjuly.png" /></
```

```
div>
            <a href="skype:echo123?call" data-role="button" data-
corners="false"><img src="imagenes/skype-icon.png" width="18px"
height="18px" />  Llamar por Skype</a>
            <a href="bbm://BBMPIN?chat" data-role="button" data-
corners="false"><img src="imagenes/bbm-icon.png" width="18px" height="18px"
/> Blackberry Messenger</a>
          </div>
        </div>
    </body>
    </html>
```

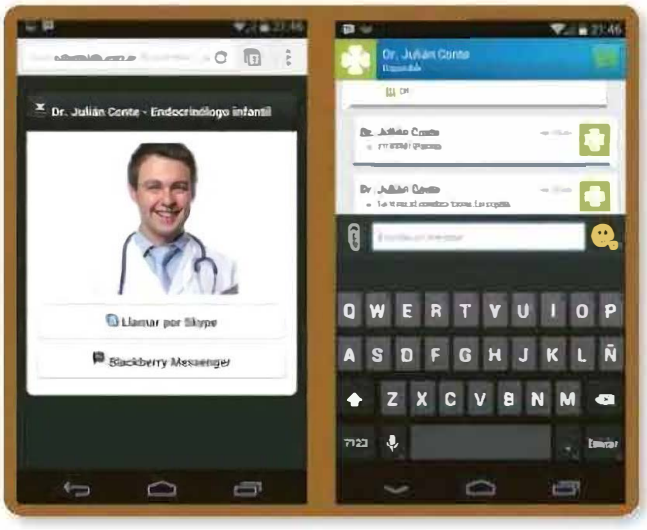


Figura 16. El segundo de los profesionales habilita la comunicación a través de Skype o BBM.

RESUMEN

A lo largo de este capítulo, aprendimos a invocar los diferentes sistemas de comunicación que podemos utilizar desde un teléfono móvil o tablet. Llamados telefónicos, SMS, Skype, FaceTime y BBM son algunas de las opciones que podemos aprovechar para entablar una comunicación entre dos o más partes. También, el uso de URI schemes nos permitió aprovechar las características de las redes sociales para entablar mensajes entre usuarios. Y, por supuesto, seguimos explorando más características de los widgets propuestos por jQuery Mobile para hacer más atractivos nuestros portales móviles.

Actividades

TEST DE AUTOEVALUACIÓN

1 ¿Para qué sirve **ListView**?

2 ¿Qué atributos podemos utilizar para crear un ítem **ListView**?

3 ¿Qué parámetros se pueden utilizar junto al hipervínculo SMS?

4 ¿Cómo podemos personalizar la imagen de un widget **button**?

5 ¿Cómo deshabilitamos un widget **button**?

EJERCICIOS PRÁCTICOS

1 Investigue, a través de internet, qué otros URI schemes existen.

2 Investigue los diferentes URI schemes de Facebook. Idee un widget **button** adicional que dispare otro esquema que aproveche algún atributo nuevo.

3 Investigue en **jquerymobile.com** acerca de **Collapsible Set** e incluya alguna nueva funcionalidad en el proyecto desarrollado en este capítulo.

4 Utilice la página **profesionales.html** y cree un **ListView** alfabético de al menos seis profesionales, incluyendo un link a su perfil.

5 Utilice algún simulador o emulador móvil y testee en él los ejercicios realizados en este capítulo.

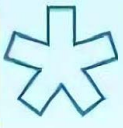
PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*06

Lenguajes de programación

A lo largo de este capítulo repasaremos el proceso de creación de WebApps dinámicas aprovechando el potencial que nos brindan los lenguajes de programación web. A través de estos, aprenderemos a trabajar con información almacenada en una base de datos, a grabar registros y a generar pantallas dinámicas integrando los widgets de jQuery Mobile desde el lenguaje PHP.



↗↗↗

Evolución

A lo largo de la historia de la informática, los lenguajes de programación jugaron un papel importante y fueron adaptándose, durante las nuevas generaciones, a los avances en estética, al funcionamiento de los sistemas operativos y a la integración de nuevos servicios, como lo fue el nacimiento de la **WWW**.

Los primeros lenguajes de programación utilizaban editores básicos (generalmente, los mismos editores de texto), y luego para desplegar un programa se recurría a compiladores específicos para cada sistema operativo. El mundo evolucionó, y con la llegada de Windows se integraron diversas plataformas que brindaban un IDE completo que reconocía errores en la escritura de sentencias, funciones y declaración de variables. Estos IDE, además, permitían el desarrollo de las interfaces visuales de una manera mucho más amigable.

Luego, con la llegada de la Web, los IDE más populares debieron readaptarse para soportar múltiples lenguajes de programación y, a su vez, poder integrarlos en una única solución. Esto es lo que ocurrió con los editores de código como **Eclipse**, **Netbeans**, **Dreamweaver**, que daban soporte para desarrollo de soluciones basadas en Web.

Estos editores debieron adaptarse, de forma nativa o a través de plugins, para detectar y soportar código **HTML**, **CSS**, **JavaScript**, **PHP**, **C#**, **Visual Basic**, **VBScript**, **Ajax**, **CGI** y otros tantos lenguajes más que hoy se utilizan a diario en el desarrollo web.

La importancia de lo dinámico

En muchos casos, la Web promedio fue orientada a lo corporativo; por lo tanto, los cambios que en esta ocurrieron fueron mínimos y, hasta en muchos casos, nulos. Pero con el nacimiento, hace casi una década, de la llamada **Web 2.0**, de los **blogs** y de la integración de contenido multimedia en los sitios, muchas páginas web, hasta incluso las más sencillas, debieron mudar su comodidad estática hacia la generación de contenido dinámico, lo que permite asegurar que los usuarios volverán reiteradas veces en busca de nuevos contenidos.

También las redes sociales se integraron a las webs particulares. De esta manera, se generan contenidos compartidos, tanto en una

web empresarial como en el rincón social de esta. Por eso, cuando pensamos el desarrollo de una página web, sea de escritorio o móvil, es conveniente delinearla de forma dinámica desde su versión prototipo, para que sea posible cambiar su contenido en el momento en que sea necesario.

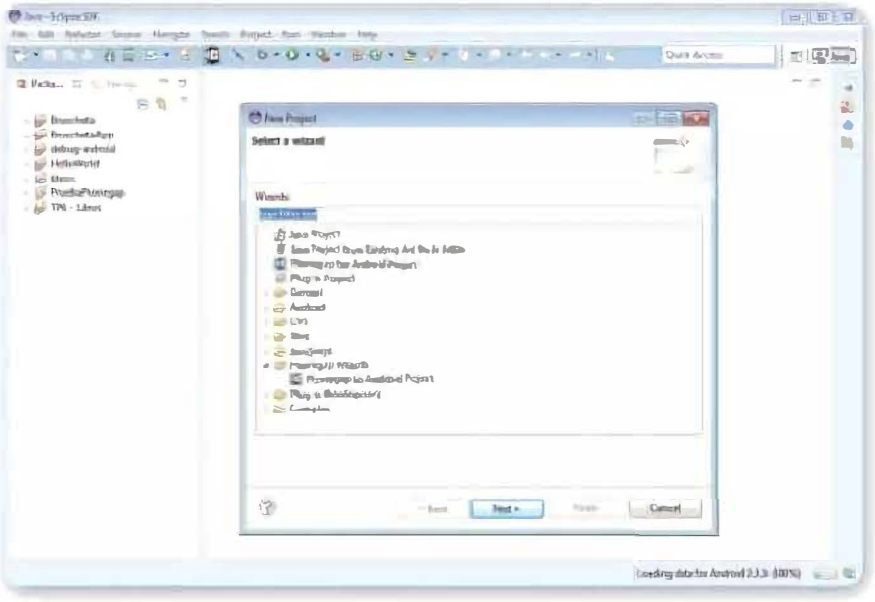


Figura 1. Eclipse es uno de los entornos de desarrollo que soporta una importante variedad de lenguajes de desarrollo.

Las plataformas más comunes

En el terreno web, podemos encontrar decenas de plataformas móviles que pueden adaptarse rápidamente a mostrar un contenido dinámico. En cuanto a lenguajes de desarrollo, este terreno está mucho más acotado, y hasta se podría decir que limitado a dos o tres grandes jugadores. Entre ellos, podemos mencionar las plataformas **.NET**, **PHP** y –algo más lejos y casi olvidado– **JSP**.

En cuanto a implementación del lado del servidor, los proveedores de hosting brindan planes que soportan la plataforma PHP y, en menor medida, **ASP.Net**. Entre ambas plataformas, PHP es adaptable prácticamente a cualquier servidor web existente, sin importar el sistema operativo que este tenga, y permite también conectarse a un amplio abanico de bases de datos casi de forma transparente.

ASP.Net, por su parte, requiere para su implementación que el servidor web esté montado sobre una plataforma Windows y corra

ASP.NET ESTÁ ASOCIADO A SQL SERVER, MIENTRAS QUE PHP ESTÁ ASOCIADO A MYSQL

el servidor web **IIS**. Si bien también permite conectividad a una cantidad importante de fuentes de datos, para muchos casos se requiere disponer de drivers de terceros, instalados en el servidor, que permitan iniciar la conexión, consulta y almacenamiento de información en las bases de datos.

En la relación lenguaje de programación/base de datos, podemos asociar rápidamente a ASP.Net con bases de datos **SQL Server**, y a PHP con bases de datos **MySQL**. Estas duplas son casi inamovibles, a pesar de que cada plataforma asegura brindar conectividad con su contraparte.

A lo largo de este capítulo, explicaremos cómo funciona el lenguaje PHP y también enfocaremos nuestros ejemplos en esta plataforma. La elección de este lenguaje se debe a que casi todos los servidores de hosting lo soportan, y algunos de ellos brindan una cuenta de usuario gratuita en la cual podremos contar con PHP como lenguaje soporte y bases de datos MySQL.

¿Qué es PHP?

PHP es un lenguaje de programación de uso general que proporciona la codificación de soluciones del lado del servidor. Fue creado en 1995 por **Rasmus Lerdorf** para uso personal. Se diseñó con el fin de proveer contenido web dinámico, y fue pensado desde sus inicios para poder visualizar la información solicitada al servidor dentro del contenido HTML sin necesidad de tener que llamar a un archivo o libreria externo que procese los datos solicitados.

PRIMERAS VERSIONES DE PHP

Dado que PHP nació de un desarrollo personalizado, las primeras versiones de este lenguaje aceptaban tanto las extensiones de archivo terminadas en **.PHP** como las extensiones de archivo **.PHP2** o **.PHP3**, donde originalmente se indicaba con qué versión del lenguaje había sido desarrollada. A partir de la versión 4.0, esto quedó obsoleto.

El código de este lenguaje es procesado por el servidor web con un módulo de procesamiento PHP, el cual se encarga de generar una página HTML pura que luego se entrega al servidor web. Todas sus versiones fueron desplegadas de forma gratuita como software libre, bajo la licencia **GNU/Public**.

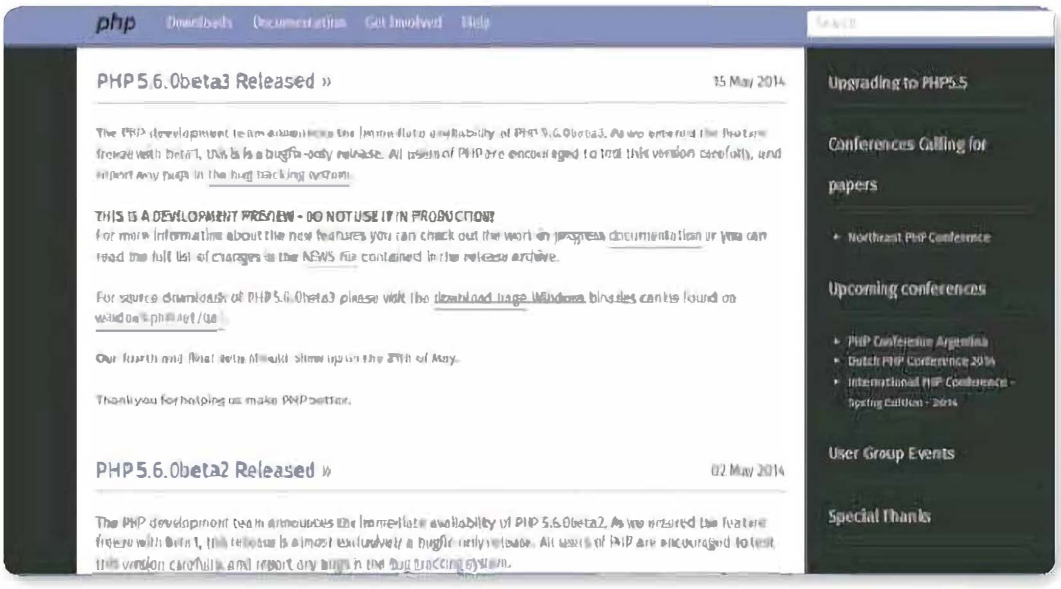


Figura 2. PHP es un lenguaje web popular que nació de un proyecto individual y que actualmente está presente en gran parte de la Web.

Simpleza en su sintaxis

PHP posee una sintaxis simple y muy fácil de implementar en cualquier desarrollo web. La sintaxis propia de PHP puede integrarse sin problemas dentro del código HTML que compone una página. A su vez, el mismo lenguaje puede generar código HTML dinámico, definiéndolo entre sus sentencias, para visualizar los resultados solicitados con la menor complejidad posible.

PHP en el mundo mobile

El lenguaje PHP se caracteriza por ser uno de los más veloces en procesar peticiones en el servidor y devolver resultados formateados en HTML al usuario. Este lenguaje no tiene una versión pensada para el mundo mobile, pero, dada su velocidad de respuesta, se ha vuelto el lenguaje ideal para implementar en soluciones web móviles.

Combinar tecnologías en una web móvil

La introducción de PHP en un desarrollo web implica cambiar las páginas con extensión **HTML** por la extensión **PHP**, propia del lenguaje. Esto permitirá que el servidor interprete que la página web solicitada posee contenido PHP y, por lo tanto, que la procese para obtener el resultado solicitado del código PHP incluido en dicha página.

> INTRODUCIR PHP EN UN DESARROLLO WEB IMPLICA CAMBIAR LA EXTENSIÓN HTML POR LA EXTENSIÓN PHP

En los ejemplos realizados en los capítulos anteriores, HTML predominó por sobre cualquier otro lenguaje de marcado o de programación. A continuación, haremos que las diferentes tecnologías repasadas hasta ahora puedan convivir en un único desarrollo. Aprovecharemos el lenguaje PHP para generar los tags y el código dinámico necesario que nos permitirá abandonar la web estática, tanto en la visualización de texto e imágenes obtenidos desde una base de datos MySQL como en la generación de widgets propios de jQuery Mobile.

Comandos básicos del lenguaje

La utilización de código PHP en una página HTML se puede realizar de dos maneras diferentes. La primera es generando todo el contenido HTML desde PHP, y la segunda, generando el contenido básico de la página en HTML y solo el contenido PHP dentro de una o más funciones locales o remotas, para integrarlo luego en las páginas HTML con los tags correspondientes.

Veamos, a continuación, algunos de los comandos más básicos de este lenguaje, para comprender mejor su concepto y uso cotidiano dentro de una página web.

EDITORES DE CÓDIGO

La mayoría de los editores de código gratuitos, como **Notepad++**, **Eclipse** o **NetBeans**, poseen reconocimiento de sintaxis PHP de manera nativa o a través de un plugin de fácil instalación en la plataforma. Dreamweaver, en todas sus versiones, también reconoce de manera nativa la sintaxis de este lenguaje.

Invocar un código PHP

La invocación del código PHP siempre se debe realizar entre **<?** y **?>**. Todo lo que se escriba dentro de este tag será interpretado y resuelto por el motor de PHP instalado en el servidor web y visualizado luego en la página HTML de forma transparente.

Cada línea de comando que escribimos en PHP debe ser finalizada por el punto y coma, al igual que se hace en el lenguaje **C** y sus variantes.

Mostrar resultados con echo

Cuando queramos escribir un texto específico que deba ser visualizado en pantalla, dentro del código PHP, debemos utilizar la sentencia: **echo 'texto a visualizar';**.

```
<body>
    <div id="miTextoPHP">
        <? Echo "Hola mundo. Esto es PHP!"; ?>
    </div>
</body>
```

Declaración de variables

A diferencia de otros lenguajes, PHP permite la declaración de variables de manera libre, sin un tipo específico. Por lo tanto, es posible que una misma variable pueda contener tanto números como letras o valores booleanos. Al definir una variable, se debe anteponer el símbolo **($)** en su nombre.

Veamos un ejemplo de esto:

```
<?
    $intNumero = 3;
    $strAlfa = "Esto es un texto";
    echo $intNumero;
    echo "<br/>";
    echo $strAlfa;
?>
```

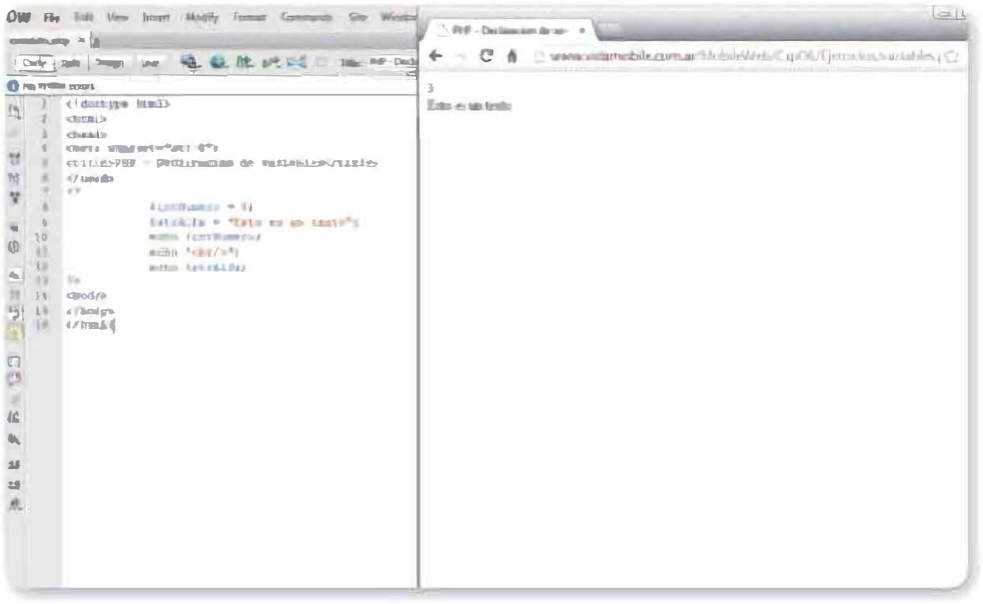


Figura 3. La declaración de variables y su representación en pantalla en tan solo escasas líneas de código.

Sentencia If

La sentencia **if** permite definir que una condición se cumpla y, de ser así, se ejecute determinada porción de código. Veamos un ejemplo de ello:

```
<?
        If ($intNumero = 3) {
      Echo "Este es el número tres.";
        }
?>
```

If – Else

La sentencia **If – Else** permite definir, a través de **if**, la ejecución de una porción de código ante una determinada condición y, a través de **else**, otra porción de código, si la condición definida en **if** no se cumple.

```
<?
  if ($intNumero == 3) {
```

```
        echo "Este es el número tres";
    } else {
        echo "No se definió un valor en la variable";
    }
?>
```

Do – While

La sentencia **do** permite ejecutar una determinada acción hasta que **while** cumpla con una condición determinada. Mientras **while** no cumpla la condición, **do** seguirá ejecutando la porción de código que tiene definida. En esta expresión, la validación que finaliza la ejecución de **do** se realiza al final.

```
<?
    $j = 0;
    Do {
        $j ++;
        echo "El valor de la variable $j es menor a mil.";
            echo "<br>";
    } While ($j < 999);
?>
```

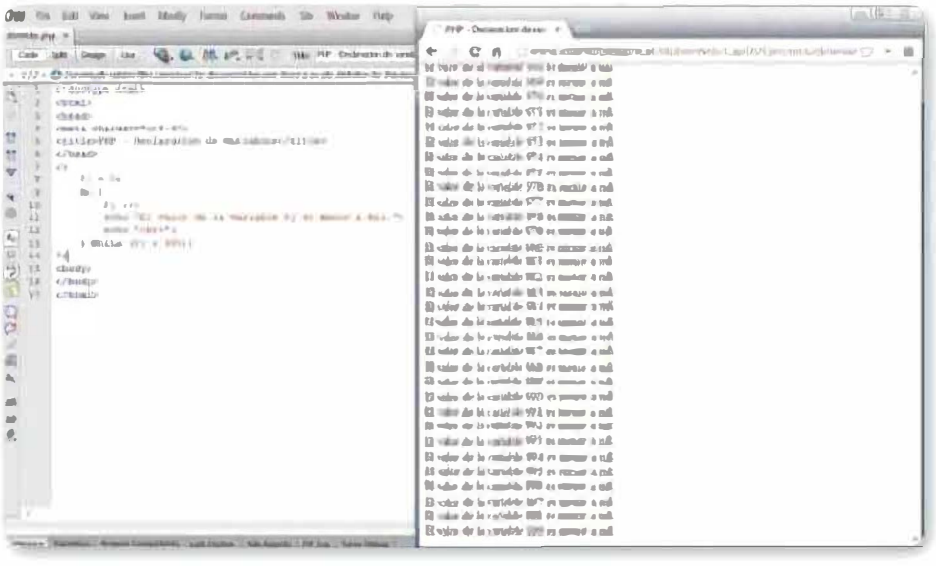


Figura 4. El bucle **Do – While** en acción, escribiendo cada recorrido realizado en la página HTML.

While

Esta sentencia genera un bucle que ejecuta determinado código mientras una condición específica no se cumpla. Al momento en que se cumple, el bucle finaliza.

```
<?
$f = 0;
While ($f <= 100) {
   Echo $f++;
}
echo "La variable $j alcanzó el valor Cien.";
?>
```

For

La sentencia **for** trabaja, igual que en el lenguaje C, evaluando tres condiciones. La primera condición se ejecuta al inicio del bucle de manera incondicional. Cuando vuelve a empezar el ciclo, se evalúa la segunda condición, la cual devuelve un valor **TRUE** para que el ciclo continúe, o, de lo contrario, el bucle finalizará.

La tercera condición es evaluada cada vez que finaliza la ejecución de cada bucle contenido en **for**. Veamos un ejemplo de esta sentencia:

```
<?
   For ($f = 0; $f <= 100; $f++) {
      Echo $f;
   }
?>
```

El código de ejemplo indica que la variable **$f** posee un valor inicial de **0** (cero). El ciclo **for** se ejecutará mientras la variable **$f** sea menor o igual a **100**, sumando, en cada iteración, una unidad a **$f** a través de la tercera expresión.

Funciones en PHP

Al igual que en otros lenguajes, PHP permite establecer funciones para que puedan ser utilizadas en más de una sección de la página

web, evitando así repetir el código y, a su vez, estructurándolo de manera prolija.

```
Function Calcular()
  {
            $f = 1;
            $j = 2;
            $n = $f + $j;
            return $n;
  }
```

Estas funciones pueden o no recibir uno o más parámetros para ser procesados dentro de la función, y estos parámetros, a su vez, pueden devolver un resultado. Veamos cómo hacerlo:

```
Function CalculoDinamico($parametro1, $parametro2)
  {
  //Validamos recibir números para poder realizar la suma
  If (is_numeric($parametro1) && is_numeric($parametro2)) {
    $n = $parametro1 + $parametro2;
    }
    $n = $parametro1 + $parametro2;
    Return $n;
  }
```

Include

La sentencia **include** permite agregar archivos con contenido puro en PHP a la página web actual para, de esta manera, utilizar dicho contenido y así poder realizar determinadas operaciones.

CREAR PÁGINAS WEB EN PHP DESDE CERO

PHP es un lenguaje de programación que, mediante la sentencia **echo**, puede reproducir sin problema cualquier sintaxis de tipo HTML o JavaScript. A su vez, esta última también puede invocar sintaxis de PHP a través de su funcionalidad **innerHTML**.

LAS BASES DE DATOS SON FUNDAMENTALES EN LA CREACION DE SITIOS WEB DINÁMICOS

Por ejemplo: cuando creamos funciones que deben procesar algo y devolver un resultado, estas habitualmente se crean en un archivo definido que será incluido en cada una de las páginas que requieran consumir estas funciones.

Todo el contenido que se crea en este archivo tiene carácter global; por lo tanto, si definimos variables con valores determinados, al utilizarlas desde una página web que incluya este archivo, la variable podrá consumirse con el valor que ya tiene definido.

Veamos un ejemplo a continuación. Para ello, crearemos un archivo al que llamaremos **variablefruta.php**. En su interior, agregaremos el siguiente código:

```
<?
   //Archivo variables.php
   $fruta = "Arándanos";
   $color = "Azul";
?>
```

Ahora crearemos un archivo PHP con contenido HTML, donde agregaremos la referencia al archivo **usodeinclude.php**:

```
<html>
<? include 'variablefruta.php'; ?>
<head>
<!-- contenido de header -->
</head>
<body>
<div id="frutadescripcion">
   <p>Fruta: <? echo $fruta; ?></p>
   <p>Color: <? echo $color; ?></p>
</div>
</body>
</html>
```

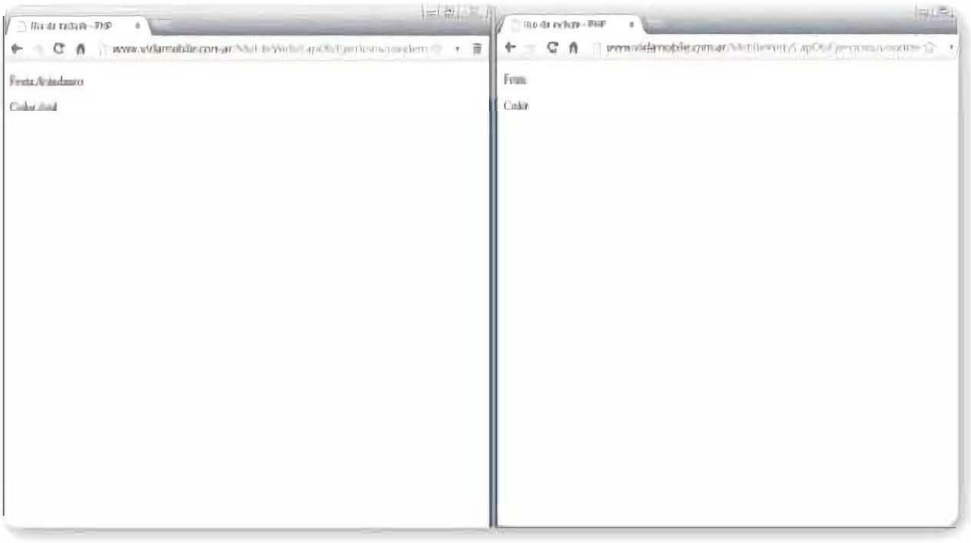


Figura 5. El resultado del ejemplo realizado con **`include`**, mostrando el valor de las variables, y el mismo resultado eliminando la página **`variablefruta.php`**.

Require

La sentencia **require** es idéntica a la sentencia **include**, con la diferencia de que la primera detendrá la carga de la página si el archivo que se desea incluir no existe físicamente en la ruta especificada. **Include** solo visualiza un error en la línea que procesa la inclusión del archivo, pero permite seguir ejecutando el resto del código PHP que se encuentra en la página.

La sentencia de **require** es la siguiente:

```
<? require 'variables.php' ?>
```

Base de datos

En la creación de sitios web dinámicos, las bases de datos juegan un papel fundamental, almacenando gran parte del contenido que suele mostrarse en estos sitios. A continuación, haremos una breve introducción a **MySQL**, un sistema de base de datos relacional que es el fiel compañero de PHP en casi todos los sitios donde este lenguaje es protagonista.

MySQL

MySQL es un sistema de gestión de bases de datos relacional desarrollado por la empresa **MySQL AB**, que desde enero de 2008 es una subsidiaria de **Sun Microsystems**, la cual pertenece, a su vez, a **Oracle Corporation** desde principios de 2010. El desarrollo de MySQL está basado en una licencia **GNU GPL**, excepto para las empresas que requieran incorporar esta base de datos para distribuirla con productos cerrados o licenciados, para las cuales sí se requiere el pago de una licencia por su uso.

Entre los sitios web más populares que utilizan este motor de base de datos destacamos **Google**, **Facebook**, **Wikipedia**, **Twitter** y **YouTube**. A su vez, la plataforma **Wordpress** también utiliza como motor de datos la base MySQL para el almacenamiento del contenido de cada blog que se genere con esta tecnología.

> MYSQL POSEE SOPORTE PARA CASI TODOS LOS SISTEMAS OPERATIVOS MÁS POPULARES, COMO LINUX Y WINDOWS

MySQL posee soporte para casi todos los sistemas operativos más populares, entre los cuales podemos encontrar a **Linux**, **BSD**, **OS-X**, y **Windows** (desde su versión 95 en adelante). Por lo tanto, cualquiera sea el sistema operativo que utilicemos en el servidor de nuestra página web, este dispondrá de una instalación de este sistema de base de datos.

Como bien dijimos, MySQL se complementa con PHP casi de manera transparente, aunque la base de datos se inicia a través de un servicio por línea de comandos. Igualmente, existen soluciones como **PHPMyADMIN**, que nos permite manipular todas las tareas habituales en una base de datos, de manera gráfica. La mayoría de los servidores web que prestan soporte para PHP y MySQL tienen instalado, por defecto, el gestor de base de datos PHPMyADMIN.

TIPO DE SINTAXIS A INVOCAR DESDE PHP

PHP permite no solo crear, mediante la sentencia **echo**, sintaxis propia de HTML, sino que también permite invocar sintaxis para la creación de input types o bloques de código HTML que requieren de la intervención de CSS.

Si no poseemos actualmente un servidor que nos brinde acceso a PHP y MySQL, podemos descargar e instalar de manera gráfica la suite **XAMPP** (Apache + MySQL + PHP + Perl) que nos permitirá, en poco tiempo, montar un sitio web con soporte para todas estas tecnologías. XAMPP dispone de versiones para Windows, Linux y OS-X en su sitio web oficial: **https://apachefriends.org/es/index.html**.

Figura 6. En el sitio web de MySQL (**www.mysql.com**) podemos encontrar la información oficial de esta base de datos y un manual para el usuario.

Comandos básicos SQL

El **lenguaje SQL** (*Structured Query Language*) es utilizado también en MySQL para realizar las operaciones básicas en la base de datos (crear bases de datos, tablas, usuarios, listar registros almacenados en tablas, agregar, eliminar, actualizar, etcétera).

Si no conocemos en profundidad el lenguaje SQL, para manipular los ejercicios que nos restan hacer, podemos tomar la referencia básica que nos provee el sitio **W3Schools**, donde encontraremos un excelente tutorial de MySQL y todos sus comandos: **www.w3schools.com/sql**.

También, a través de esta misma editorial, podemos acceder a una diversidad de libros que tratan el lenguaje PHP y MySQL con mayor profundidad y que nos serán de gran utilidad si deseamos llevar adelante un sistema amplio en funcionalidades. Accedemos a estos libros a través de la siguiente URL: **http://usershop.redusers.com/libros.asp?categoria=php**.

Y si preferimos la versión electrónica de estos libros, podemos acceder a través del siguiente link: **http://usershop.redusers.com/libros.asp?tipo=ebook&categoria=php**.

Crear nuestra primera base de datos

A continuación, crearemos nuestra primera base de datos, que nos servirá luego para agregar las tablas necesarias que utilizaremos en los ejercicios a realizar hacia el final de este capítulo. Si disponemos de un sitio web donde tenemos PHP y MySQL corriendo en el webserver, puede que a través de este tengamos que gestionar la creación de una nueva base de datos desde el panel de control de usuario.

Esto se debe realizar con el fin de administrar eficientemente los permisos de seguridad, los cuales no siempre coinciden entre la plataforma de administración del back-end de un sitio web y los correspondientes para la administración de las bases de datos. En otros casos, puede que la gestión para crear una nueva base de datos se realice directamente desde la interfaz PHPMyADMIN. Veremos, a continuación, las dos opciones.

Creación de una base de datos desde el panel de administración web

En el proveedor de hosting utilizado para realizar estos ejercicios, podemos encontrar que el panel de control nos ofrece la creación de una base de datos MySQL. Veamos, a continuación, un ejemplo de cómo se visualiza el panel de creación de base de datos:

MySQL no solo es administrable visualmente a través de la herramienta PHPMyADMIN. También existen otras aplicaciones instalables en sistemas operativos como Windows que permiten administrar fácilmente este motor de base de datos: entre ellas, están **dbForge Studio**, **SQLwave MySQL Client** y **MyDB Studio**.

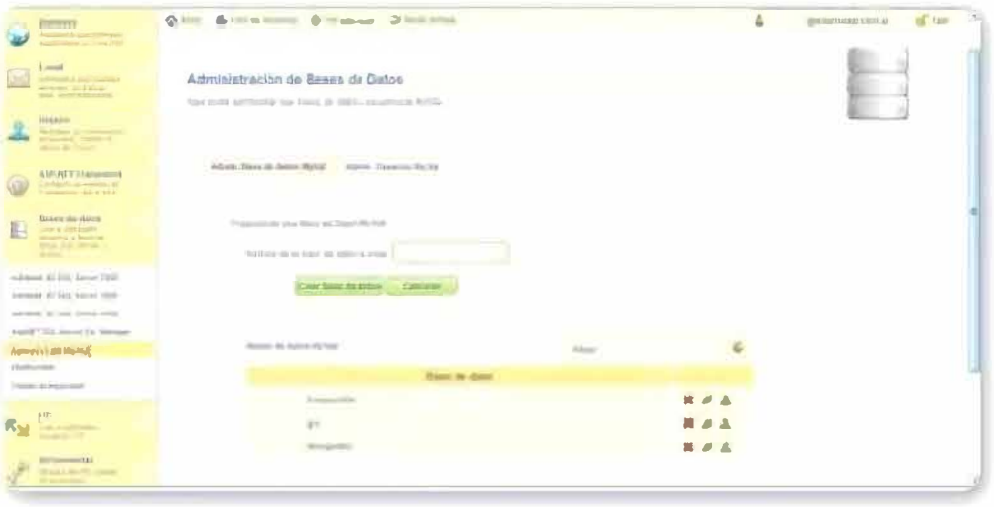


Figura 7. Panel de creación de una base de datos MySQL desde el back-end de un proveedor de hosting.

Como primer paso, debemos crear la base de datos, llamada, en nuestro caso, **usershop**. Luego crearemos un usuario, que será el administrador de la base de datos. Para poder administrarla, el paso siguiente es dirigirnos al panel **Administrar usuarios MySQL** y allí elegir un nombre de usuario administrador, una contraseña y la base de datos a la cual se aplicará.

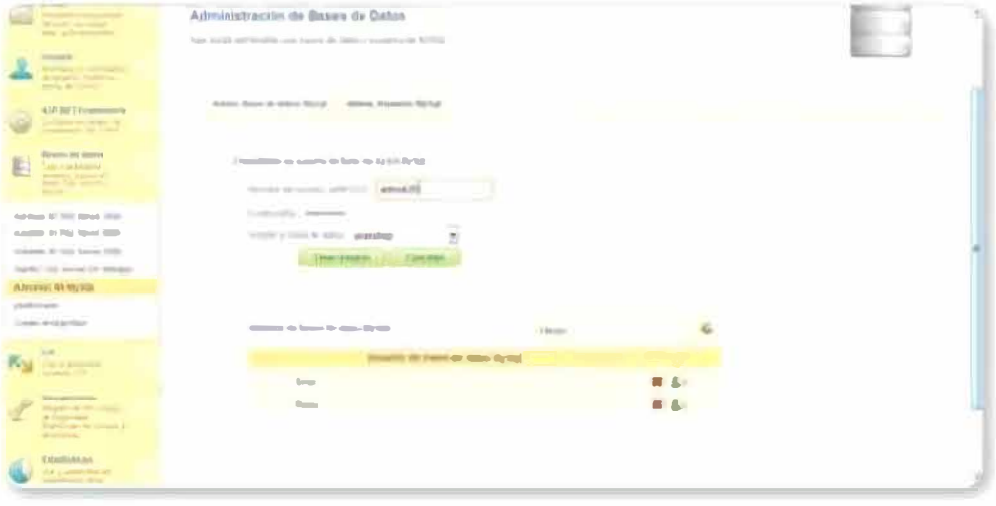


Figura 8. Desde el mismo back-end de un proveedor de hosting, podemos asignar el usuario que administrará la base de datos MySQL.

Recordemos que este paso no está disponible en todos los proveedores de hosting. Dependiendo del servicio de hosting con el que tengamos suscripción, el panel puede variar o tal vez no existir,

por lo cual deberemos realizar la creación y administración de una base de datos directamente desde la interfaz PHPMyADMIN.

Creación de una base de datos desde PHPMyADMIN

El panel PHPMyADMIN nos permite visualizar, crear y manipular todo el contenido de una base de datos MySQL. Dispone de una serie de pestañas desde donde podremos realizar todas las tareas más comunes. Habitualmente, las bases de datos creadas en MySQL suelen visualizarse en el marco izquierdo de la pantalla, que contiene un hipervínculo. Haciendo clic en él podemos acceder a visualizar el contenido de la base de datos y otras operaciones.

Veamos entonces cómo crear una base de datos desde el panel de administración PHPMyADMIN. Para acceder al panel PHPMyADMIN instalado en nuestro servidor web local, debemos ingresar en el navegador la **dirección IP** de nuestra PC o de la PC donde montamos **XAMPP**, seguido de la barra **/** y la carpeta **phpmyadmin**. Por ejemplo: **http://127.0.0.1:8086/phpmyadmin/**.

Luego, ingresamos el usuario y contraseña de administrador que seleccionamos al instalar la plataforma XAMPP o la provista por nuestro proveedor de hosting. Como resultado, veremos el panel de administración PHPMyADMIN representado en la **Figura 9**.

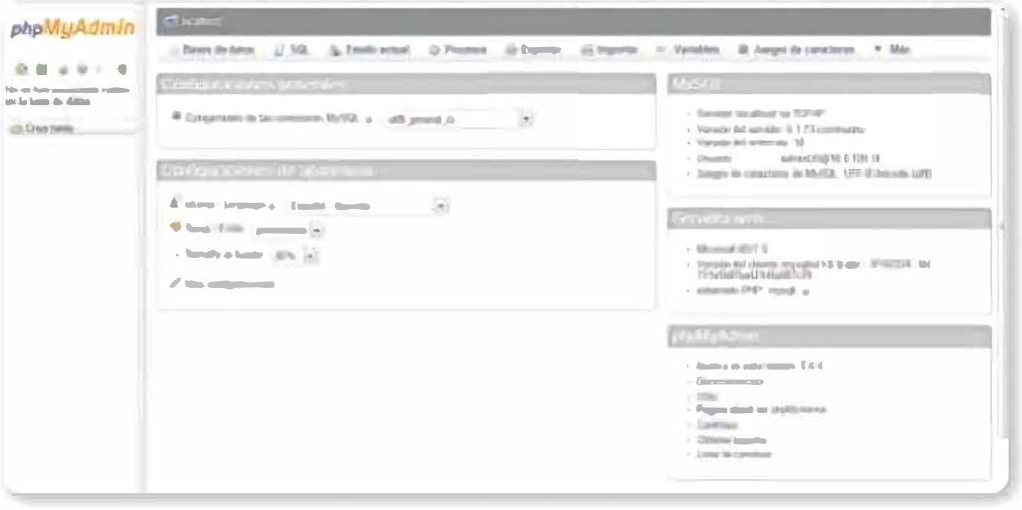


Figura 9. El panel **PHPMyADMIN** es la interfaz desarrollada en PHP que permite administrar, de forma óptima y cómoda, las bases de datos MySQL.

Si PHPMyADMIN lo permite, podemos crear la base de datos desde la interfaz gráfica. Si no, desde la pestaña SQL podemos acceder a un panel de comandos y crear desde allí la base de datos a través de la sentencia SQL:

```
create DATABASE usershop;
```

Ya creada la base de datos, debemos definir las tablas que utilizaremos en ella. Esta acción podemos realizarla desde el botón ubicado en el panel izquierdo: **Crear Tabla**. Presionamos sobre él y se abrirá una pantalla en la que debemos definir los campos y tipos de datos de la tabla.

A continuación, definimos las siguientes propiedades para la tabla a crear:

SUCURSALES

COLUMNA	TIPO	LONGITUD/ VALORES	INDICE	AUTOINCREMENTO
Id	Bigint		Primary	True
Nombredelocal	Varchar	50		
Domicilio	Varchar	100		
Localidad	Varchar	50		
Teléfono	Varchar	15		
E-mail	Varchar	50		
Ubicación	Varchar	100		
Motor de almacenamiento:	MyISAM			
Cotejamiento:	Latin1_spanish_ci			

Tabla 1. Definición técnica de la tabla **Sucursales**.

Ya creada la tabla, solo nos queda incorporar algunos registros respetando la completitud de todos los campos que incluiremos. Podemos agregar nuestros propios registros o, en su defecto, utilizar

la importación de datos que PHPMyADMIN nos provee en la sección **Importar**. Junto al material adicional de este libro, disponemos de un archivo descargable llamado **sucursales.sql.zip**, en cuyo interior se encontrará un archivo con extensión **.SQL** desde el cual podemos realizar la creación de la tabla con todos sus campos y algunos registros definidos previamente para este ejemplo.

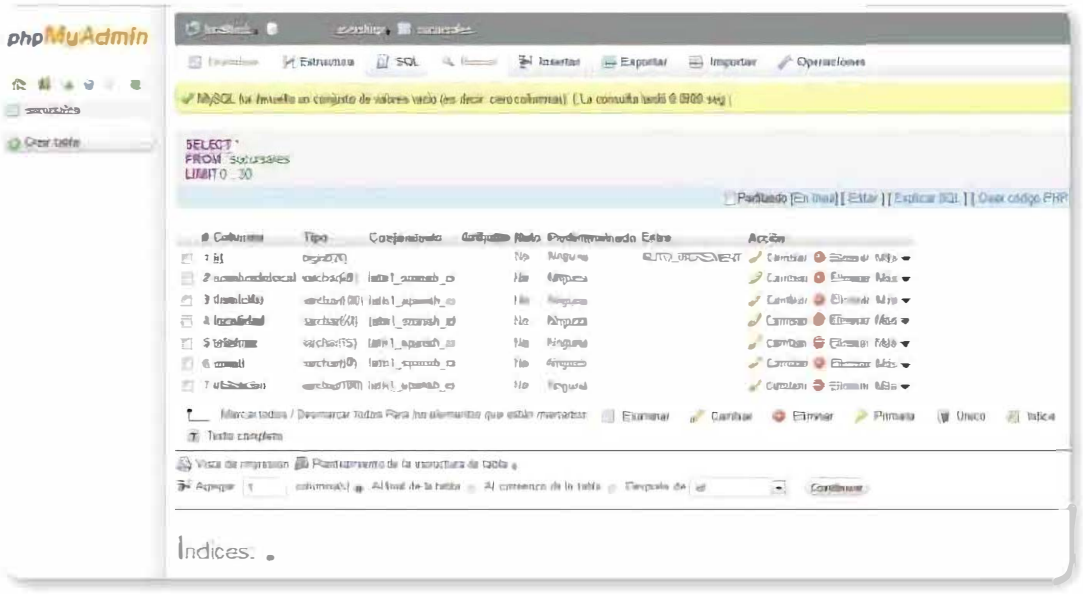


Figura 10. La vista desde PHPMyADMIN de la tabla **Sucursales** creada para los próximos ejercicios que realizaremos.

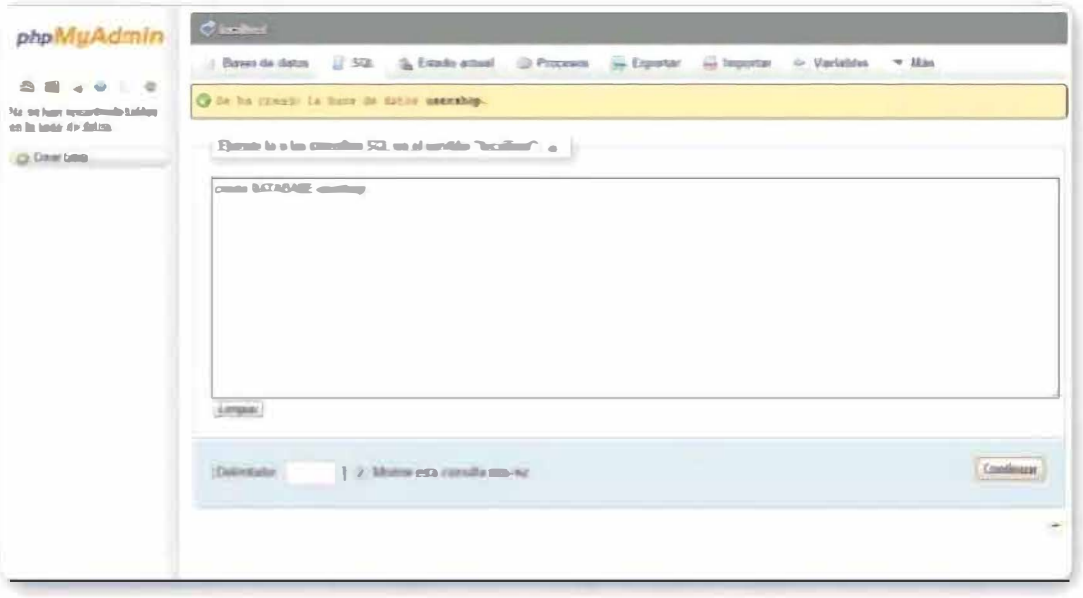


Figura 11. Desde el panel **SQL** de PHPMyADMIN también es posible ejecutar comandos de creación de base de datos y tablas, entre otros.

PHP y MySQL

PHP dispone de una función interna que nos permite conectarnos a las bases de datos, ya sea MySQL o cualquier otra de las soportadas por este lenguaje. En esta plataforma, cada vez que debemos leer el contenido de una o más tablas, primero se debe invocar la apertura de la base de datos y luego se realiza la lectura de las tablas correspondientes.

Para poder avanzar con los ejercicios correspondientes a este capítulo, desarrollaremos a continuación un archivo **.PHP** que dispondrá de las funciones que precisemos para conectarnos a la base de datos y realizar las operaciones necesarias sobre sus tablas.

Lo que haremos ahora es crear un archivo nuevo al que llamaremos **cn.php**. Notemos que, a diferencia de lo visto en capítulos anteriores, el tipo de archivo que crearemos a continuación no es un HTML, sino un archivo PHP puro, que contendrá todas las funciones y variables que necesitemos.

Ya creado el archivo, lo primero que debemos hacer es definir, en él, la estructura que nos permite escribir código PHP. Por ello, escribimos la siguiente sentencia:

```
<?
    ...
?>
```

Dentro de esta llave, propia del lenguaje PHP, definiremos todas las funciones y variables que utilizaremos. Lo primero que haremos será declarar una serie de variables que contendrán: el nombre del servidor, la base de datos, la tabla, el usuario y la contraseña que utilizaremos para acceder a ella.

```
$servidor = 'localhost';
$usuario = 'usuario';
$passwd = 'password';
$tablasucursales = 'sucursales';
$basededatos = 'usershop';
$query = '';
```

Funciones mysql_connect() y myql_select_db()

Por el momento, utilizaremos estas variables. La primera de ellas contiene el nombre del servidor donde está alojada la base de datos MySQL. Como PHP es un código que se ejecuta en el servidor, debemos definir **localhost** como valor por defecto. La segunda variable contiene el nombre de usuario; la tercera, la contraseña utilizada para MySQL; la cuarta, el nombre de la tabla a la cual queremos acceder; la quinta, el nombre de la base de datos a la que deseamos conectarnos; y la última está vacía, pero será la que contendrá la consulta SQL que recuperará los datos que deseemos visualizar.

Seguido a la creación de variables, crearemos la conexión a la base de datos y la lectura de la tabla Sucursales. Este código estará a continuación de la declaración de variables realizada. Veámoslo:

```
$conn = mysql_connect($servidor, $usuario, $passwd) or die ('Ocurrio un error al
conectarse al servidor mysql '.mysql_error());
mysql_select_db($basededatos) or die('No se pudo seleccionar la base de datos');
$query = 'SELECT DISTINCTROW(localidad) AS localidad FROM '.$tablasucur-
sales;
$result = mysql_query($query) or die('Falló la consulta: ' . mysql_error());
```

La variable **$conn** recibe el resultado de la conexión a la base de datos MySQL, a través de la función **mysql_connect()**, donde pasamos los parámetros **$servidor**, **$usuario** y **$passwd**. Seguida a esta función, controlamos si dio error o no, mediante **or die()**, donde debemos agregar el texto a visualizar si se produjo una falla en el intento de conexión.

PACKAGES O SP EN MYSQL

Los **Stored Procedures** ('procedimientos almacenados', en español), también se pueden crear en MySQL. A diferencia de otros motores de base de datos, muchos ISP tienen por defecto bloqueada esta funcionalidad. Se debe consultar con el soporte técnico del ISP si es factible o no habilitar esta funcionalidad en el panel de administración web.

Cuando establecemos conexión con el motor MySQL, nos queda definir, a través de la función **mysql_select_db($basededatos)**, la base de datos en sí con la que vamos a trabajar. Al igual que con la función anterior, controlamos si se produce un error en el intento de selección de la base de datos mediante **or die()**.

Por último, nos queda realizar la consulta SQL sobre la tabla Sucursales. Esto lo realizamos poniendo la consulta SQL dentro de la variable **$query**, la que luego pasamos como parámetro a la función **mysql_query**.

Si todo va bien, nos conectaremos al motor MySQL, seleccionaremos la base de datos y leeremos la tabla Sucursales en base a la consulta SQL realizada.

Consulta SELECT DISTINCTROW

Para quienes nunca trabajaron con sentencias SQL, destacamos que la consulta realizada a la tabla Sucursales solo recupera uno de todos los campos que contiene. El único campo que, por el momento, queremos visualizar es **Localidad**, y solo deseamos mostrar localidades que no se repitan; por ello, utilizamos la sentencia **DISTINCTROW()** que seleccionará sólo un registro de todos los que existan.

Visualización de datos

Ya creamos el código del lado del servidor que nos permite acceder a la base de datos y consultar su contenido. Como este lo realizamos sobre un archivo con extensión PHP, al ser un lenguaje de servidor, si alguien conoce este archivo y lo llama de manera independiente a través de la URL de un navegador web, no podrá visualizar el código utilizado, dado que este se ejecuta en el servidor y devuelve una página en formato HTML, que es lo que recibe el navegador web.

LIMITAR LOS REGISTROS OBTENIDOS

Cada consulta en MySQL puede limitar la cantidad de registros que se desea obtener. Esto se puede realizar con el comando **LIMIT x, y**, el cual limitará con **x** el registro inicial y con **y** el registro final a mostrar. Esta funcionalidad se utiliza al final de la sentencia SQL.

Si ejecutamos esta página en un browser y seleccionamos la opción **Ver código fuente**, solo obtendremos una página en blanco. Si esta contiene generación de HTML a través de la sentencia **echo** de PHP, solo veremos un HTML estático como resultado, pero nunca veremos variables, funciones y sentencias SQL escritas dentro de PHP.

Para finalizar la prueba realizada hasta ahora, crearemos una simple página con extensión **.PHP** que nos permita mostrar los registros obtenidos de la consulta realizada a la tabla Sucursales. La llamaremos **sucursales.php** y, dentro de ella, escribiremos el siguiente código:

```
<!DOCTYPE html>
<html>
<head>
  <title>Listar localidades</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
      <?
      // Visualizar resultados en HTML
      require 'cn.php';
      echo "<table>\n";
      while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
        echo "\t<tr>\n";
        foreach ($line as $col_value) {
          echo "\t\t<td>$col_value</td>\n";
        }
        echo "\t</tr>\n";
      }
      echo "</table>\n";
      mysql_free_result($result); //Liberar la tabla
      mysql_close($link); // Cerrar la conexión MySQL
      ?>
</body>
</html>
```

En principio, utilizamos **require** para que, junto con la página **sucursales.php**, se cargue el código desarrollado en la página **cn.php**. Seguido a esto, generamos, mediante la sentencia **echo**, una tabla como

se utilizaba en las primeras versiones de HTML para estructurar datos en pantalla. Luego nos queda, a través de la sentencia **while()**, recorrer la tabla Sucursales, para poder leer cada uno de los registros devueltos, y así visualizarlos dentro de una nueva celda de la tabla creada. La sentencia **mysql_free_result()** nos permite liberar la tabla invocada, y la sentencia **mysql_close()** cierra la conexión con el servidor MySQL.

Notemos que uno de los parámetros utilizados en la función **mysql_fetch_array()** es **$result**, la variable que recibe el resultado de la consulta SQL en la página **cn.php**. Al incluir esta última página dentro de **sucursales.php**, estamos heredando de ella las funciones y variables creadas y seteadas.

Figura 12. El resultado de la visualización de las localidades representado en una página PHP.

Integrar PHP con jQuery Mobile

Ya con los conceptos básicos de PHP y MySQL, aprovecharemos la tabla Sucursales que hemos creado para realizar un ejercicio que nos enseñe a complementar PHP con jQuery Mobile. Esto nos permitirá desarrollar aplicaciones web móviles dinámicas, definiendo una estructura que se ocupe del diseño de interfaz de usuario,

y que esta estructura a su vez consuma datos de una base de datos sin necesidad de estar rediseñando la aplicación web cada vez que necesitemos agregar o quitar información.

Proyecto: librerías

A continuación, desarrollaremos un nuevo ejercicio que consistirá en representar la web móvil de una cadena de librerías. La aplicación permitirá visualizar un listado de sucursales agrupadas por localidad, donde, al seleccionar una localidad, podremos entrar a ver todas las sucursales que existen en ella. Luego, podremos seleccionar una sucursal y ver en detalle la información, junto con un mapa de su ubicación física, el cual se creará de forma dinámica utilizando las coordenadas (latitud y longitud) del local.

Para poder llevar a cabo este ejercicio, debemos haber realizado el ejercicio anterior, ya que utilizaremos MySQL como base de datos de las sucursales y PHP para acceder a esta información dinámica y visualizarla en pantalla.

Figura 13. Nuestro proyecto, bautizado **User Book Store**, nos permitirá explorar el poder en conjunto de PHP, MySQL y jQuery Mobile.

Preparar el entorno

Para poder llevar a cabo este ejercicio, descargaremos de la sección **Redusers Premium** el material correspondiente a este capítulo, ubicado dentro del archivo **Cap06.userbookstore.rar**. Aquí encontraremos una carpeta que contiene la estructura básica del sitio, la página principal (**home.php**), una carpeta (**images**) con las imágenes que utilizaremos en el ejercicio y los archivos básicos de PHP para conectarnos a MySQL y poder acceder al contenido de la tabla Sucursales.

El código a modificar del archivo **cn.php** es el siguiente:

```
$servidor = 'MISERVIDORMYSQL';
$usuario = 'MINOMBREDEUSUARIO';
$passwd = 'MIPASSWORD';
$tablasucursales = 'sucursales';
$basededatos = 'BASEDEDATOSCREADA';
$query = ''; //variable a utilizar
$querylibrerias = ''; //variable a utilizar
$filtrolocalidad = ''; //variable a utilizar
```

En el archivo **cn.php** debemos modificar las variables correspondientes que nos permiten acceder a la base de datos MySQL de nuestro servidor web personal o el instalado mediante la solución XAMPP mencionada en el inicio de este capítulo.

Con esto ya ajustado, nos queda crear la página **sucursales.php**, donde listaremos las sucursales de esta cadena de librerías, visualizando un ListView con las localidades donde hay uno o más locales. Para ello, agregamos el siguiente código base en el archivo **sucursales.php**:

```
<!DOCTYPE html>
<html>
<head>
   <title>USER Book Store</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
```

```
    <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
    <?
      //Llamar los archivos Require
      ?>
  </script>
  </head>
  <body>
    <div data-role="page" id="index" data-theme="d">
      <div data-role="header" data-theme="c" data-position="fixed">
          <div id="imagenLogo" align="center">
            <img src="images/Logo250x42.png" style="max-width:250px;min-
width:250px" alt="USER Book Store" longdesc="USER Book Store">
          </div>
      </div>
      <div data-role="content" data-theme="d">
        <p align="center"><strong>Sucursales USER BOOK STORE</strong></
p>
            <br>
        <p>
                  <?
                      //Código PHP
                  ?>
        </p>
      </div>
      <div data-role="footer" data-position="fixed" data-theme="c" data-
position="fixed">
                  <div data-role="navbar">
          <ul>
            <li><a href="home.php" data-icon="home">Inicio</a></li>
            <li><a href="#" data-icon="search">Sucursales</a></li>
            <li><a href="#" data-icon="plus">Profesionales</a></li>
          </ul>
                  </div>
      </div>
    </div>
  </body>
  </html>
```

Esto nos permite tener el código base para la página que listará las localidades. Dentro de la sección **content** de jQuery Mobile, reservamos el espacio correspondiente para escribir el código PHP que creará, de manera dinámica, el ListView que nos mostrará todas las localidades existentes en la tabla, tomándolas del campo **localidad** de la tabla Sucursales.

Antes de escribir el código correspondiente, crearemos un nuevo archivo llamado **selectsuc.php**, donde incluiremos la consulta SQL correspondiente que nos permitirá filtrar las localidades de forma individual cargadas en cada registro de la tabla Sucursales.

El código de este archivo es el siguiente:

```
<?
        $querylibrerias = 'SELECT DISTINCTROW (localidad) AS localidad
FROM sucursales LIMIT 0 , 30';
        $result = mysql_query($querylibrerias) or die('Falló la consulta: ' . mysql_
error());
?>
```

A través de la variable **$querylibrerias**, generamos una consulta SQL, en la cual utilizamos la cláusula **DISTINCTROW**, que nos permitirá obtener un único registro entre **N** repetidos. De esta manera, podemos conocer el listado de cada localidad única. Luego, a través de la variable **$result**, ejecutamos la consulta SQL en la base de datos para obtener el conjunto de registros correspondiente.

Ahora volvemos al código del archivo **sucursales.php**. En el apartado **<Header>**, justo debajo de todas las declaraciones **<script>**, encontramos una pequeña sentencia PHP, a través de la cual incluimos el archivo **cn.php**, que contiene el conjunto de variables con la información de usuario, password y base de datos, y la conexión a MySQL.

Debajo de este archivo, incluimos la llamada a **selectsuc.php**, en el que, al cargarse junto con la página invocada, ejecutamos la consulta en sí. Debemos tener en cuenta que el llamado al archivo **cn.php** ya se hizo, por lo que ya tenemos conexión a la base de datos MySQL.

```
require 'cn.php';
require 'selectsuc.php';
```

Ahora solo nos resta escribir dentro del cuerpo principal de la página el código PHP que nos listará, a través de un widget ListView, el conjunto de localidades que tienen una o más librerías. Ubicamos el código PHP iniciado cuando creamos la página y eliminamos la línea correspondiente al comentario. En su lugar, agregamos la siguiente sentencia:

```
<?
echo "<ul data-role='listview' data-inset='true'>";
while($row = mysql_fetch_assoc($result)){
        $filtrolocalidad = $row['localidad'];
        $filtrolibrerias = "librerias.php?l=".$filtrolocalidad;
        echo "<li><a href='".$filtrolibrerias."'>".$row['localidad']."</a></li>";
        }
echo "</ul>";
?>
```

LA SENTENCIA ECHO SE UTILIZA PARA CREAR DE FORMA DINÁMICA UN WIDGET LISTVIEW

Este bloque de código PHP utiliza la sentencia **echo** para crear de forma dinámica un widget ListView. La instrucción **while** permite recorrer el set de registros que cargamos leyendo la tabla Sucursales en la variable **$result**. Recorre desde el primero hasta el último y se asigna a la variable **$filtrolocalidad** el valor del campo **localidad** del registro que se está leyendo.

Luego asignamos, en la variable **$filtrolibrerias**, una URL parametrizada que nos permitirá, al presionar sobre ella, listar todas las librerías existentes en esa localidad. Por último escribimos, mediante la sentencia **echo**, el ListItem correspondiente cuyo nombre corresponde a la localidad que estamos recorriendo mediante la sentencia **while**.

MENÚ INFERIOR O LATERAL

Tal como vimos en los capítulos anteriores, podemos optar por la creación de un menú inferior, utilizando el comando **NavBar** de jQuery Mobile, o de un menú lateral, utilizando el comando **Panel** de este framework. Este último nos será útil para aquellos proyectos que requieran más espacio en pantalla.

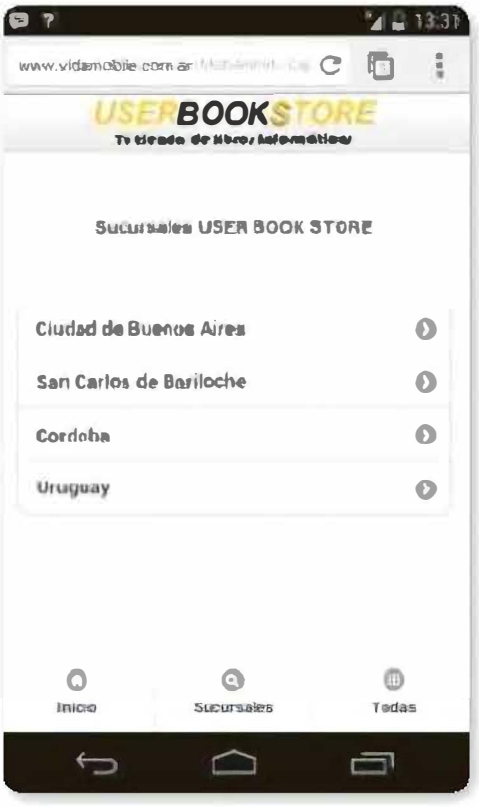


Figura 14. La página **sucursales.php** ya visualiza las localidades que tienen al menos una sucursal de la cadena de librerías.

Librerías por localidad

A continuación, creamos una nueva página PHP que nos permitirá obtener el listado de librerías de la localidad seleccionada con el ListView creado en la página **sucursales.php**. La estructura básica del nuevo archivo, al que llamaremos **librerias.php**, es el siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <title>USER Books Store</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
  <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
  <?
    //obtener librerías según la localidad seleccionada
```

```
?>
</script>
</head>
<body>
  <div data-role="page" id="index" data-theme="d">
    <div data-role="header" data-theme="c" data-position="fixed">
        <div id="imagenLogo" align="center">
          <img src="images/Logo250x42.png" style="max-width:250px;min-
width:250px" alt="USER Book Store" longdesc="USER Book Store">
        </div>
    </div>
    <div data-role="content" data-theme="d">
      <p align="center"><strong>Localidad: <? //filtro aplicado ?></strong></p>
          <br>
      <p>
            <?
                  //Listar las sucursales obtenidas
        ?>
      </p>
    </div>
    <div data-role="footer" data-position="fixed" data-theme="c" data-
position="fixed">
                  <div data-role="navbar">
        <ul>
          <li><a href="home.php" data-icon="home">Inicio</a></li>
          <li><a href="#" data-icon="search">Sucursales</a></li>
          <li><a href="#" data-icon="plus">Profesionales</a></li>
        </ul>
                  </div>
    </div>
  </div>
</body>
</html>
```

Esta página tiene la particularidad de estructurar el llamado a archivos **.PHP** que utilizamos con la sentencia **require**, de una manera distinta a la que vimos en la página **sucursales.php**. En el **<Header>** de esta página abrimos una llave con código PHP y escribimos lo siguiente:

```
require 'cn.php';
$filtrolocalidad = $_GET["l"];
if (trim($filtrolocalidad) == "") {
        $filtrolocalidad = "Todas";
        $sqlquery = "SELECT nombredelocal, domicilio FROM sucursales
LIMIT 0 , 100";
    }
    else {
        $sqlquery = "SELECT nombredelocal, domicilio FROM sucursales
WHERE localidad = '".$filtrolocalidad."' LIMIT 0 , 100";
    }
    require 'selectlocalidad.php';
```

En el **<Header>** incluimos, en la primera línea, el llamado al archivo **cn.php**, el cual obtiene las variables necesarias y se conecta a la base de datos MySQL. Luego, al llamar la página **librerias.php** desde la página **sucursales.php**, en la URL pasamos como parámetro una variable denominada **$l**, a la que le indicamos qué localidad debe filtrar.

Esta variable es la que hereda el filtro que debemos utilizar en la consulta a la base de datos. A continuación, asignamos en la variable **$filtrolocalidad** el valor obtenido a través del paso de parámetros de la URL. Para ello, utilizamos la sentencia **$_GET[]**, poniendo entre corchetes y comillas dobles el nombre de la variable **$l** pasada como parámetro, pero obviando el signo **$**, que no es necesario. Seguido a esto, a través de la sentencia **if**, consultamos si en **$filtrolocalidad** se guardó o no un valor. Si no se guardó, le asignamos a esta variable el valor **'Todas'**, y armamos una consulta en **$sqlquery** que devuelva todas las librerías. Si la variable **$filtrolocalidad** tiene un valor asignado, armamos la consulta SQL indicando como parámetro el valor de esta variable. Por último, llamamos a **require 'selectlocalidad.php'** para ejecutar el valor de la variable **$sqlquery**.

> SEGÚN EL RESULTADO DEL IF, SE CONSULTARÁ POR TODOS O PARTE DE LOS REGISTROS

Entonces, dependiendo del resultado obtenido a través de la condición **if**, se creará una consulta SQL específica para que devuelva la información filtrada o el total de registros existentes.

Informar las sucursales que se listan

En base al control que hicimos a través de la condición **if**, aprovechamos que tenemos la información de la localidad a visualizar y la mostramos como título en el cuerpo de la página. Dentro de las llaves de código PHP que preceden al título **Localidad**, agregamos el siguiente código:

```
echo $filtrolocalidad;
```

Esto permitirá conocer si filtramos una localidad específica o si visualizamos todas las localidades.

Por último, nos queda agregar el código correspondiente para crear el ListView dinámico y que este nos muestre en detalle el nombre del local listado y el domicilio. Esta información la visualizará en el mismo ListItem, el cual duplica el contenido que visualiza cada ítem, formateando la información con un título y un párrafo.

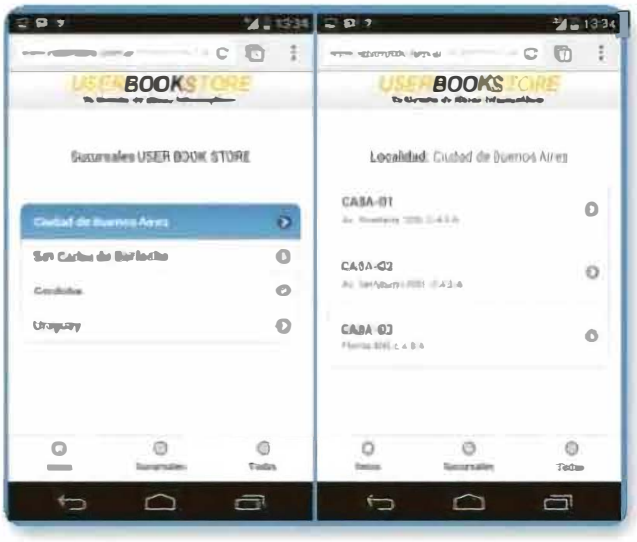


Figura 15. Al seleccionar una localidad, el sistema cargará todas las librerías, obteniéndolas a través de la consulta SQL.

Veamos a continuación el código:

```
echo "<ul data-role='listview' data-inset='true'>";
while($row = mysql_fetch_assoc($resultado)){
        echo "<li><a href='detalle.php?d=".$row['nombredelocal']."'>";
```

```
        echo "<h2>".$row['nombredelocal']."</h2>";
        echo "<p><p>";
        echo "<p>".$row['domicilio']."</p>";
        echo "</a></li>";
        }
echo "</ul>";
```

Nuevamente creamos, a través de la sentencia **echo**, la estructura básica de un ListView, combinándola con la sentencia **while()**, que nos permite recorrer el set de registros obtenidos. Creamos un hipervínculo para cada ítem, lo que nos permitirá ir al detalle de la información de la librería. Seguido a esto, visualizamos, mediante el formato **<h2>**, el nombre del local, y el domicilio, mediante el formato **<p>**.

Visualizar la información completa

En el tramo final del ejercicio, nos queda desplegar una última página que nos permita acceder al detalle de una librería. En esta página podremos ver el nombre del local, su domicilio, un mapa con la ubicación física, el teléfono y su e-mail. En estos últimos dos campos crearemos los hipervínculos necesarios para iniciar una comunicación telefónica desde la WebApp o enviar un correo de consulta.

Creación de mapa dinámico utilizando Bing Maps

Para poder visualizar un mapa estático que se cree de forma dinámica enviándole las coordenadas correspondientes, en este ejercicio utilizaremos los servicios de **Bing Maps**. Si tenemos cuenta de correo electrónico de **Hotmail.com** u **Outlook.com**, podremos utilizarla para registrarnos en el servicio de **Bing Maps Portal** y poder obtener el **token** correspondiente para utilizar en nuestro sitio web.

Este token es una clave alfanumérica que se debe pasar como parámetro en la URL que genera la imagen del mapa estático de Bing. Es un requisito importante para poder desarrollar esta parte del ejercicio. Podemos, como alternativa, crear un archivo de imagen que capture las coordenadas del domicilio y asociarlo con el correspondiente registro, pero esta tarea deberíamos hacerla a futuro por cada nuevo registro que se agregue en la base de datos.

Por ello, es más útil crear el correspondiente token o **API Key** y solo ocuparnos de tener las coordenadas almacenadas en un registro para luego pasarlas como parámetro y que la imagen del mapa se genere automáticamente.

Para crear la clave, nos dirigimos a **www.bingmapsportal.com** y nos logueamos como usuarios. Una vez realizado esto, dentro de la pantalla principal de este recurso encontraremos el menú **My Accounts** y, por debajo de este, la opción **Create or view keys**. Ingresamos a esta opción y cargamos la información que nos solicita.

En el apartado **Application Type** podemos especificar el tipo de objetivo que tendrá el consumo de mapas Bing. Si es para una aplicación comercial, seguramente deberemos pagar un canon mensual. Podemos utilizar, a modo de prueba, las opciones **Public Website, Education** o **Private Website**.

Por último obtendremos la clave, que será una cadena alfanumérica similar a la siguiente:

AuO6VfPiYbbaDSf5987aSFdsfdsfHjyyhmapsdariWSF557mssdRy77er.

Figura 16. A partir de **www.bingmapsportal.com** podemos crear las claves necesarias para utilizar los mapas de Bing desde nuestra aplicación.

Crear el detalle de la sucursal

Una vez que la gestión de la API Key de Bing Maps fue realizada, nos queda crear una nueva página a la que llamaremos **detalle.php**,

que corresponde a la que estamos invocando desde los hipervínculos dinámicos creados en la página **sucursales.php**.

La estructura de esta página es la siguiente:

```
<!DOCTYPE html>
<html>
<head>
   <title>USER Books Store</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
   <?
            //Obtener sucursal a visualizar
   ?>
</head>
<body>
   <div data-role="page" id="index" data-theme="d">
      <div data-role="header" data-theme="c" data-position="fixed">
         <div id="imagenLogo" align="center">
            <img src="images/Logo250x42.png" style="max-width:250px;min-
width:250px" alt="USER Book Store" longdesc="USER Book Store">
         </div>
      </div>
      <div data-role="content" data-theme="d">
         <?
               //Mostrar el detalle de la sucursal
         ?>
      </div>
      <div data-role="footer" data-position="fixed" data-theme="c" data-
position="fixed">
               <div data-role="navbar">
         <ul>
            <li><a href="#" data-rel="back" data-icon="back">Volver</a></li>
            <li><a href="sucursales.php" data-icon="search">Sucursales</
```

```
a></li>
                <li><a href="librerias.php?$l=" data-icon="grid">Todas</a></li>
            </ul>
                    </div>
        </div>
    </div>
</body>
</html>
```

El objetivo de esta página será listar el detalle de la librería. Visualizaremos el nombre, el mapa de su ubicación, el domicilio, la localidad y, por último, dos widgets **button**, donde podremos invocar el llamado telefónico a la sucursal o generar una nueva consulta por e-mail a su dirección de correo. Como hicimos en las páginas anteriores, creamos un código en el apartado **<header>** de esta página, donde agregamos la siguiente sentencia PHP:

```
require 'cn.php';
$filtrodetalle = $_GET["d"];
if (trim($filtrodetalle) == '') {
    header('Location: sucursales.php');
    }
else {
    $sqlquery = "SELECT * FROM sucursales WHERE nombredelocal = '".$fil-
trodetalle."' LIMIT 0 , 1";
    }
require 'selectdetalle.php';
```

Invocamos inicialmente al archivo **cn.php** para conectarnos a MySQL y a la base de datos correspondiente. Luego obtenemos, en la variable **$filtrodetalle**, el valor del parámetro pasado por URL. Seguidamente, mediante la condición **if**, controlamos que la variable tenga un valor asignado. Si no lo tiene, entonces redireccionamos al usuario a la página **sucursales.php**, para que seleccione una sucursal válida. De esta manera evitamos que el usuario cargue directamente la página **detalle.php** sin un valor válido como parámetro.

Si la variable **$filtrodetalle** tiene un valor, entonces obtenemos, mediante la consulta SQL, el nombre del local que deseamos

visualizar. Por último, invocamos el archivo **selectdetalle.php**, que crearemos a continuación:

```
<?
        $resultado = mysql_query($sqlquery) or die('Falló la consulta: ' . mysql_er-
ror());
?>
```

Este archivo, al igual que los anteriores, sólo se ocupa de invocar la consulta SQL cargado en la variable **$sqlquery**, y obtener el registro especificado como parámetro. Volvemos al archivo **detalle.php**. En el cuerpo del mismo agregamos el siguiente código:

```
while($row = mysql_fetch_assoc($resultado)){
        echo "<p align='center'><strong>".$row['nombredelocal']."</strong></p>";
        echo "<div id='verMapa' align='center'>";
        $coordenada = $row['ubicacion'];
        $ancho = 350;
        $alto = 150;
        $mapa = "http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/".$coo
rdenada."/16?mapSize=".$ancho.",".$alto."&pp=".$coordenada.";64;U&key=A
PI_KEY_DE_BINGMAPS";
        echo "<img border='3' style='border-color:grey' src='".$mapa."'
title='".$coordenada."'>";
        echo "</div>";
        echo "<p><strong>Domicilio:</strong> ".$row['domicilio']."</p>";
        //echo "<br>";
        echo "<p><strong>Localidad:</strong> ".$row['localidad']."</p>";
        $link = "tel:".$row['telefono'];
        echo "<a href='".$link."' data-role='button' data-corners='false'>Llamar:
".$row['telefono']."</a>";
        //echo "<br>";
        $link="mailto:".$row['email']."?subject=Consulta%20desde%20la%20
web";
        echo "<a href='".$link."' data-role='button' data-corners='false'>Enviar
consulta x email</a>";
        }
```

Al igual que en la carga de un conjunto de registros, se debe iniciar un bucle **while** para poder cargar el único registro obtenido. Luego creamos un párrafo con alineación al centro de la página, donde visualizamos el nombre del local. Creamos un **div**, centrado también, donde ubicaremos la imagen del mapa dinámico que invocaremos desde Bing Maps.

Luego, en las tres variables siguientes, **$coordenada**, **$ancho** y **$alto**, ajustamos los valores que permitirán identificar y formatear el mapa que necesitamos visualizar. En la variable **$coordenada**, configuramos el dato almacenado en el campo **ubicación**, el cual contiene las coordenadas que debemos identificar en el mapa. Luego, en las variables **$ancho** y **$alto**, seteamos los valores **width** y **height**, respectivamente, que le daremos al mapa en pantalla.

Por defecto establecemos un valor pequeño, pensado para un smartphone, pero este puede volverse dinámico si combinamos PHP con JavaScript para obtener el ancho de la pantalla del equipo. Igualmente, debemos verificar la documentación de Bing Maps, ya que, en estos casos, los mapas estáticos poseen un ancho máximo definido que no puede superar determinada cantidad de pixeles. Esta documentación es cambiante, por lo tanto, debemos consultarla periódicamente dependiendo del tipo de proyecto que abordemos con mapas estáticos o dinámicos.

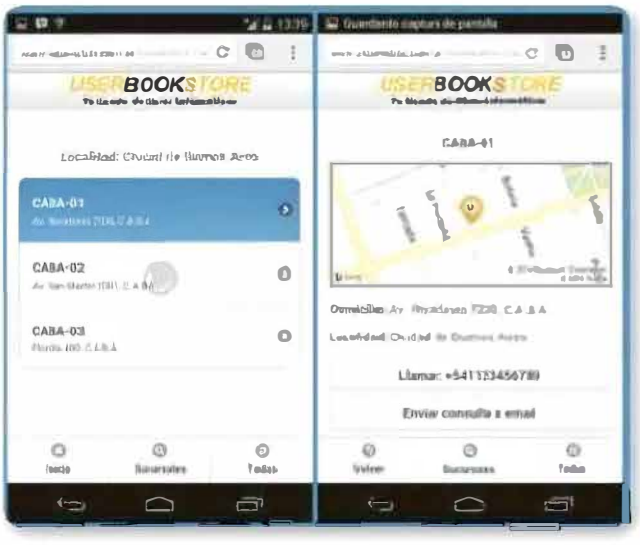


Figura 17. Al seleccionar una librería, se listará el detalle de esta, visualizando su ubicación desde un mapa estático generado con Bing Maps.

Siguiendo en la conformación del detalle de los datos de la librería consultada, en la variable **$mapa** seteamos la URL correspondiente a

Bing Maps, combinando en la URL los valores obtenidos en la variable anterior. Esta URL está conformada por una serie de parámetros personalizables que podremos consultar en el portal de Bing Maps, para agregar o modificar los datos por defecto aquí generados.

Seguido a esto, creamos una imagen con un borde de **3 pixeles** en color gris y establecemos como destino el valor contenido en la variable **$mapa**. Luego nos queda mostrar el dato del domicilio y localidad de la librería. Por último, creamos dos hipervínculos del tipo **button**, donde establecemos: en el primero, el discado del número telefónico del local, y en el segundo, un hipervínculo que inicie un nuevo mensaje de correo electrónico, donde podemos volcar la consulta a realizar.

Nuestro ejercicio ya está completo. Solo nos resta navegar por él y probar a fondo las funcionalidades implementadas.

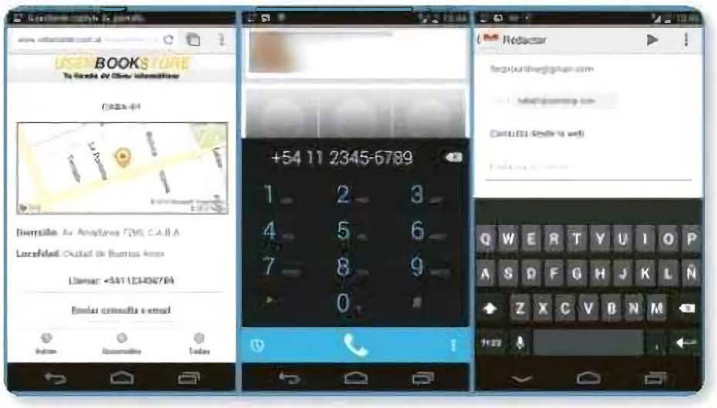


Figura 18. Los botones generados en el detalle también poseen la funcionalidad de iniciar un llamado o enviar un correo electrónico a la sucursal visualizada.

RESUMEN

A lo largo de este capítulo, realizamos una rápida introducción al lenguaje PHP y al sistema de base de datos MySQL. Conocimos herramientas como XAMPP, que nos permite contar con un sistema web montado en nuestra computadora personal, repasamos la herramienta PHPMyADMIN, con el fin de administrar MySQL de forma visual, y realizamos un ejercicio completo que nos permitió combinar HTML5, JavaScript, jQuery Mobile, PHP y MySQL para generar un portal dinámico que aproveche las características de una base de datos, el sistema de comunicaciones de los teléfonos móviles y los mapas estáticos que Bing Maps pone a nuestra disposición.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué significa **PHP**?
2. ¿Cuál es la forma de iniciar un bloque de código en este lenguaje?
3. ¿Cuál es la estructura de una función PHP?
4. ¿Las funciones PHP pueden recibir parámetros?
5. ¿Las funciones PHP pueden devolver un resultado?
6. ¿Cómo se puede administrar visualmente una base de datos **MySQL**?

EJERCICIOS PRÁCTICOS

1. Agregue al menos un registro más con una nueva localidad en MySQL y su correspondiente coordenada (utilice **Bing Maps** o **Google Maps** para conseguir las coordenadas correctas de la dirección agregada).
2. Liste la sección **Sucursales** para validar que la nueva localidad se visualice.
3. Desde la aplicación, ingrese a la localidad y al detalle de esta para visualizar su correcto funcionamiento.
4. Agregue un campo más a la tabla MySQL que sea del tipo booleano y modifique la aplicación para que solo liste localidades y librerías que tengan el valor **TRUE** en este nuevo campo.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*07

Almacenamiento local y aplicaciones offline

En este capítulo repasaremos las diferentes propuestas de HTML5 para almacenar sitios y datos en los dispositivos del usuario. A través del almacenamiento local, podremos guardar desde información simple hasta una base de datos en el navegador web móvil. También conoceremos las aplicaciones offline que permiten descargar una WebApp completa y utilizarla sin necesidad de estar conectados a internet.

Almacenamiento local

El conglomerado de características que HTML5 trajo al mundo web y a la Web móvil incluye, también, cambios importantes en cuanto a almacenamiento local del lado del cliente. Desde los inicios de la Web –tal como la conocemos hoy–, todo tipo de almacenamiento de información en las computadoras del cliente se llevó a cabo utilizando el método de cookies. Este impartía la creación de una especie de archivo que oficiaba de variable en el equipo cliente, en el que se almacenaba información importante que serviría para reutilizar en el sitio web durante todo el tiempo de navegación de un usuario.

La información de datos almacenada en cookies podía estar o no atada a una fecha de vencimiento específica. Si, por ejemplo, le preguntamos al usuario de una página web su nombre cuando ingresa por primera vez, el valor ingresado por el usuario puede almacenarse en una cookie y reutilizarse todas las veces que este vuelva a ingresar a al sitio web, o bien eliminarse luego de un período de tiempo determinado.

Sin embargo, con la evolución del lenguaje HTML, desde el modelo impuesto por HTML5, se planteó una mejora en cuanto a almacenamiento de la información en los equipos del usuario, que brinda una alternativa más completa a las clásicas cookies, desde la característica de almacenamiento local (o, en inglés, *local storage*).

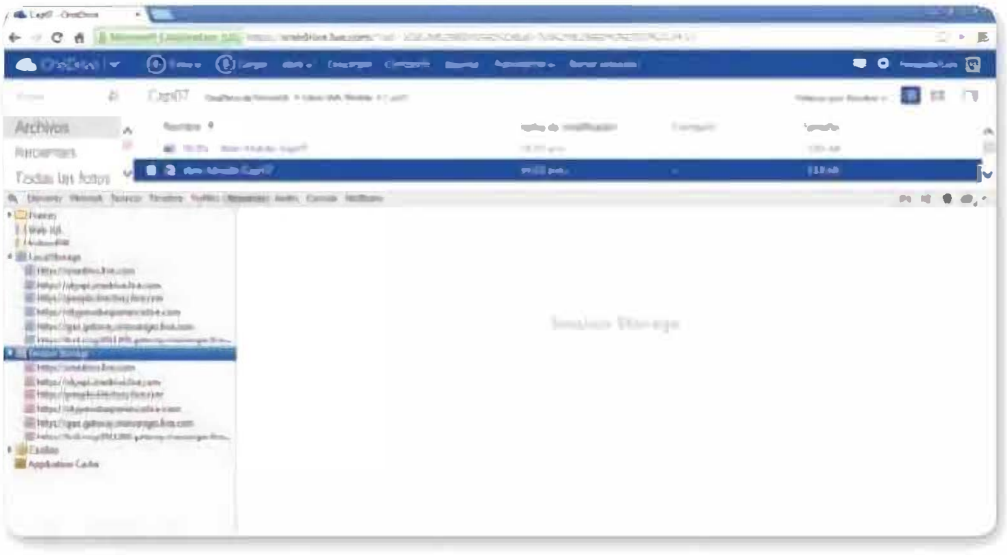


Figura 1. Desde los navegadores de escritorio, utilizando las herramientas para el desarrollador, podemos acceder a visualizar las diferentes opciones de almacenamiento que nos brinda HTML5.

Local storage

El almacenamiento local, conocido como **local storage** en HTML5, nos permite guardar una cantidad mucho mayor de datos en el equipo del cliente que lo que habitualmente permite guardar una cookie. Local storage nos ofrece una capacidad de **5 megabytes** de almacenamiento local, contra los **4 kilobytes** máximos que permite guardar cada cookie.

Otra de las ventajas destacadas del almacenamiento local es que cada sitio web que utiliza cookies en el equipo cliente envía los datos de esta por cada petición cliente/servidor que se realiza entre el browser y la web visitada. Esto sucede comúnmente cuando se crea una cookie de sesión de usuario, que debe viajar durante cada visita o actualización de información en una página web.

> LOCAL STORAGE NOS PERMITE GUARDAR MÁS DATOS QUE UNA COOKIE EN EL EQUIPO DEL CLIENTE

Si bien las cookies no dejan de ser útiles para muchas implementaciones de comunicación y manejo de datos, este envío constante de información entre el equipo cliente y el servidor genera ralentización de la comunicación entre los extremos y hasta puede presentar problemas de seguridad informática si la información almacenada es sensible y no está cifrada.

Así es como HTML5, en la creación de la especificación de almacenamiento de datos locales, decidió brindar una alternativa a las clásicas cookies, presentando la solución a través de la implementación de local storage y **session storage**, para almacenar datos locales varios o información segura sobre la sesión de un usuario mediante **login**.

WEB STORAGE EN HTML5

Tanto local storage como session storage son implementaciones que nacen de la API **Web Storage** detallada en la especificación de HTML5. La creación de estos elementos se realizó al concebirlos como atributos del objeto **Window**, presente en **JavaScript**, para el manejo de diferentes funcionalidades dentro de un sitio web.

Comportamiento de local storage

A través de local storage podemos definir una serie de atributos y métodos con los cuales realizaremos, luego, diferentes acciones de almacenamiento y recuperación de datos.

COMPONENTES DE LA METODOLOGÍA DE ALMACENAMIENTO

CLAVE	TIPO	DESCRIPCIÓN
GetItem(key)	**Método**	Devuelve una cadena con el valor del elemento clave (key).
SetItem(key, value)	**Método**	Almacena un valor (value), referenciado por una clave (key).
RemoveItem(key)	**Método**	Elimina la clave y el valor especificados.
Length	**Atributo**	Contiene el número de elementos almacenados como par (clave/valor).
Key(index)	**Atributo**	Devuelve una cadena con la clave del elemento que ocupa la posición indicada en (index), dentro de la colección de datos almacenados en (key).
Clear()	**Método**	Elimina todos los elementos previamente creados.

Tabla 1. Descripción de los elementos que componen la metodología de almacenamiento propuesta en HTML5.

LOS DATOS GUARDADOS MEDIANTE LOCAL STORAGE SON ALMACENADOS DE MANERA INDEFINIDA

Todos los datos que almacenamos mediante el uso de local storage son guardados de manera indefinida, al menos hasta que ejecutamos el método **clear()**, que se ocupa de eliminar todo elemento creado.

Comportamiento de session storage

Session storage se comporta de la misma manera que local storage, almacenando datos mediante la especificación de sus diferentes atributos y métodos. La diferencia radica en que, cuando finalizamos la navegación en el sitio web y cerramos el navegador o la ventana en cuestión, toda esta información es eliminada del equipo.

Comprobación de compatibilidad de un navegador

Si bien nuestro enfoque está orientado por completo a la tecnología mobile, siempre debemos asegurarnos, a la hora de implementar este tipo de solución en una web móvil, que el navegador web del cliente tenga compatibilidad con la propuesta de Web Storage. Esto se puede realizar fácilmente desde JavaScript, y una buena práctica es implementarlo en el inicio de una WebApp. Así, les informamos a los usuarios que visitan el sitio si el navegador de su dispositivo móvil permitirá ejecutar o no ciertas acciones de la WebApp. Esta premisa nos permite a nosotros, como programadores, tener en cuenta si permitimos que el usuario utilice o no nuestro desarrollo, evitando asi tener errores no controlados desde la perspectiva de JavaScript.

Para poder comprobar si cualquier navegador web que accede a nuestra WebApp dispone de compatibilidad con la funcionalidad de Web Storage, debemos escribir la siguiente sentencia JavaScript:

```
<script type="text/javascript">
 function comprobarSoporteWS() {
   if (window.sessionStorage && window.localStorage) {
       alert('Su dispositivo permite utilizar almacenamiento local.');
   } else {
      alert('Su dispositivo no soporta el uso de Web Storage.');
       //Aquí podemos aplicar redirect hacia otra página.
     }
 }
</script>
```

COMPROBAR LA COMPATIBILIDAD

A través del sitio web **www.w3.org**, podemos acceder en detalle a todas las características técnicas del lenguaje HTML en todas sus variantes. Allí podemos conocer, además, las especificaciones actualmente vigentes, como también aquellas que ya no son soportadas por lo navegadores modernos.

En este bloque de código encontramos que se utiliza el objeto **window** para determinar si el navegador web soporta local storage o session storage. Estas dos características nacieron juntas dentro de la especificación Web Storage de HTML5. Aun así, siempre es bueno realizar esta validación, para asegurarnos por completo de que sean soportadas por el navegador web.

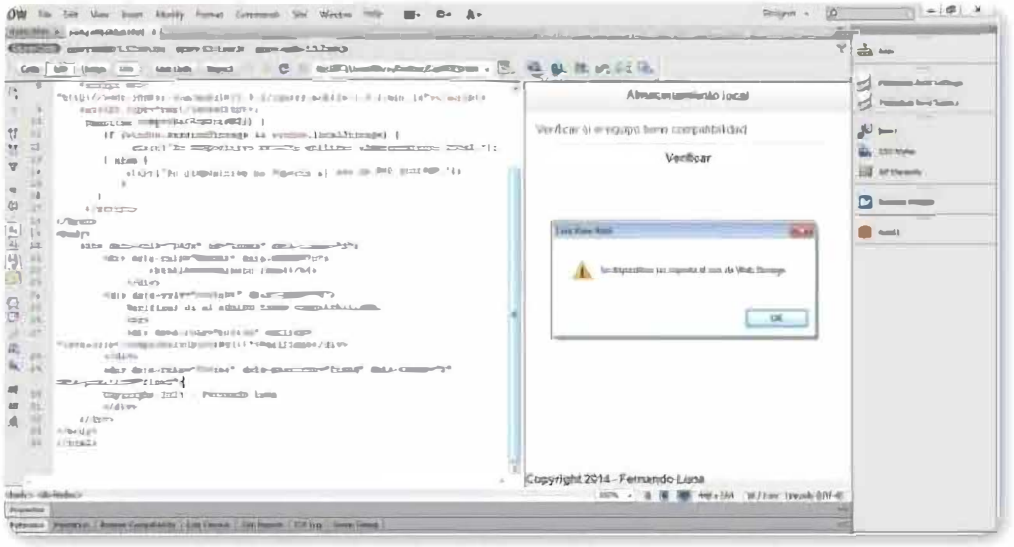


Figura 2. Si utilizamos Dreamweaver para el desarrollo, podemos verificar que el navegador integrado en esta herramienta no soporta el almacenamiento local.

A modo de ejemplo, desarrollaremos, a continuación, una pequeña web que nos permitirá verificar la compatibilidad de un navegador web con local storage y session storage. Veremos luego, implementando el código JavaScript en el que creamos la función **comprobarSoporteWS()**, si el navegador que tenemos instalado en nuestra computadora o teléfono móvil soporta las características mencionadas. Así, sabremos si nuestro navegador soporta el almacenamiento local o no.

EL SITIO WEB CAN I USE

La página web oficial **www.w3.org/TR/webstorage/#storage** nos permite conocer en detalle todas las funcionalidades que tenemos disponibles mediante el uso de Web Storage. Esta página web nos lista en forma automática información de aquellas sentencias que no conocemos en detalle del lenguaje HTML5.

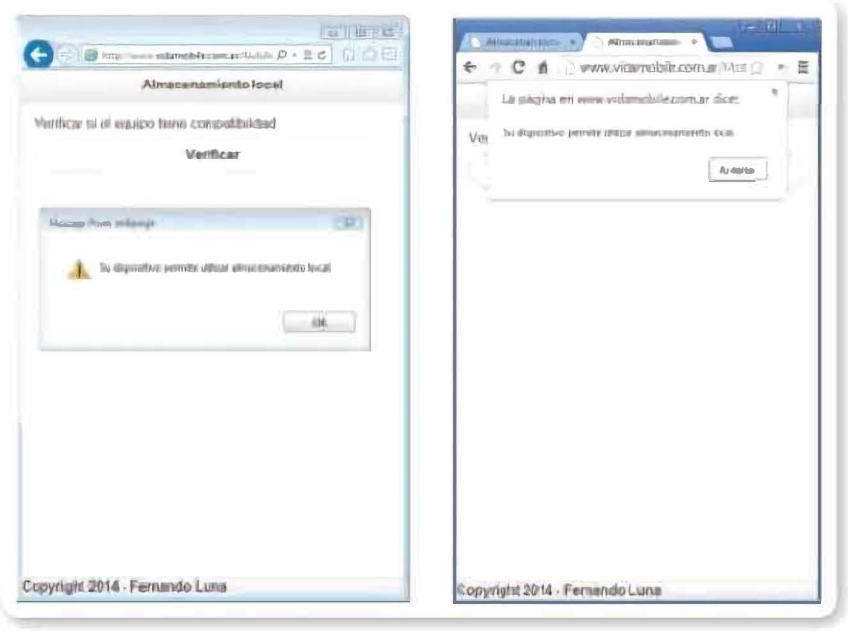


Figura 3. Tanto Internet Explorer como Google Chrome soportan el almacenamiento local propuesto por local storage y session storage.

Realicemos entonces una pequeña página web que nos informe si el navegador soporta o no el almacenamiento local. Para ello, agregaremos en una página simple un botón que contiene la función JavaScript que vimos anteriormente.

A continuación, el código de la página web:

```
<!DOCTYPE html>
<html>
<head>
   <title>Almacenamiento local</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
      <script type="text/javascript">
   ...
   //Aquí incorporamos el código JavaScript
   //de la función comprobarSoporteWS()
   ...
   </script>
```

```
    </head>
    <body>
      <div data-role="page" id="index" data-theme="d">
        <div data-role="header" data-theme="c">
              <h4>Almacenamiento local</h4>
             </div>
        <div data-role="content" data-theme="d">
          Verificar si el equipo tiene compatibilidad.
          <br>
          <div data-role="button" onClick="javascript:comprobarSoporteWS();">
Verificar</div>
        </div>
        <div data-role="footer" data-position="fixed" data-theme="c" data-
position="fixed">
        Copyright 2014 - Fernando Luna
        </div>
      </div>
    </body>
    </html>
```

En el apartado correspondiente al script, luego de la declaración de los componentes correspondientes a jQuery Mobile, agregamos el script correspondiente de verificación. Una vez realizado esto, ya estamos listos para subir la página web a nuestro hosting y cargarla en el navegador web local o en nuestro dispositivo móvil.

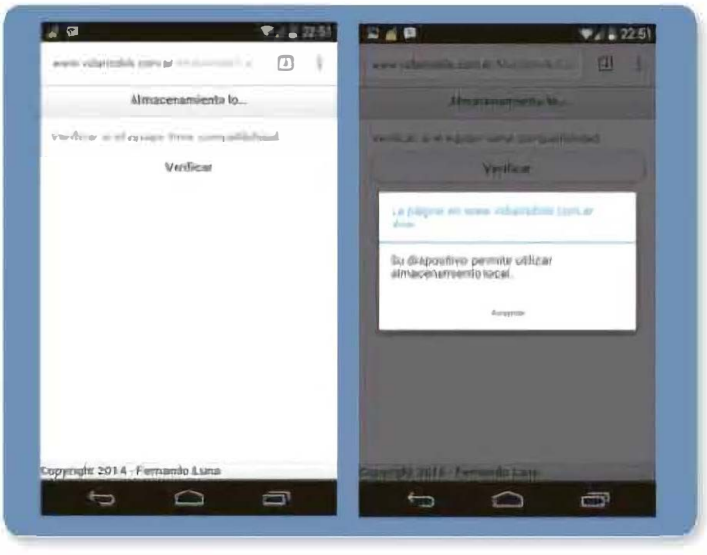


Figura 4. Google Chrome para Android también soporta el almacenamiento local.

Ejercicio práctico: almacenamiento local

A continuación, realizaremos un pequeño formulario que hará uso del almacenamiento local de los datos cargados en los respectivos campos. Para ello, agregaremos cuatro campos y dos botones. En los campos guardaremos datos de tipo texto, y los botones invocarán dos funciones. Una de ellas almacenará en el navegador web del cliente los datos cargados previamente en los campos, y la otra permitirá recuperar los datos almacenados mediante local storage.

Veamos el código HTML:

```
<!DOCTYPE html>
<html>
<head>
   <title>Almacenamiento local</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
   <script type="text/javascript">
            var storage = localStorage;

            function guardar() {
                  var clave = "nombre";
                  var valor = document.getElementById('nombre').value;
                  storage.setItem(clave, valor);
                  var clave = "apellido";
                  var valor = document.getElementById('apellido').value;
                  storage.setItem(clave, valor);
                  var clave = "correo";
                  var valor = document.getElementById('correo').value;
                  storage.setItem(clave, valor);
                  var clave = "telefono";
```

```
                var valor = document.getElementById('telefono').value;
                storage.setItem(clave, valor);
                alert('Todos los elementos fueron almacenados.');
                }

            function recuperar() {
                var valor = storage.getItem('nombre');
                document.getElementById('nombre').value = valor;
                var valor = storage.getItem('apellido');
                document.getElementById('apellido').value = valor;
                var valor = storage.getItem('correo');
                document.getElementById('correo').value = valor;
                var valor = storage.getItem('telefono');
                document.getElementById('telefono').value = valor;
                }
        </script>
    </head>
    <body>
      <div data-role="page" id="index" data-theme="d">
        <div data-role="header" data-theme="c">
            <h4>Almacenamiento local</h4>
           </div>
         <div data-role="content" data-theme="d">
            <form id="email" method="put" action="enviar" >
              <input type="text" id="nombre" name="nombre"
placeholder="Nombre" />
              <input type="text" id="apellido" name="apellido"
placeholder="Apellido" />
               <input type="email" id="correo" placeholder="su correo
electr&oacute;nico" />
              <input type="text" id="telefono" name="telefono"
placeholder="Telefono" />
              <div align="center">
              <br>
              <div data-role="button" data-theme="b" onClick="javascript:guard
ar();">Guardar</div>
              <br>
```

```
                <div data-role="button" onClick="javascript:recuperar();">Recup
erar</div>
                </div>
            </form>
        </div>
        <div data-role="footer" data-position="fixed" data-theme="c" data-
position="fixed">
        Copyright 2014 - Fernando Luna
        </div>
    </div>
  </body>
  </html>
```

Este formulario simple posee dos funciones. La primera, llamada **guardar()**, se ocupa de guardar, en la variable **valor**, los datos ingresados en cada uno de los campos. Luego utiliza la función **setItem(clave, valor)**, donde almacenará, bajo el mismo nombre del campo, el valor que ha sido ingresado en este.

La segunda función se llama **recuperar()** y se ocupa de volver a cargar, en cada uno de los campos, el valor almacenado mediante local storage, utilizando la función **getItem(valor)**.

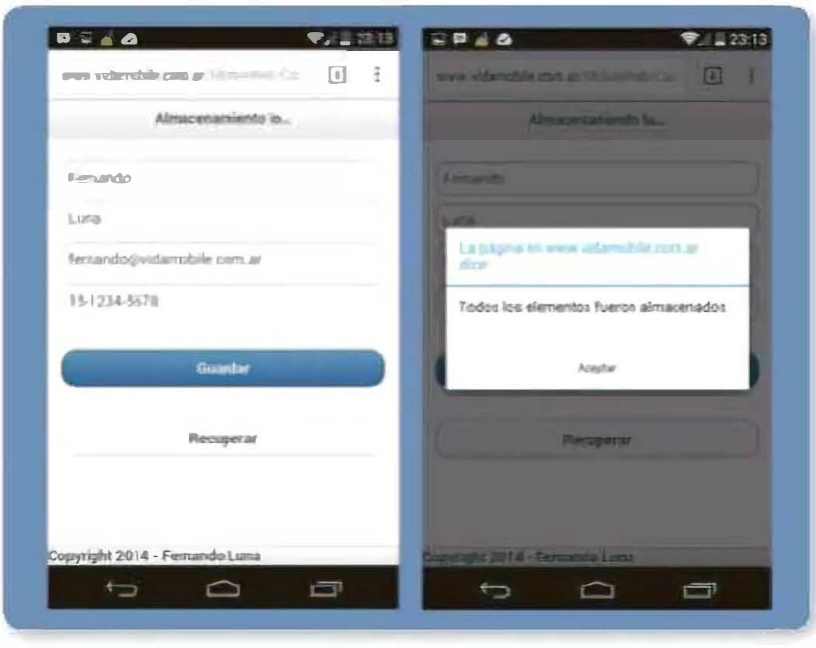


Figura 5. Al finalizar el almacenamiento local de las variables, JavaScript nos alerta con un mensaje que la tarea se llevó a cabo correctamente.

Si realizamos esta prueba en un equipo de escritorio, podemos acceder a la información de almacenamiento local que Google Chrome nos provee. Para esto, simplemente debemos ingresar a **Menú/Herramientas/Herramientas del desarrollador**.

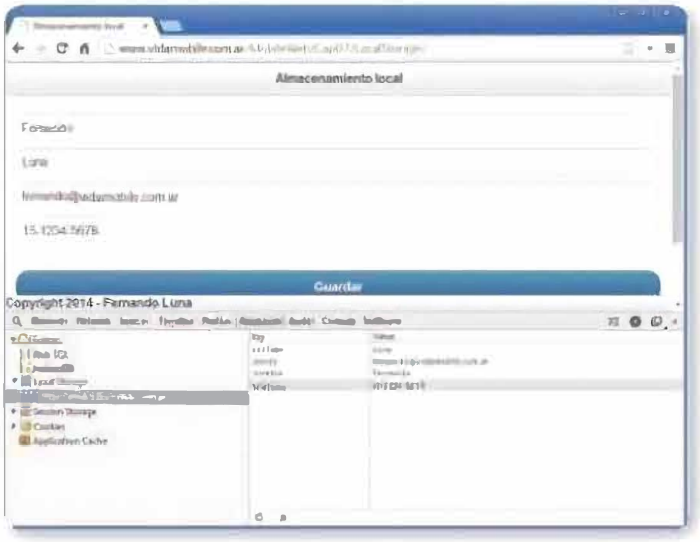


Figura 6. Podemos acceder a los datos de local storage seleccionando el sitio web desde el cual fue invocado el almacenamiento de la información.

Bases de datos Web SQL

Otra de las características fundamentales implementada junto con el lenguaje HTML5 es la posibilidad de utilizar bases de datos del tipo SQL creadas directamente en la computadora del usuario. Esto permite desarrollar una WebApp en la que, en base al manejo de grandes volúmenes de datos, la información constantemente utilizada pueda cargarse de forma local y, de esta forma, pueda estar disponible todo

INTERNET EXPLORER

Dada la diferencia que Internet Explorer siempre mantuvo con respecto a sus principales competidores, recomendamos consultar el sitio web **http://status.modern.ie** para explorar la documentación y compatibilidad de este navegador con el almacenamiento local.

el tiempo, sin sobrecargar el servidor web ni consumir un mayor ancho de banda.

Otra de las ventajas es que se pueden crear aplicaciones web y aplicaciones web móviles que funcionen completamente desconectadas del servidor remoto. La API de base de datos **Web SQL** se implementó para manipular estas bases de datos del lado del cliente, mediante peticiones SQL de forma asíncrona. Así, cualquier sentencia que conocemos y solemos utilizar en el desarrollo de base de datos –como ser **SQL Server**, **Oracle** o **MySQL**– también podrá ser ejecutada contra la base de datos Web SQL, de forma local, sin importar que esta sea un **Update**, **Create**, **Delete** o **Insert**.

> EN APLICACIONES WEB, EL USO DE WEB SQL DEBE REALIZARSE MEDIANTE CÓDIGO JAVASCRIPT

Sistemas operativos que soportan Web SQL

Actualmente, los sistemas operativos móviles que soportan la implementación de Web SQL son aquellos que pueden correr un navegador web basado en **WebKit**, como lo son Google Chrome, Safari (desktop y mobile), Opera, Opera Mini, Android Browser y Chrome para Android.

Por lo tanto, desde la perspectiva de generar aplicaciones web móviles, los sistemas operativos más populares que soportan esta característica son Android, iOS y Windows 8.x.

Lamentablemente, al momento de escribir esta obra, los navegadores web Internet Explorer, Internet Explorer mobile, Mozilla Firefox, Mozilla Firefox mobile, Mozilla Firefox para Android y BlackBerry Browser no soportan la implementación de Web SQL.

BLACKBERRY BROWSER Y WEB SQL

Si bien BlackBerry Browser está basado en WebKit, por el momento Web SQL no fue incluido en el soporte de esta plataforma. Debemos tener muy en cuenta esta información al momento de decidir si vamos a realizar una aplicación web móvil desconectada que utilice una base de datos Web SQL.

Figura 7. En esta imagen podemos observar, resaltados en color verde, los navegadores web que soportan el uso de Web SQL.

Manejo de sentencias Web SQL

El uso de Web SQL en aplicaciones web debe hacerse mediante código JavaScript, al igual que el resto de las características de almacenamiento que podemos realizar con HTML5. La creación de una base de datos, de una o más tablas y la lectura de registro, inserción, actualización y eliminación de datos deben estar agrupadas en diferentes funciones de JavaScript, para que –por supuesto– sea mucho más cómodo invocar cada una de ellas en el momento preciso.

Repasemos, a continuación, cada una de las sentencias a implementar para el uso de una base de datos Web SQL.

Cómo crear una base de datos Web SQL

En la creación de una base de datos Web SQL, debemos tener en cuenta que esta tendrá por defecto un tamaño de 5 megabytes. Este es el tamaño máximo predeterminado para las bases de datos locales, y para que este crezca por arriba de su valor máximo se requerirá la autorización del usuario del equipo, quien habilitará o no el crecimiento de la base de datos.

Se debe tener en cuenta esto cuando desarrollemos el proceso de inserción de datos. Si supera los 5 megabytes y el usuario no habilita el crecimiento de la base de datos, se producirá un error en nuestra aplicación, que deberá ser controlado de la forma más conveniente.

La creación de una base de datos Web SQL se realiza de la siguiente manera:

```
var webdb = {};
webdb.db = null;
webdb.open = function(opciones)
    {
    If (typeof openDatabase == "undefined") return;
            var opciones = options || {};
            opciones.name = opciones.name || 'nombredelabbdd';
            opciones.mb = opciones.mb || 5;
            opciones.desription = opciones.description || 'Descripción de la base
de datos';
            opciones.version = opciones.version || '1.0';          var dbSize
= opciones.mb * 1024 * 1024;
    }
```

Hasta aquí solo definimos cuáles son las características de la base de datos que estamos creando. Le asignamos un nombre, una descripción, el tamaño inicial en megabytes y la versión (inicialmente, 1.0).

Ahora nos queda abrir la base de datos. Veamos el código correspondiente para su apertura:

```
Webdb.db = openDatabase(opciones.name, opciones.version, opciones.description,
dbSize);
```

Mediante la función **openDatabase** podemos acceder a la base de datos que creamos anteriormente. Si la base de datos no existe en el equipo, se creará, según lo que establecimos en la definición de parámetros del bloque de código anterior. Por último, una vez que tenemos las sentencias de creación y/o apertura de la base de datos, nos queda ejecutar la operación declarada en la función **Function()**. Veamos el código correspondiente:

```
webdb.executeSql = function(sql, data, onSuccess, onError){
    if (!webdb.db) return;
    webdb.db.transaction(function(tx){tx.executeSql(sql, data, onSuccess, onEr-
ror);});
}
```

Cómo crear una tabla en una base de datos Web SQL

Una vez creada la base de datos y realizada su apertura mediante código JavaScript, podemos comenzar a crear tablas y a agregarles registros. Esto también se realiza desde JavaScript de una manera simple y muy similar a la forma de crear tablas mediante código en cualquier entorno de base de datos corporativa.

Veamos, a continuación, un ejemplo de esto:

```
webdb.executeSql('CREATE TABLE IF NOT EXISTS tabladeejemplo (ID INTE-
GER PRIMARY KEY ASC, texto TEXT, fecha_alta DATETIME"', [],
                        function(tx, r){
                                alert("Se ha creado la tabla: TABLADEEJEM-
PLO");
                        },
                        function(tx, e){
                                alert("Se ha producido un error al intentar
crear la tabla: "e.message);
                        });
```

Este simple bloque de código crea una tabla en la base de datos previamente abierta, llamada **TABLADEEJEMPLO**. Dentro de la tabla se define un campo **ID**, del tipo **integer**, al cual le establecemos la propiedad de clave primaria (o **Primary Key**). Luego definimos un campo **Texto**, del tipo **Text**, y un tercer campo llamado **Fecha_alta**, del tipo **DATETIME**. Ahora, nos resta agregar un registro de pruebas. Veamos, a continuación, el código para agregar registros mediante la sentencia **INSERT**:

```
Webdb.executesql('INSERT INTO TABLADEEJEMPLO (Texto, Fecha_alta)
VALUES (?,?)', ['Texto de ejemplo'], new Date()],
        function(tx, r) {
                        alert("Se ha agregado un nuevo registro.");
    },
        function(tx, e){
            alert("Se produjo un error en el alta: " + e.message);
    });
```

Indexed Database

Indexed DB es un nuevo método de almacenamiento de datos en HTML5. Las bases de datos web, al igual que en Web SQL, son almacenadas y actualizadas en el navegador web del usuario. Esto permite que los desarrolladores creen aplicaciones en las que, a través de simples consultas SQL, se puedan visualizar datos en el navegador de usuario más rápidamente. Haciendo uso de algunas funcionalidades de control offline y online de una conexión a internet, estas bases podrán actualizarse de manera transparente para el usuario que navega por la página web.

Figura 8. En este gráfico podemos apreciar el mapa de almacenamiento local compuesto por Web Storage (local y session), Indexed DB y Web SQL, según el navegador web que soporta cada uno.

Aplicaciones offline

Otro de los grandes cambios incluidos en HTML5 es que se puede trabajar con aplicaciones offline. Dado que el número de aplicaciones web crece día a día, y teniendo en cuenta todas las capacidades que este lenguaje de marcado desarrolló a lo largo de su última versión, solo faltaba incluir un soporte que le permitiera al usuario de una WebApp poder seguir trabajando desconectado, ya sea cuando pierde la conexión a internet, o simplemente para aprovechar el ahorro de datos en los equipos móviles.

La modalidad de trabajo de aplicaciones offline permite, en una primera instancia, descargar al dispositivo o equipo del usuario todas las páginas web, archivos JavaScript, hojas de estilo CSS, imágenes y archivos multimedia en general que puede contener un sitio web.

Esto se realiza una sola vez de forma completa; el resto de las veces que accedemos a la web, conectados a internet, se seguirán descargando aquellos archivos que han sido modificados en la versión en línea de la web o aquellos que fueron incorporados posteriormente.

Así, se realiza una actualización parcial de la WebApp en el equipo local y, en caso de que deseemos desconectarnos de internet para ahorrar datos, podremos hacerlo sin problema.

Cómo descargar una WebApp a un dispositivo

La descarga de los archivos para trabajar WebApps de forma offline se realiza incorporando un archivo de manifiesto junto a nuestra web. Este archivo contendrá el listado completo **directorio/subdirectorio/archivo.web** del contenido de toda la WebApp. De esta manera podremos descargar la aplicación y hasta incluir rutinas específicas para que la WebApp se comporte de manera diferente mientras está conectada o no a internet.

AppCache

El caché de aplicación o AppCache que se genera cuando desarrollamos una aplicación que trabaja sin conexión permite que el desarrollador especifique si todos los archivos que componen una WebApp o solo una parte de ellos deben almacenarse en el caché del navegador web, para que estén a disposición de los usuarios que eligen trabajar sin conexión en nuestro sitio.

Como ventaja, podemos destacar que cualquier aplicación que genere un caché local permitirá, por ejemplo, que el usuario navegue por este sitio sin conexión a internet. También permitirá que pueda acceder a los recursos, imágenes o contenido de texto más rápido, dado que estarán almacenados en su equipo y no en la red. Otra ventaja es que se generará una menor carga en el servidor, por lo que el navegador web sólo se ocupará de descargar del servidor aquellos recursos que hayan cambiado.

Archivo de manifiesto

Veamos, a continuación, un ejemplo de cómo debemos conformar el archivo de manifiesto que permita descargar el contenido de una web al caché del navegador local del usuario.

El primer cambio que debemos incorporar es en el tag **<html>** de la página web principal del sitio web o WebApp, para que el navegador identifique dónde se encuentra el archivo de configuración de la caché.

El código a utilizar es el siguiente:

```
<html manifest="http://www.misitioweb.com.ar/WebApp/example.mf">
...
</html>
```

Dentro del archivo de manifiesto podemos seleccionar la URL absoluta o relativa al contenido que descargaremos en el equipo cliente, pero siempre debemos tener en cuenta que, si utilizamos una URL absoluta, esta deberá tener una ruta hacia el mismo origen de nuestra WebApp.

Si nuestra WebApp está alojada en **www.misitio.com.ar/WebApp/index.html**, dentro del archivo de manifiesto no podremos declarar una imagen que esté alojada, por ejemplo, en **www.miotrositioweb.com.ar/imagenes/imagenweb.jpg**.

Otro cambio que debemos realizar es la modificación del archivo **.htaccess** de nuestro servidor web, agregándole la sentencia de tipo **MIME**, para que pueda relacionar la WebApp con los archivos de manifiesto:

```
AddType text/cache-manifest .appcache
```

Luego nos queda declarar el archivo de manifiesto completo, incluyendo en este cada uno de los archivos raíz, carpetas y subcarpetas con sus respectivos archivos que componen nuestra WebApp. Veamos un ejemplo a continuación:

```
CACHE MANIFEST
CACHE:
index.html
stylesheet.css
```

```
images/logo.png
images/imgtoolbar1.png
images/imgtoolbar2.png
images/imgtoolbar3.png
scripts/funcionesJS.js
scripts/variablesJS.js
```

Archivos online

Si necesitamos que la WebApp esté conectada para, por ejemplo, identificarnos en una red con un nombre de usuario y contraseña, podemos destacar esto dentro del archivo de manifiesto de la siguiente manera:

```
NETWORK:
login.php
/myapi
http://api.facebook.com
```

En caso de falla

Si necesitamos que algún contenido se consulte de manera online, y la red donde corre nuestra WebApp no lo permite, podemos incluir en el archivo de manifiesto un apartado que contemple esto y muestre una página o sentencia JavaScript alternativa:

```
FALLBACK:
/login.php /staticError.html
images/large/online.jpg images/offline.jpg
```

Actualización mediante JavaScript

También podemos aprovechar JavaScript para que nos ayude a actualizar una WebApp desde su versión online. Esto debemos combinarlo con sentencias que permitan detectar si el navegador está o no conectado a internet. En caso de que esté conectado a internet, simplemente invocamos la sentencia **applicacionCache_update()**.

Finalizada la actualización, nos queda cambiar al caché offline nuevo para que los datos en pantalla se actualicen. Para ello, invocaremos

la sentencia **applicacionCache.swapCache()**, la cual sustituirá la antigua caché por la nueva.

Veamos, a continuación, un ejemplo de JavaScript:

```
...
var appCache = window.applicationCache;

appCache.update(); // Actualizando la caché con los datos online.
if (appCache.status == window.applicationCache.UPDATEREADY) {
  appCache.swapCache();  // Actualización completa, visualizar la nueva caché.
}
...
```

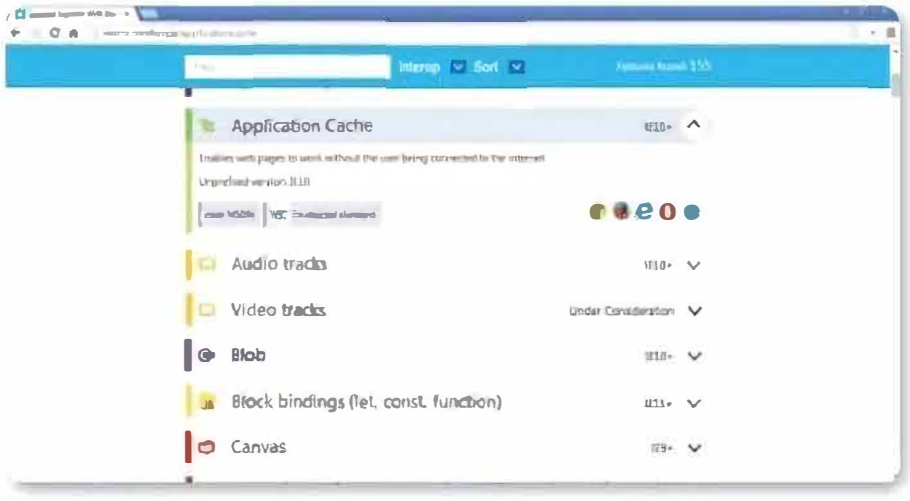


Figura 9. A partir de la versión 10 de Internet Explorer, este navegador comenzó a prestar soporte al almacenamiento offline del contenido de WebApps.

RESUMEN

Las diferentes opciones de almacenamiento local mediante claves, cookies, bases de datos indexadas o bases de datos Web SQL convierten a HTML5 en el lenguaje perfecto, que permite a una WebApp trabajar en línea y también sin conexión a internet. Con algunos ajustes básicos a nuestras aplicaciones y el aprovechamiento del almacenamiento local, podremos controlar en detalle el comportamiento de nuestras WebApps, para que nunca dejen al usuario desorientado ante un error o imprevisto no controlado durante la navegación web.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cuál fue, durante muchos años, el modo de almacenamiento local de datos más común en las aplicaciones web?
2. ¿Qué ventajas supone el uso de Web Storage respecto de una cookie?
3. ¿Cuántos tipos de almacenamiento local encontramos en Web Storage?
4. ¿Qué tipo de base de datos local podemos aprovechar al crear una WebApp?
5. ¿Qué es Indexed DB?

EJERCICIOS PRÁCTICOS

1. Investigue en profundidad el almacenamiento con Indexed DB.
2. Cree un formulario con campos que permitan insertar los datos cargados en una base de datos local Web SQL.
3. Desarrolle una función JavaScript que permita recuperar los datos almacenados en Web SQL para listarlos en pantalla.
4. Elija un proyecto desarrollado previamente en este libro y cree el archivo de manifiesto para utilizarlo en modo offline.
5. Investigue qué otras funciones JavaScript se pueden aprovechar con AppCache.

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

08

WebApps para iOS

En este capítulo, exploraremos las entrañas del sistema operativo para móviles y tablets de Apple, iOS. Conoceremos algunos secretos que nos permitirán explotar al máximo las funcionalidades del navegador web Safari, para que una web móvil pueda verse como una aplicación nativa, instalada en el sistema operativo de Apple.

Diseñar una WebApp para iOS

Como bien repasamos en el **Capítulo 1** de esta obra, se considera que **iOS** es el sistema operativo que marcó un quiebre importante en el ecosistema de los teléfonos móviles inteligentes a partir de 2007. iOS trajo consigo una batería de opciones que permitieron a cada desarrollador explotar la plataforma en tiempo récord, creando aplicaciones y soluciones web compatibles con el terreno que **Apple** propuso desde la creación de iPhone e iPod Touch.

Luego, en 2010, con la llegada de iPad, nuevamente Apple se mostró imbatible en el terreno mobile presentando un nuevo producto sólido y, a su vez, heredando gran parte de las características de **iPhone** e **iPod Touch**. Durante el nacimiento del sistema operativo iOS, Apple propuso una idea inicial que permitiera dotar de información lo más rápido posible al sistema operativo móvil iOS, con el objetivo de diferenciarse de todas las propuestas de smartphones que ya existían en el mercado. Apple planteó la idea de poblar de aplicaciones instalables el sistema operativo iOS, que estuvieran diseñadas por una comunidad de desarrolladores, respetando ciertamente los estándares impuestos por la empresa de Cupertino.

Estos estándares eran bastante exigentes para la época. Requerían de una capacitación inicial en el desarrollo en **XCode** + **Cocoa** para la plataforma iOS, para todo aquel nuevo desarrollador de la plataforma de la manzanita. Por eso, habilitaron una segunda opción para atraer usuarios al terreno del sistema operativo de Apple.

La nueva alternativa consistía en que cualquier aplicación web móvil o sitio web con información relevante pudiera adaptarse rápidamente e integrarse en el escritorio del sistema operativo iOS como si fuese una aplicación nativa. ¿La forma? Utilizar, en el desarrollo de la página web, determinadas etiquetas que permitieran al navegador web Safari detectar, por ejemplo, el icono distintivo de la web y el nombre, entre otros parámetros más, y así poder crear un acceso directo en el escritorio del smartphone de cada usuario.

La acogida de este conjunto de parámetros por la comunidad de desarrolladores y diseñadores web fue muy buena. En muy poco

tiempo lograron adaptar los sitios web al formato mobile para que fueran fácilmente instalados en los teléfonos inteligentes de Apple, hasta tanto otro equipo de desarrolladores pudiera llegar a diseñar una solución nativa de cada portal, producto o servicio para todo aquel cliente interesado en conquistar esta plataforma con sus productos y/o servicios.

Así fue como Apple supo balancear el ecosistema de su futura plataforma **iTunes**, la cual llegó al mercado algunos meses después con una base de decenas de miles de aplicaciones nativas. Sumado a este conjunto de etiquetas propuestas para la web móvil que lanzó Apple junto con iOS, llegaron otros concursos que buscaban talentos en el terreno del desarrollo móvil, y de estos surgieron alternativas a la plataforma que hasta hoy siguen vigentes, como **PhoneGap**.

El navegador Safari y sus prestaciones

El conjunto de etiquetas que Apple propuso a través de Safari marcaron un antes y un después en la plataforma. También en el resto de los navegadores web, que con el tiempo fueron incorporando soluciones basadas en estas etiquetas y otras más personalizadas. Veamos, a continuación, un repaso de estas etiquetas **<meta>** que permiten ver una aplicación web móvil como una aplicación nativa en el sistema operativo iOS.

Figura 1. El navegador **Safari Mobile**, disponible tanto en iPhone como en iPad y iPod Touch.

Viewport

La etiqueta **viewport** permite establecer el tamaño de visualización de una página web cuando esta se carga en el navegador web Safari para iOS. Este meta tag nos facilita establecer el ancho y la escala inicial con la cual se visualiza una página web en este navegador, recibiendo un parámetro inicial y uno final, que pueden diferir o ser iguales. Así, podemos establecer el ajuste del ancho de una página web para visualizarse en el browser y establecer una escala de **zoom** o no. Veamos un ejemplo:

```
<meta name = "viewport" content = "width = 320, initial-scale = 2.3, user-scalable = no">
```

La etiqueta **viewport** declarada establece un ancho de **320 pixeles**. La página se cargará aplicando un zoom inicial de **2.3** veces lo establecido por defecto. Por último, el atributo **user-scalable=no** impide realizar un zoom sobre dicha página. Si dicho atributo estuviera establecido en **yes**, el visitante de la página podría realizar zoom sobre el contenido web visualizado.

Establecer el ancho del dispositivo

Si, por ejemplo, quisiéramos que el ancho de nuestra web móvil se ajustara de manera predeterminada al ancho del dispositivo en el que se carga, podríamos establecer el atributo **width** para que se ocupe de detectar el ancho de la pantalla del dispositivo. Esto es útil cuando una web debe visualizarse, por ejemplo, en un **iPhone 3G**, **iPhone 4** e **iPhone 5**, cuyas pantallas varían en resolución.

```
<meta name = "viewport" content = "width = device-width">
```

LENGUAJE DE PROGRAMACIÓN PARA MAC

El desarrollo de aplicaciones nativas para OS-X y iOS se realiza sólo desde la plataforma OS-X a través de la aplicación **XCODE**, siendo este el entorno de desarrollo oficial que provee Apple. El lenguaje a utilizar en la codificación es **Objective-C**.

Las etiquetas **viewport** no son propias de la plataforma Apple. También son utilizadas en otras plataformas móviles y de escritorio, alcanzando por igual el mismo resultado que logramos sobre iOS.

Correr una WebApp a pantalla completa

Cuando el meta tag **apple-mobile-web-app-capable** se establece en **yes**, la aplicación web correrá en **modo pantalla completa**. El comportamiento predeterminado de este meta tag puede ser consultado utilizando JavaScript. De esta manera, podemos detectar si la WebApp corre o no a pantalla completa y, de acuerdo con este resultado, mostrar o no una funcionalidad que permita establecer la web como aplicación nativa. Veamos un ejemplo:

```
<meta name="apple-mobile-web-app-capable" content="yes">
```

Utilizando el tag, a través de una funcionalidad del navegador web Safari Mobile, podemos establecer que la web móvil sea visualizada a pantalla completa. Si queremos mostrar un botón de ayuda en esta web para que sea instalada por el usuario en el escritorio de iOS, a través de la propiedad de solo lectura **window.navigator.standalone** detectaremos si la web móvil se visualiza o no a pantalla completa.

Definiendo el estilo de la barra de estado

También existe un meta tag que permite definir el color de la **barra de estado** de iOS, donde se visualiza el nombre del dispositivo, la conectividad y la hora. Veamos cómo personalizar la barra de estado desde una web móvil, de la siguiente manera:

```
<meta name="apple-mobile-web-app-status-bar-style" content="black">
```

Por defecto, el meta tag representado muestra la barra de estado de iOS en color negro. Podemos cambiarlo a su color por defecto estableciendo el atributo **content="default"**, o podemos volver translúcida la barra de estado ajustando el atributo a **content="black-translucent"**. Si no especificamos este meta tag, su valor por defecto asignado por el navegador Safari es **content="black"**.

Format Detection

La plataforma iOS, específicamente en iPhone, detecta de manera automática todos los números de teléfono visualizados en una página web. Los transforma al hipervínculo **tel:5491142344567**, que permite, con un simple toque en la pantalla, iniciar la llamada a ese número de teléfono. Este parámetro también puede ser controlado de manera automática por nuestra aplicación, deshabilitando dicha funcionalidad establecida por defecto en Safari Mobile. Esto se realiza de la siguiente manera:

```
<meta name="format-detection" content="telephone=no">
```

Iconos para nuestra web móvil

También es importante, para visualizar una web móvil a pantalla completa en iOS, definir un icono determinado que permita identificar la WebApp en la pantalla de iOS. Si bien podemos no especificar nada y dejar que Safari Mobile se ocupe de asignar un icono determinado a nuestra WebApp, esto no es conveniente porque el navegador web solo establecerá como icono una captura de pantalla de nuestra web móvil.

Si bien la captura puede lucir espléndida, las buenas prácticas siempre definen que toda empresa, producto o servicio debe tener un **isologo** específico que permita identificar rápidamente la marca; por lo tanto, será conveniente también llevar esta práctica a la plataforma iOS. Debemos tener en cuenta, al momento de diseñar un icono para nuestra web móvil, que esta plataforma posee actualmente un amplio abanico de dispositivos en el mercado, con diferentes resoluciones de pantalla.

> TODA EMPRESA, PRODUCTO O SERVICIO DEBE TENER UN ISOLOGO ESPECÍFICO, TAMBIÉN EN IOS

Quienes están acostumbrados a desarrollar soluciones web, seguramente alguna vez crearon un **SHORTCUT ICON**. La propuesta de Apple es similar, con la diferencia de que no se limita el icono a los **32 x 32 pixeles** que la web de escritorio requiere, sino que, dependiendo de la versión de iOS –o, mejor dicho, de la versión del dispositivo móvil (iPad2, New iPad, iPad Air,

iPad Mini, iPhone 3GS, iPhone 4S, etcétera)–, el icono deberá tener un tamaño específico que se ajuste a mostrar una resolución óptima en cada pantalla. Veamos, a continuación, la información correcta para los iconos que debemos crear:

TAMAÑOS INDICADOS PARA ICONOS

DISPOSITIVO APPLE	ANCHO Y ALTO DE LA IMAGEN
iPhone / iPod Touch (no Retina)	57 x 57 pixeles
iPhone / iPod Touch (Retina)	114 x 114 pixeles
iPad (no Retina)	72 x 72 pixeles
iPad (Retina)	144 x 144 pixeles

Tabla 1. Tamaño de iconos de pantalla según el dispositivo Apple.

Como podemos observar, el tamaño de cada icono varía según si el dispositivo posee o no una pantalla del tipo **Retina**. Estas necesitan el doble de resolución que las pantallas comunes, dado que duplican la densidad de pixeles representados en un espacio reducido: por lo tanto, si utilizamos un icono no preparado para las pantallas Retina, este se verá pixelado cuando nuestra WebApp se instale en un dispositivo que sí cuente con una pantalla de alta densidad de pixeles. Esto hace que cuando deseemos agregar una WebApp a la pantalla de un dispositivo iOS debamos crear estos cuatro iconos en tamaños diferentes. El navegador Safari Mobile decidirá cuál es el icono correcto que debe asignarle a nuestra aplicación al momento de crear el acceso directo.

NACIMIENTO DE PHONEGAP

El framework más popular para el desarrollo de aplicaciones híbridas, **PhoneGap**, nació de un concurso organizado originalmente por Apple para potenciar, con determinadas herramientas, las WebApps desarrolladas para la plataforma iOS, explotando al máximo los meta tags que ofrecía el navegador Safari Mobile.

Cómo crear el icono

Para no complicarnos de entrada en la creación del icono correcto para la pantalla iOS, lo primero que debemos hacer es armar la imagen base estableciendo una cantidad mayor de pixeles al máximo definido en la API del dispositivo. En este ejemplo, armaremos una imagen base de **200 x 200 pixeles**, que luego redimensionaremos desde nuestro editor de imágenes preferido, guardando cada nuevo tamaño con un nombre de archivo que lo referencie.

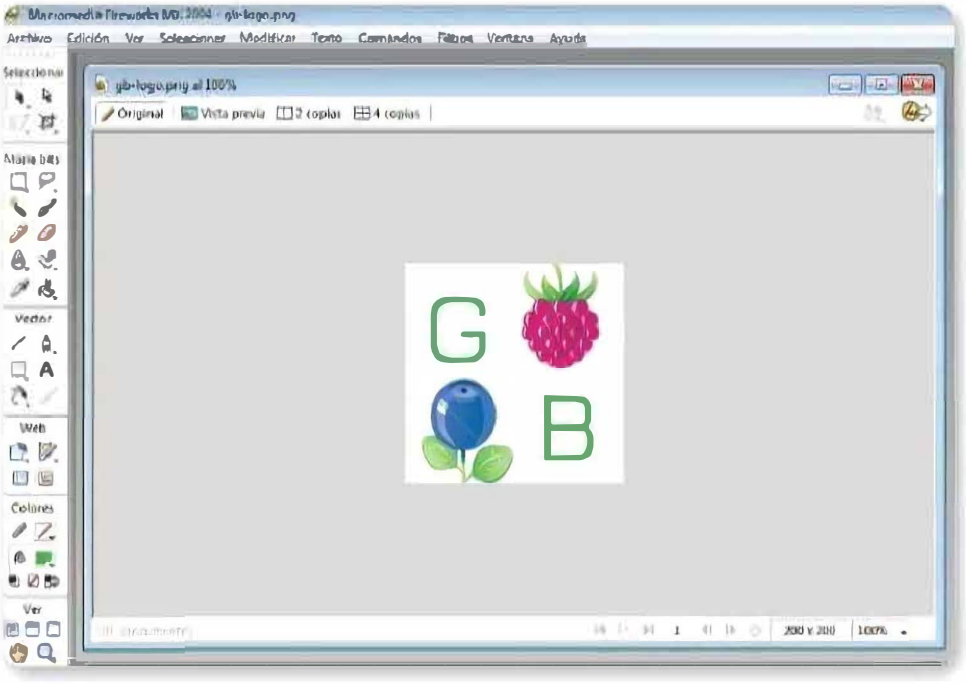


Figura 2. El icono de nuestra WebApp creado en alta resolución, a partir del cual iremos reduciendo sus dimensiones para los distintos dispositivos iOS.

Para ahorrar pasos en el próximo ejercicio a realizar, debemos descargar el proyecto asignado a esta unidad desde **http://premium.redusers.com**. Allí encontraremos el proyecto a modificar y la imagen correspondiente en tamaño superior, para luego redimensionarla para cada una de las versiones de pantalla del sistema operativo iOS que deseemos cubrir.

El navegador Chrome en iOS

El navegador **Google Chrome** tiene su propia versión para el sistema operativo iOS. Pero, según las políticas de Apple, puede

correr en este sistema operativo si utiliza el motor nativo del navegador web Safari Mobile, y no su propio motor **WebKit**. Por lo tanto, hay muchas de las funcionalidades de Google Chrome con las cuales no podremos contar en iOS.

> MUCHAS DE LAS FUNCIONALIDADES DE GOOGLE CHROME NO ESTÁN DISPONIBLES EN IOS

Una de las características ausentes en Google Chrome para iOS es la instalación y ejecución de extensiones creadas para el navegador web de Google. Tampoco contamos con la posibilidad de agregar una WebApp a la pantalla de iOS desde Google Chrome, por ser esta una función existente ya en Safari Mobile.

Por lo tanto, cada vez que ejecutemos una WebApp instalada en iOS, esta solo correrá utilizando el navegador web Safari Mobile y el soporte que este brinda para HTML5 y JavaScript, que generalmente difiere de Google Chrome y de otros navegadores web existentes.

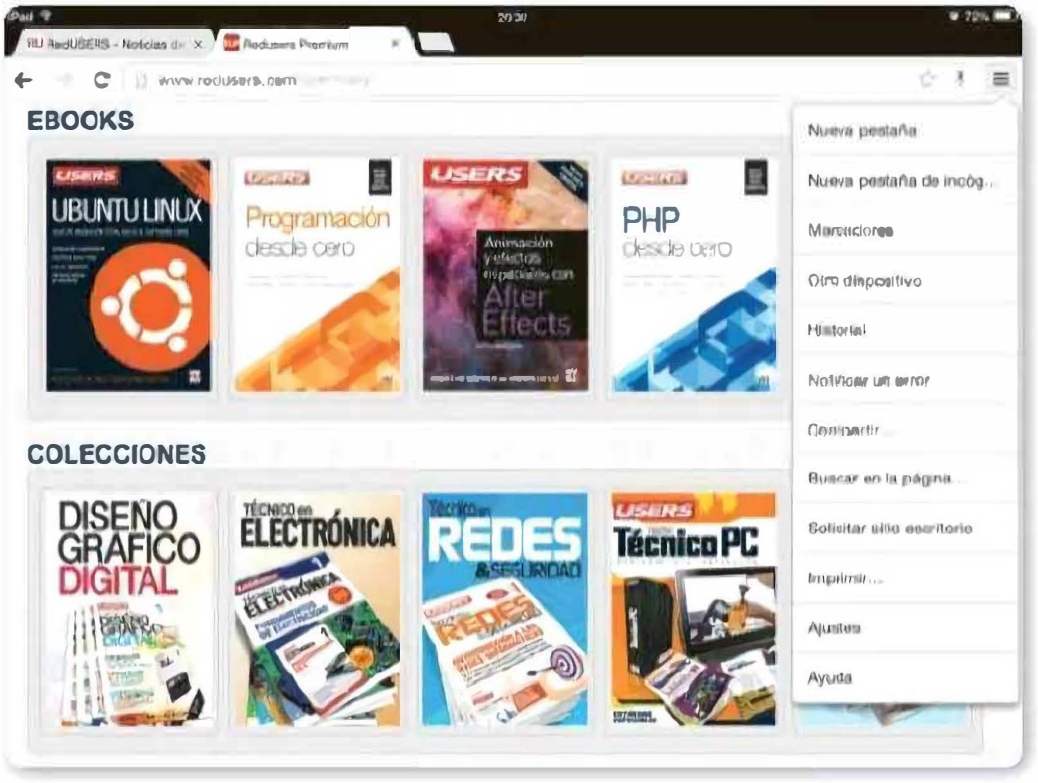


Figura 3. El navegador **Google Chrome** corriendo en iOS. Por políticas de Apple, no contamos con la posibilidad de instalar WebApps en iOS desde Chrome.

Ejercicio integrador: Add to home screen

A continuación, adaptaremos el ejercicio realizado en los primeros capítulos de este libro, **Green&Berries Farming**, para que pueda ser instalado en los dispositivos iOS, independientemente del tipo de pantalla que estos tengan. Si ya disponemos de los archivos de este ejercicio, descargados desde el portal **https://premium.redusers.com**, encontraremos la carpeta del proyecto **Greenberries** y otra carpeta denominada **iconos**.

En esta última, encontraremos una imagen llamada **gb-logo.png**, que debemos abrir con nuestro editor de imágenes preferido y redimensionarla, primero, a **144 x 144 pixeles**. Una vez realizado esto, debemos guardarla con el siguiente nombre: **gb-logo-144x144.png**. Luego debemos redimensionarla nuevamente a **114 x 114 pixeles** y guardarla con el nombre **gb-logo-114x114.png**.

A continuación, volvemos a redimensionar la imagen a **72 x 72 pixeles** y la guardamos con el nombre **gb-logo-72x72.png**. Por último, a **57 x 57 pixeles** y la guardamos como **gb-logo-57x57.png**. El resultado obtenido debe ser similar al que visualizamos en la **Figura 4**.

Figura 4. El icono creado con sus distintas dimensiones, especificadas en el nombre del archivo.

Mostrar la WebApp como nativa en iOS

Ya con el repaso por las prestaciones del navegador Safari Mobile y con los iconos creados y redimensionados correctamente, podemos adaptar nuestra WebApp para instalarla en la pantalla del sistema operativo iOS. Lo primero que debemos realizar es abrir el proyecto que descargamos desde **RedUsers Premium** en nuestro editor web favorito.

Luego, debemos editar el archivo **index.html** de nuestro proyecto, al cual le agregaremos el código necesario para poder instalar la WebApp en la pantalla de iOS. Ubicamos dentro de este código el apartado **<HEAD>** del archivo HTML e inmediatamente debajo de la línea de código **<meta name="viewport" content="width=device-width, initial-scale=1">**, agregamos el siguiente código:

```
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
```

Este código ya nos permite definir que nuestra WebApp es un sitio web apto para instalarse dentro de la pantalla principal del sistema operativo iOS. Luego definimos que la barra de tareas superior de iOS sea translúcida. Podemos optar también por que sea negra o del color por defecto del sistema operativo.

Icono de la aplicación web

Seguido al código agregado en el archivo **index.html**, debemos sumar la descripción para cada uno de los iconos redimensionados para las distintas pantallas de iOS. Para ello, agregamos el siguiente código:

```
<link rel="apple-touch-icon" sizes="57x57" href="gb-logo-57x57.png" />
<link rel="apple-touch-icon" sizes="72x72" href="gb-logo-72x72.png" />
```

NAVEGADORES WEB PARA IOS

Todos los navegadores web que existen actualmente para la plataforma iOS utilizan como **motor de renderizado** de las páginas web la versión **WebKit** de Safari Mobile. Es un requisito de Apple. Chrome y Opera Mobile también son obligados a utilizar este motor web y no sus propias versiones.

```
<link rel="apple-touch-icon" sizes="114x114" href="gb-logo-114x114.png" />
<link rel="apple-touch-icon" sizes="114x144" href="gb-logo-144x144.png" />
```

Con esto ya tenemos definido el icono para cada una de las posibles pantallas de los dispositivos que corren iOS, incluyendo hasta la generación de **iPad Mini** e **iPad Air**, los últimos modelos de iPad lanzados por Apple al momento de escribir esta obra. Seguramente las futuras versiones de iPad aumentarán la resolución de su pantalla y hasta agrandarán el área de visualización, lo que puede traer cambios en la resolución correcta del icono a crear.

Llegado el momento, solo debemos tomar el archivo **PNG** base que utilizamos al momento de desarrollar los ejercicios de este capítulo y adaptarlo a las futuras nuevas dimensiones, agregando la correspondiente línea de código en la página **index.html**.

HASTA LA VERSIÓN 6.X DE IOS, APPLE INCLUÍA TANTO ICONOS COMUNES COMO PERSONALIZADOS

Iconos prediseñados

Hasta la versión **6.x** del sistema operativo iOS, Apple incluía dos tipos de iconos que se podían crear para identificar una WebApp instalada en la **home screen**: los iconos comunes como los que creamos en este ejercicio y los personalizados, con forma y diseño creado por uno mismo y que no pueden ser modificados por el navegador Safari Mobile.

A partir de la versión **7.0**, en la que sufrió un rediseño importante, mucha de la estética **3D** que se aplicaba a los iconos cuando las apps o WebApps eran instaladas en la home screen de este sistema operativo ya no es tan visible, dado que **iOS 7.0** o superior aplica actualmente una estética denominada **2D** en el diseño de los iconos. Lo único que maneja esta nueva estética en paralelo a su versión anterior es el borde redondeado de cada icono.

Si deseamos que nuestra WebApp posea características determinadas en su icono, y que este no sea intervenido por iOS, debemos utilizar la leyenda **-precomposed** al final del nombre de archivo de nuestros iconos. El nombre quedaría, por ejemplo, como **gb-logo-57x57-precomposed.png**.

El código correspondiente al icono **precomposed** debe quedar de la siguiente manera:

```
<link rel="apple-touch-icon" Sizes="57x57" href="gb-logo-57x57-precomposed.
png" />
```

Recordemos que esto sólo funcionará para las versiones de iOS anteriores a la 7.0.

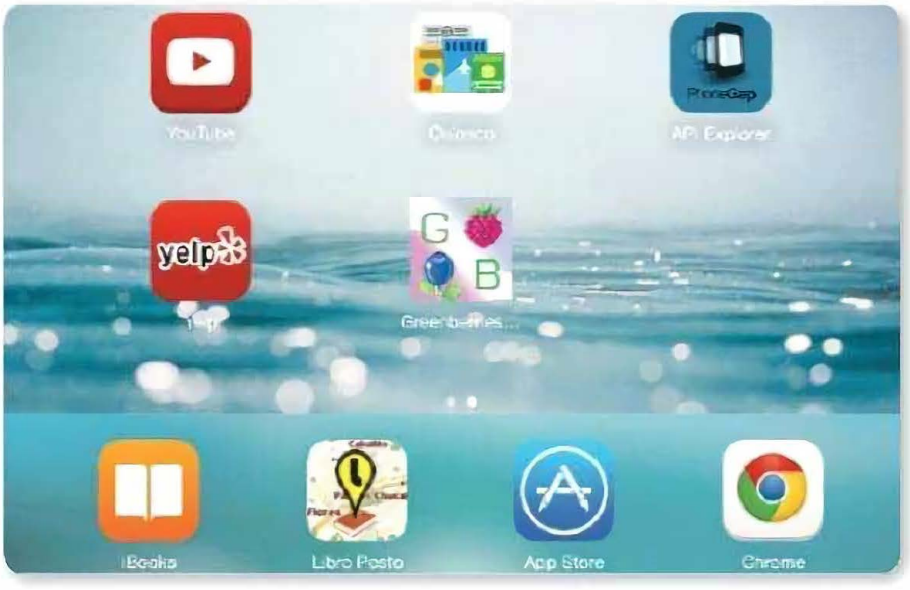


Figura 5. Hasta la versión 6.x de iOS, los iconos con efecto **precomposed** eran válidos para respetar el diseño personalizado de la web y no forzar a tener la estética de iOS. Desde la versión 7.0 esto ha sido discontinuado.

Splash screen de la aplicación

Al igual que en las aplicaciones nativas o híbridas que se instalan desde la tienda de aplicaciones **App Store**, los meta tags del navegador Safari Mobile nos permiten establecer una imagen de inicio de WebApp, conocida como **splash screen**. Estas imágenes se mostrarán en el dispositivo en el momento en que se ejecuta el inicio de nuestra WebApp y estarán visualizadas algunos segundos mientras se carga la página principal de la aplicación. A diferencia de lo que vimos con el icono de la WebApp, las imágenes referentes a la splash screen se segmentan por dispositivo (iPhone/iPod o iPad) según el tipo de pantalla.

Splash screen para iPhone o iPod

La splash screen para iPhone o iPod solo soporta la orientación **portrait** (**portarretrato** o **vertical**). Esto ocurre porque las WebApps no pueden ser iniciadas en modo **landscape** (**apaisado** u **horizontal**). Por lo tanto, mientras se inicia la WebApp desde la pantalla principal de iOS, la visualización de la imagen splash screen se hará en modo portrait, sin importar si el dispositivo se encuentra en modo apaisado o vertical.

El tamaño de las imágenes recomendadas por Apple para splash screen a visualizar en los dispositivos iPhone o iPod que no poseen pantalla Retina es de **320 x 460 pixeles**, mientras que las imágenes splash screen para los dispositivos iPhone que sí poseen pantalla Retina tienen exactamente el doble de resolución: **640 x 920 pixeles**. El código para visualizar las imágenes splash screen para iPhone es el siguiente:

```
<!-- iPhone -->
<link rel="apple-touch-startup-image" media="(device-width: 320px)"
href="apple-touch-startup-image-320x460.png">
<!-- iPhone (Retina) -->
<link rel="apple-touch-startup-image" media="(device-width: 320px) and (-web-
kit-device-pixel-ratio: 2)" href="apple-touch-startup-image-640x920.png">
```

Comparando uno con el otro, notamos que se utiliza un **media query** de CSS para especificar la imagen correspondiente a iPhone Retina. Esto se debe a que en todos los dispositivos iPhone, tengan o no pantalla Retina, cuando se inicia Safari Mobile, este reporta a JavaScript una resolución de pantalla correspondiente a **320 x 460 pixeles**, notando la diferencia entre una pantalla Retina y una no Retina. La propiedad **Pixel Ratio** tiene un valor **2** para las pantallas de alta resolución, mientras que para las pantallas no Retina posee un valor **1**.

MOBILE-WEB-CAPABLE

El uso de las WebApps que prescinden de todas las funcionalidades del navegador web en los dispositivos móviles se fue propagando hacia el resto de los navegadores móviles de las plataformas Android, Symbian y BlackBerry 10, tomando la premisa impuesta por Apple desde el año 2007.

Splash screen para iPad

A diferencia de lo visto con los dispositivos iPhone e iPod, en las tablets iPad sí se puede configurar una imagen splash screen específica para el dispositivo que se encuentra apaisado y otro para el dispositivo que se encuentra en posición vertical. La técnica a aplicar es similar a la descripta para iPhone, con excepción de los iPad que reportan una resolución de **768 x 1024 pixeles.**

Veamos entonces un ejemplo de código para los iPad apaisados, con pantalla común y pantalla Retina, y para aquellos iPad que se encuentran en posición vertical, diferenciando también el tipo de pantalla.

```
<!-- iPad (vertical/portrait) -->
<link rel="apple-touch-startup-image" media="(device-width: 768px) and (orientation: portrait)" href="apple-touch-startup-image-768x1004.png">
<!-- iPad (Apaisado/landscape) -->
<link rel="apple-touch-startup-image" media="(device-width: 768px) and (orientation: landscape)" href="apple-touch-startup-image-1024x748.png">
<!-- iPad (Retina, vertical/portrait) -->
<link rel="apple-touch-startup-image" media="(device-width: 768px) and (orientation: portrait) and (-webkit-device-pixel-ratio: 2)" href="apple-touch-startup-image-1536x2008.png">
<!-- iPad (Retina, apaisado/landscape) -->
<link rel="apple-touch-startup-image" media="(device-width: 768px) and (orientation: landscape) and (-webkit-device-pixel-ratio: 2)" href="apple-touch-startup-image-2048x1496.png">
```

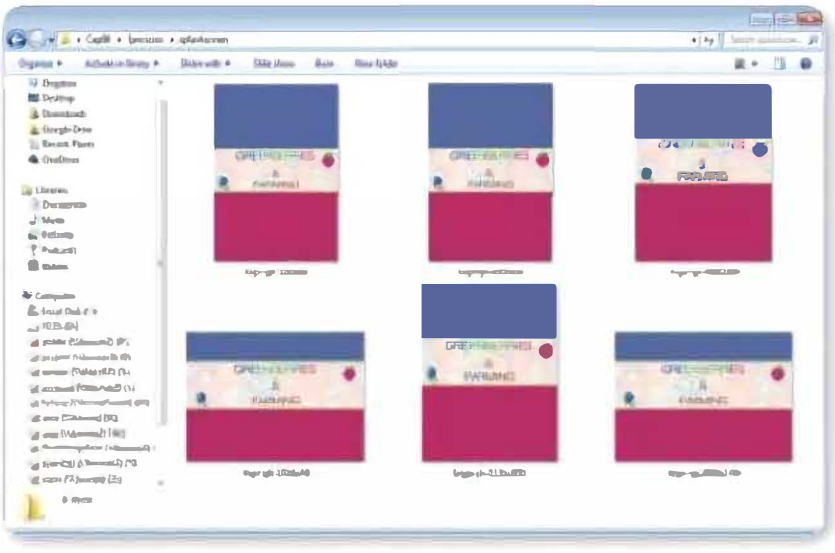


Figura 6. Ya tenemos todas las variantes de imágenes splash screen listas para incorporar a nuestra WebApp.

Agregar splash screen a nuestro proyecto

Ya con el concepto básico de cómo aplicar una pantalla de inicio a nuestra WebApp, crearemos a continuación las pantallas correspondientes que serán aplicadas a nuestro proyecto **Green&Berries Farming**, para que este responda como una aplicación nativa cuando sea agregada a la pantalla principal de un dispositivo iOS.

CREAREMOS UNA IMAGEN POR CADA RESOLUCIÓN REPRESENTADA EN EL EJEMPLO

Para ello, crearemos una imagen por cada una de las resoluciones representadas en el ejemplo. Dentro del material adicional que se puede descargar en **https://premium.redusers.com** encontraremos, dentro de la carpeta **splashscreen**, unas imágenes creadas para este propósito.

Ya con los archivos definidos, pasamos a incorporar el correspondiente código al archivo **index.html** de nuestro proyecto.

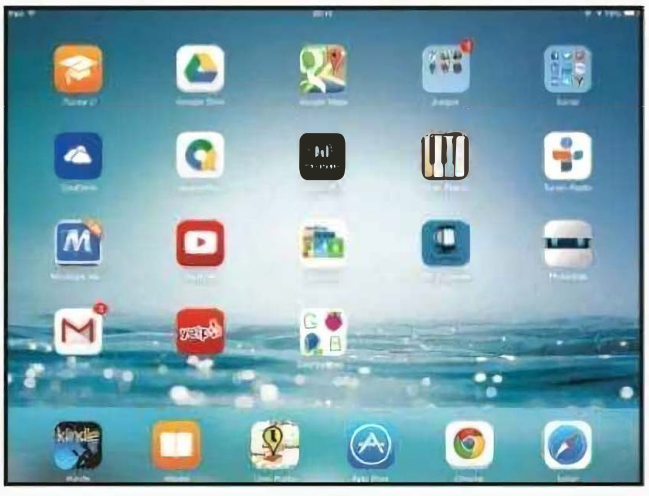
A continuación agregamos, a los iconos definidos para nuestra aplicación, el código correspondiente a las imágenes splash screen:

```
<!-- Código splash screen para iPhone-iPod -->
<link rel="apple-touch-startup-image" media="(device-width: 320px)" href="logo-gb-320x460.png">
<link rel="apple-touch-startup-image" media="(device-width: 320px) and (-webkit-device-pixel-ratio: 2)" href="logo-gb-640x920.png">
<!-- Código splash screen para iPad -->
<link rel="apple-touch-startup-image" media="(device-width: 768px) and (orientation: portrait)" href="logo-gb-768x1004.png">
<link rel="apple-touch-startup-image" media="(device-width: 768px) and (orientation: landscape)" href="logo-gb-1024x748.png">
<link rel="apple-touch-startup-image" media="(device-width: 768px) and (orientation: portrait) and (-webkit-device-pixel-ratio: 2)" href="logo-gb-1536x2008.png">
<link rel="apple-touch-startup-image" media="(device-width: 768px) and (orientation: landscape) and (-webkit-device-pixel-ratio: 2)" href="logo-gb-2048x1496.png">
```

PAP: AGREGAR UNA WEBAPP A LA PANTALLA DE INICIO DE IOS

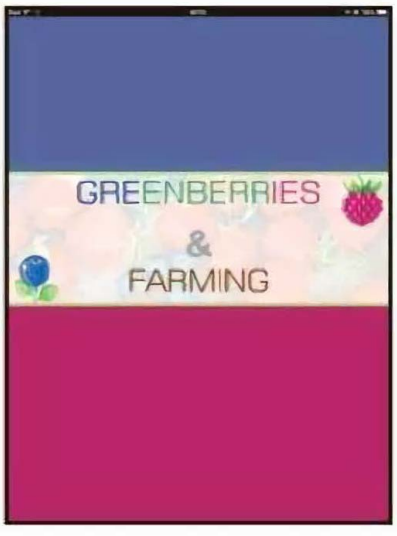
01 Presione el botón `Compartir` del navegador Safari Mobile y, desde el menú emergente, seleccione la opción `Añadir al inicio`. A continuación, deberá verificar que el icono y el título a mostrar sean los mismos que configuró a través del HTML.

02 Una vez agregada la WebApp a la pantalla de inicio, verá el icono de la aplicación junto con el resto de los iconos de aplicaciones instalados. Podrá ingresar a su WebApp desde este acceso, sin tener que escribir la URL en el navegador web de iOS.

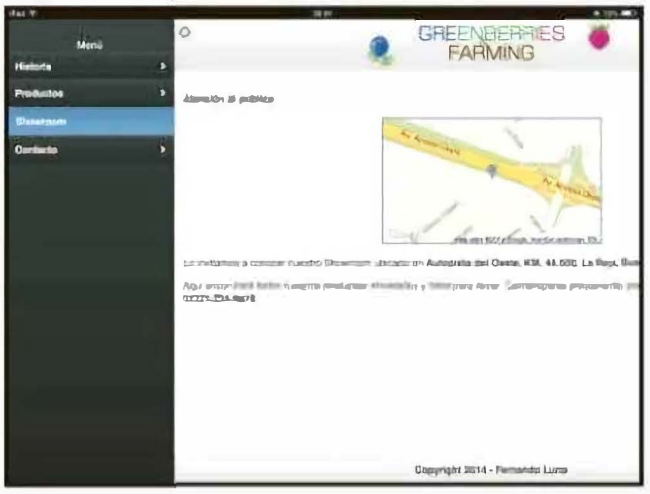


03 Al ejecutarse la WebApp, Safari Mobile visualizará en pantalla el **splash screen** correspondiente al dispositivo desde el cual se inicia la WebApp.

04 Para el caso de los dispositivos iPad, la imagen splash screen que se visualizará será la correspondiente a la posición de la pantalla del dispositivo (**portrait** o **landscape**).



05 Finalmente, ya podrá utilizar su WebApp como si fuese una aplicación instalada localmente en iOS, sin necesidad de visualizar la barra de herramientas o barra de direcciones del navegador web Safari Mobile.



RESUMEN

A lo largo de este capítulo, repasamos los conceptos básicos para poder adaptar una WebApp y agregarla a la pantalla de inicio de iOS, con su respectivo icono y splash screen, prescindiendo de la funcionalidad completa y de las herramientas que otorga un navegador web como Safari. También aprendimos a crear los diferentes iconos y pantallas de inicio, dependiendo del tipo de dispositivo iOS donde nuestra WebApp pueda llegar a ejecutarse.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Para qué sirve el meta tag **viewport**?
2. ¿Son propias de Apple las etiquetas **viewport**?
3. ¿Qué beneficio nos brinda incluir en una página web **apple-mobile-web-app-capable**?
4. ¿Qué utilidad representa el atributo **content="black-translucent"**?
5. ¿Cómo podemos desactivar el formateo automático de números telefónicos de páginas web en iPhone?
6. ¿Qué utilidad nos brinda el atributo **–precomposed** en un icono?
7. ¿Sirve utilizar el atributo **–precomposed** en las imágenes **splash screen**?
8. De todo un sitio web, ¿qué archivo debemos modificar para incorporarle los comandos necesarios que permitan instalarlo en la pantalla de iOS?
9. ¿Para qué plataforma sirve un icono de **57 x 57 pixeles**?
10. ¿En qué plataformas podemos utilizar imágenes splash screen horizontales y verticales?

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*09

WebApps para Android y otros dispositivos

En este capítulo aprenderemos los conceptos básicos que es necesario incorporar a una WebApp para poder instalarla en la pantalla principal de los dispositivos Android. También veremos qué otra funcionalidad similar nos brindan las plataformas BlackBerry 10 y los dispositivos Windows Phone.

Diseñar una WebApp para Android

La llegada del sistema operativo Android versión **4.0** trajo la madurez que los usuarios de teléfonos inteligentes y tablets estaban esperando del sistema operativo más utilizado en todo el mundo. También resultó una evolución significativa para muchos desarrolladores de aplicaciones nativas, híbridas y WebApps, puesto que debían actualizar determinadas características como el procesador del equipo y el software disponible en él.

Y, centrándonos en el terreno de los desarrolladores web, estos requerían un fuerte cambio en el navegador web por defecto del equipo, que era, desde las primeras versiones de este sistema operativo, **Android Browser**. Se trata de un navegador web que cumplía con las características básicas de visualizar WebApps y sitios web orientados a dispositivos móviles, pero que no mostraba ninguna evolución en cuanto al motor de renderizado, dejando de lado un sinfín de características que **JavaScript**, **HTML5** y **CSS3** ponían a disposición del desarrollador web orientado al terreno móvil.

Google Chrome para Android: la estrella esperada

A principios de 2012, cuando ya estaba en camino el sistema operativo móvil **Android 4.0**, Google anunció el lanzamiento de la versión beta de Google Chrome para esta plataforma. Chrome finalmente llegó para quedarse en los dispositivos Android, trayendo consigo varias de las características destacadas de su versión de escritorio, entre las cuales podemos destacar la sincronización automática de marcadores y sitios web visitados y la incorporación de una funcionalidad que permite acceder a las páginas visitadas en otros dispositivos.

Si bien Android 4.0 seguía teniendo como navegador web por defecto a Android Browser, a partir de la versión **4.1** de este sistema operativo, Google Chrome tomó las riendas y pasó a tener la exclusividad de navegación en el terreno móvil. Todos estos cambios

se dieron en el momento en que Android empezó a ganar espacio entre los dispositivos móviles y en el que Google Chrome comenzaba a ser el navegador de escritorio más utilizado a nivel mundial.

Debemos destacar que, si bien Android 4.1 o superior no se actualiza frecuentemente en todos los dispositivos móviles por igual, esto no se debe al desinterés de la empresa creadora, sino a que el distribuidor de hardware que incluye Android como sistema operativo siempre lo adapta a sus necesidades antes de lanzar la versión al mercado. También, en muchos casos, surgen nuevos dispositivos que se convierten en estrella dentro de una compañía, y por ello los priorizan, dejando de lado las actualizaciones de sus dispositivos más antiguos o con menos ventas.

Por suerte, Google Chrome pasó a ser, desde la versión de Android 4.1, el navegador web por defecto en esta plataforma, que a su vez se actualiza de forma independiente al sistema operativo. Esto nos permite contar con mejoras constantes en su motor de renderizado, el cual, hoy por hoy, tiene el papel de **núcleo del navegador web**.

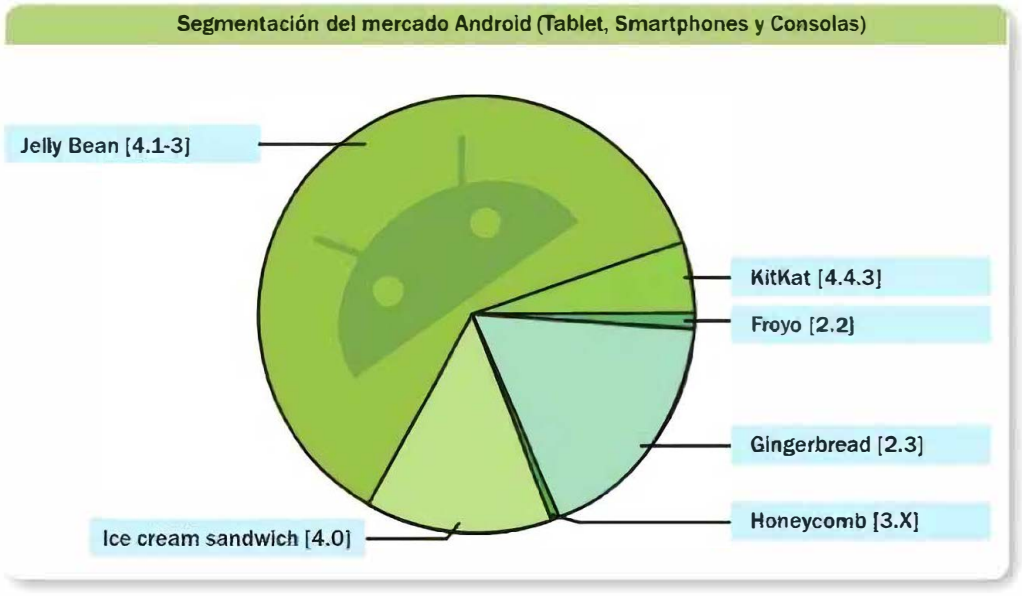


Figura 1. Este gráfico de abril de 2014 muestra la fragmentación de Android, donde se representa la fuerte adopción de los usuarios finales de dispositivos que poseen Android 4.0 o superior.

Chrome Mobile y las nuevas características

Google Chrome Mobile incorporó, a partir de la versión **31 beta**,

una nueva característica muy importante para los desarrolladores de aplicaciones web móviles, que permite configurar una WebApp en Android mediante un acceso directo en la pantalla principal del dispositivo, para poder ejecutarla en modalidad pantalla completa o **full-screen app mode**. Esta opción utiliza el motor de Chrome Mobile para Android y, además, toma la premisa de lo que ya aprendimos en el **Capítulo 8** de este libro: podemos agregar WebApps a la pantalla principal de **iOS** y que estas se ejecuten a pantalla completa.

Particularidades de la funcionalidad en Chrome Mobile

Como desarrolladores web, debemos tener en cuenta que Chrome Mobile, a partir de la versión **31.0**, adoptó una serie de funcionalidades propias que lo diferencian de lo que se puede realizar en iOS con **Safari Mobile** teniendo una WebApp funcionando a pantalla completa. Entre otras diferencias, podemos destacar que las aplicaciones web instaladas en la pantalla principal de Android trabajan exactamente igual que como lo harían al ejecutarse dentro del navegador web Chrome Mobile. Al utilizar el motor de renderizado **WebKit** de Chrome Mobile, se implantan las mismas políticas de seguridad **Sandbox** que este browser posee, y también se accederá a las mismas **APIs** que este navegador móvil brinda.

Por otra parte, en Google Chrome Mobile existe una diferencia visible con la misma característica que presta iOS: la WebApp que ejecutemos a pantalla completa en Android se visualizará en el listado de aplicaciones activas, **Task Switcher**, aunque no con su propio icono sino con el de Google Chrome. Asimismo, como nombre no figurará el título de la WebApp, sino que se podrá leer la leyenda

NOVEDADES DE LOS NAVEGADORES MÓVILES

Todos los navegadores web móviles disponen de información de las APIs y la metodología de desarrollo de sitios y aplicaciones web en las páginas principales de sus fabricantes. La información sobre Chrome para Android podemos encontrarla en la URL: **https://developer.chrome.com/devtools/index**.

"**Web App**", para Android en inglés, o "**Aplicación Web**", para la versión en español. Este indicador permitirá que el usuario sepa que está ejecutando una WebApp que consume datos de internet del dispositivo móvil.

Figura 2. En el cambiador de aplicaciones de Android podemos distinguir cuáles son nativas y cuáles son WebApps.

Al momento de escribir esta obra, Google Chrome no ha incorporado tampoco la capacidad de integrar una imagen del tipo **splash screen** a las WebApps que instalemos en la pantalla principal de Android. Por lo tanto, cuando ejecutemos una WebApp a pantalla completa, en el inicio de esta se mostrará una pantalla blanca hasta tanto se cargue la página principal de la web.

URL externas

Como medida de seguridad destacable por sobre la funcionalidad implementada en iOS, Android permite conocer cuándo una WebApp nos lleva a un sitio web externo: podemos visualizar, en el extremo superior de la pantalla, la URL hacia la cual nuestra WebApp a pantalla completa redirigió el navegador del usuario. Esto se implementó para notificar al usuario del teléfono móvil cuándo una WebApp carga una página web ajena a la URL original desde donde se instaló el acceso directo en la pantalla de Android.

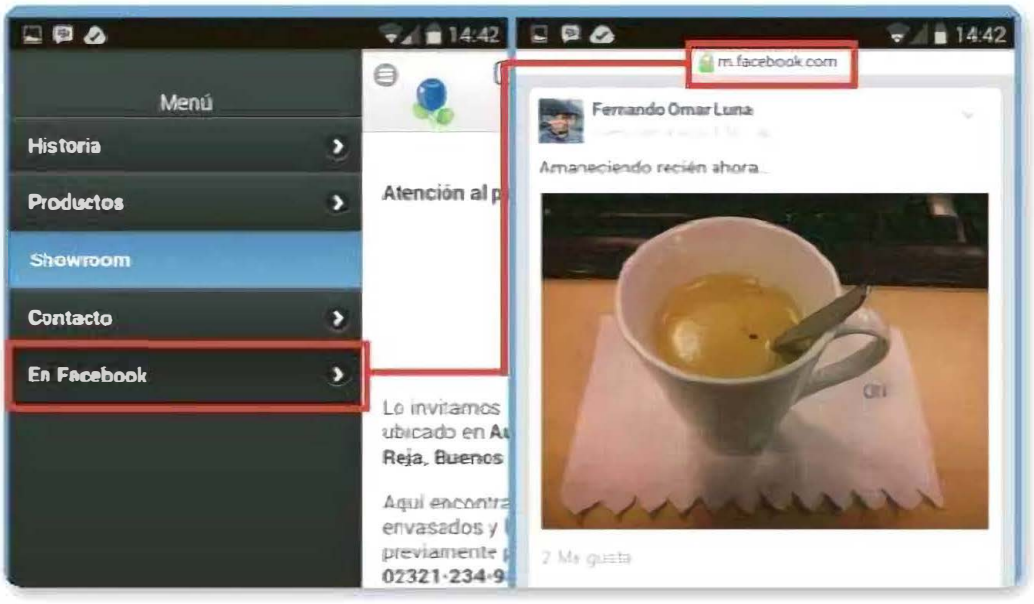


Figura 3. Al ingresar a una URL externa, desde una WebApp instalada en la pantalla principal de Android, el navegador Chrome nos visualizará la URL accedida en el extremo superior de la ventana.

Visualizar una WebApp como nativa en Android

A continuación, adaptaremos el ejercicio **User Book Store** realizado en el **Capítulo 6**, para poder instalarlo en la pantalla principal de los dispositivos Android. En este caso, incorporaremos el icono correspondiente para visualizar la WebApp en el escritorio de Android y modificaremos los **meta tags** para que la WebApp pueda cargarse a pantalla completa, al igual que lo que hemos realizado en el **Capítulo 8** para la plataforma iOS.

Crear los iconos para nuestra WebApp

Para incorporar un icono distintivo para nuestra WebApp, debemos crearlo en dos tamaños diferentes: **196 x 196 pixeles** y **128 x 128 pixeles**. Chrome para Android resolverá cuál es el más indicado para agregar a la pantalla del dispositivo donde se instalará el acceso a la WebApp. Si bien hay muchas modalidades de pantallas –dado

que Chrome para Android funciona a partir de la versión 4.0 de este sistema operativo–, las pantallas de estos equipos suelen disponer de una resolución alta. Es por esto que se dejan de lado todas las pantallas de baja resolución, como las que se pueden encontrar en equipos que corren la versión **2.x** de este sistema operativo.

Lo primero que debemos hacer, entonces, es crear un icono distintivo para nuestra WebApp de 196 x 196 pixeles. Puede tener la forma que nosotros deseemos, aunque Google siempre recomienda, para la creación de iconos para la plataforma Android, respetar las normativas y sugerencias que ellos realizan a través del sitio web **http://developer.android.com**. Si vamos a trabajar con el ejemplo realizado en este libro y no con uno propio, podemos descargar entonces, para agilizar nuestra tarea, los archivos de ejemplo desde el sitio de RedUsers Premium: **https://premium.redusers.com**.

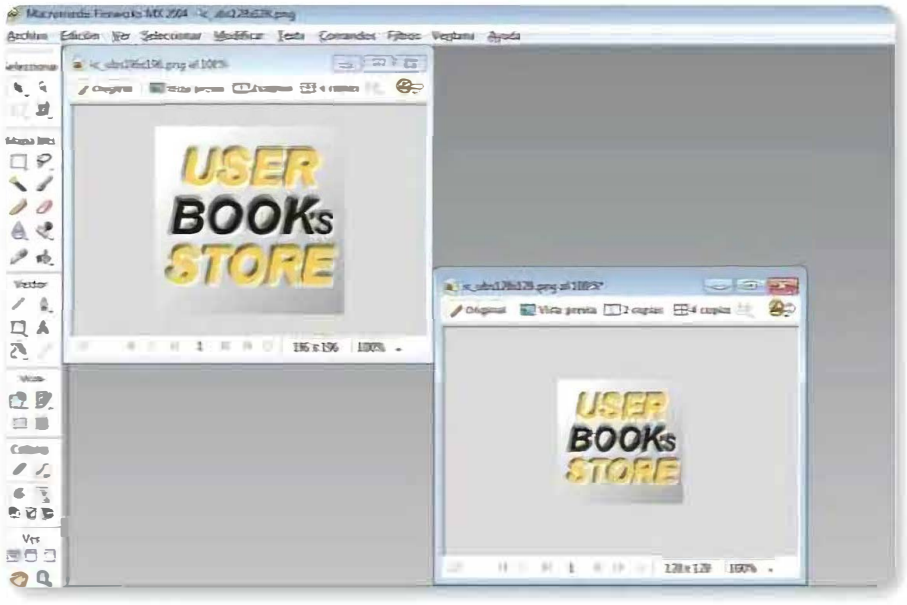


Figura 4. Los dos archivos .PNG diagramados que serán los iconos de nuestra WebApp instalable en Android.

Ya definido el icono, lo guardamos en la carpeta raíz del proyecto User Book Store con el nombre **ic_ubs196x196.png**. Luego guardamos el archivo con el nombre **ic_ubs128x128.png** y lo redimensionamos a 128 x 128 pixeles. De esta forma, ya tenemos definidos los dos iconos que incorporaremos a la WebApp en cuestión.

Ahora nos resta incluir el código específico en el archivo **home.php** de nuestro proyecto para que Chrome para Android pueda detectar

el icono correspondiente al instalarlo en la pantalla principal del dispositivo. Este código debemos agregarlo dentro del tag **<HEAD>**, igual que lo realizamos con nuestra WebApp para iOS en el capítulo anterior. El código es el siguiente:

```
...
<link rel="icon" sizes="196x196" href="ic_ubs196x196.png">
<link rel="icon" sizes="128x128" href="ic_ubs128x128.png">
</HEAD>
```

El título de la aplicación quedará definido por el tag **<TITLE>** del archivo **index.html** o **home.php** (en nuestro caso); por lo tanto, debemos tener cuidado de que este tag no sea modificado por ninguna funcionalidad programada dentro de la página. Por ejemplo, si desarrollamos una WebApp que funciona como un calendario, y en el tag **<TITLE>** muestra la fecha del día, cuando esta aplicación sea instalada en un equipo Android, quedará con el título de la fecha en la cual se instaló; por tal motivo, deberíamos evitar siempre poner una fecha en el tag **<TITLE>**.

Definir el meta tag <mobile-web-app-capable>

Ahora nos resta definir el tag que interpretará que nuestra WebApp puede instalarse en la pantalla principal de Android y correr en pantalla completa cuando sea ejecutada desde allí. Este código también debemos incorporarlo dentro del tag **<HEAD>**, justo debajo de la definición del meta tag **<TITLE>** y antes de los meta tags correspondiente a los iconos.

El código es el siguiente:

```
<HEAD>
...
   <meta name="viewport" content="width=device-width">
   <meta name="mobile-web-app-capable" content="yes">
...
```

Como podemos ver, el meta tag es similar al que escribimos para iOS, con la diferencia de que no incluye la sigla **apple-** al inicio. Ya con esto definido, podemos guardar la página **home.php** de nuestro proyecto, subirla al servidor web donde la ejecutamos y probar que todo funcione de la misma manera que antes. Dentro de este proyecto no debemos encontrar ninguna diferencia funcional con lo realizado hasta aquí.

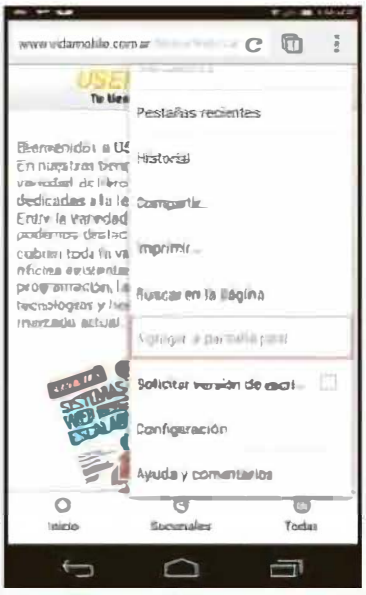
Instalar la WebApp en Android

Si todo funciona de manera óptima, ya estamos en condiciones de instalar la WebApp **User Book Store** en el sistema operativo Android. Para ello, debemos cargar la página principal en el navegador Google Chrome de nuestro dispositivo Android, y luego seguir las indicaciones del siguiente **Paso a paso**:

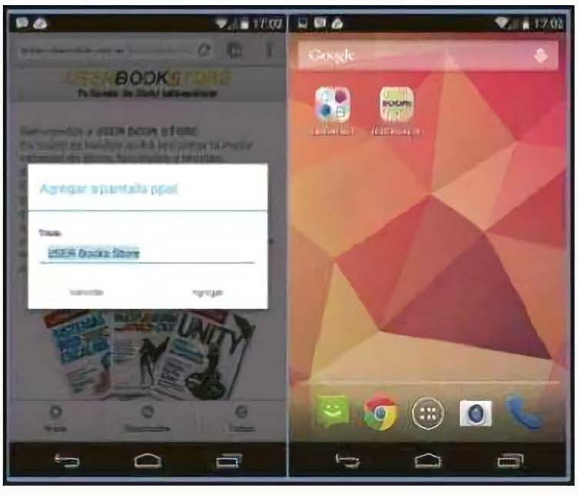
PAP: INSTALAR LA WEBAPP EN ANDROID

01 Con la WebApp preparada, ingrese a la URL correspondiente desde el navegador Google Chrome para Android. Una vez que cargó la página principal (`index.html` o `home.php`), ya puede iniciar la instalación del proyecto.

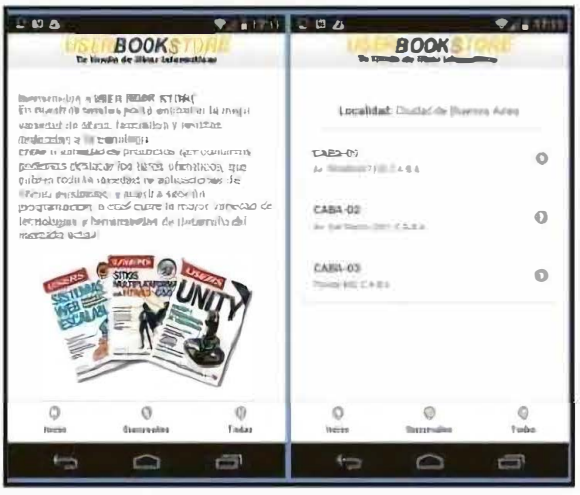
02 A continuación, despliegue el menú de Google Chrome, ubicado a la derecha de la barra de direcciones, y busque la opción `Agregar a pantalla ppal.`, `Agregar a pantalla principal` o `Add to home screen`, dependiendo de si tenemos un smartphone, una tablet en español o un dispositivo con Android en inglés.



03 Se desplegará una ventana que ofrecerá cambiar el título de la aplicación. Por defecto, traerá lo especificado en el tag `<title>` del encabezado de la página HTML del sitio web. Una vez aceptado o cambiado dicho título, presione `Agregar`.



04 Esta última acción agregará el acceso de nuestra WebApp a la pantalla principal de Android. Chrome definirá cuál es el mejor icono para visualizar dicho acceso directo. Si lo ejecuta, ya podrá utilizar la WebApp a pantalla completa, como si se tratase de una aplicación nativa de Android.



Como hemos visto, Android nos facilita la tarea de instalar WebApps en los dispositivos con sistema operativo Android 4.0 o superior. A su vez, como realizamos esta acción desde Google Chrome, nos aseguramos de que cualquier modificación que hagamos sobre la WebApp quedará actualizada de forma automática en el dispositivo de cada usuario que tenga el acceso directo a ella, sin necesidad de realizar ningún tipo de actualización de software. A su vez, nuestra WebApp siempre contará con las últimas especificaciones que se hayan incluido en Chrome, referentes a JavaScript, CSS o HTML5.

Agregar WebApps en BlackBerry

BlackBerry 10 ha marcado un cambio significativo dentro del terreno de los móviles basados en la clásica tecnología BlackBerry. Sus últimas versiones pusieron a esta plataforma a la par del resto de sus competidores en el mercado mobile. Y, de hecho, el navegador web incluido en su sistema operativo es uno de los que más soporte le da al ecosistema que rodea a HTML5.

Desde la versión 10 de BlackBerry, el browser de esta plataforma permite agregar WebApps a la pantalla principal, con ciertas restricciones, que repasaremos a continuación.

Iconos para BB10

Solo se soporta un tipo de icono que representa a la WebApp que se desea agregar a la pantalla principal de un dispositivo BB10. El icono debe respetar los **150 x 150 pixeles**, y para incorporar el meta tag correspondiente en una página web debemos hacerlo –al igual que en Android e iOS– a través de la sentencia **<link rel>**. Veamos un ejemplo:

```
<link rel="apple-touch-icon" href="ic_ubs150x150.png">
```

La primera impresión que podemos encontrar aquí es que BB10 utiliza el mismo atributo definido para el navegador Safari Mobile. La segunda diferencia es que no utiliza el atributo **sizes="xx"**, sino que directamente no se lo incluye dentro de la sentencia.

Agregar WebApp a pantalla de BB10

El navegador web de BlackBerry 10 o superior también dispone de la funcionalidad de **Agregar a pantalla principal**. Si incorporamos el icono de 150 x 150 pixeles al código de nuestro proyecto, podremos hacer que este se vea correctamente desde el escritorio de BlackBerry 10.

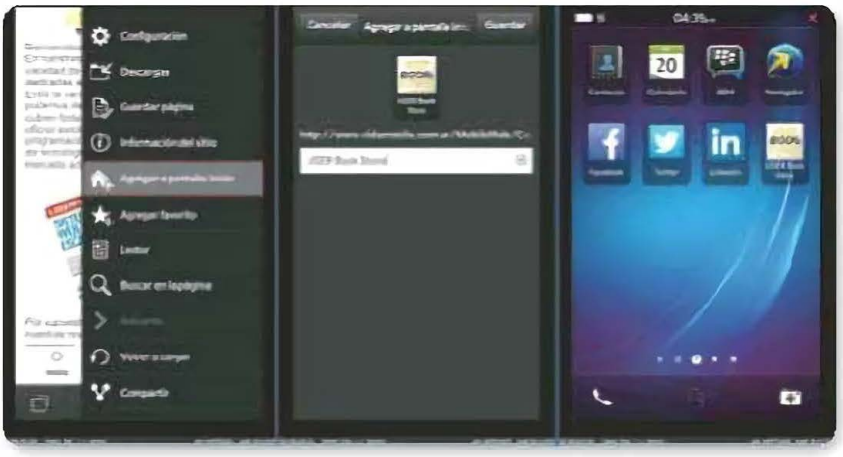


Figura 5. El archivo .PNG de 150 pixeles permitirá establecer el icono que deseemos para instalar nuestra WebApp en la plataforma BB10.

Trabajar a pantalla completa

BlackBerry 10 no incluye la funcionalidad **mobile-web-app-capable** por defecto, pero sí soporta la utilización de JavaScript, que permite llevar una WebApp a la modalidad de pantalla completa. Esto se puede realizar a través de la clase **navigator.standalone**, cuyo valor booleano deberá establecerse en **true**. Esta misma funcionalidad está soportada también por la plataforma iOS y por la casi desaparecida plataforma **Symbian**, muy popular dentro de la gama de teléfonos Nokia, previos a la llegada de Windows Phone.

Instalar WebApps en dispositivos Windows Phone

Windows Phone es el nuevo sistema operativo que, ya en su versión 8, ha demostrado ser un competidor serio para el resto de las plataformas. Actualmente, Windows Phone permite crear un acceso directo en el escritorio del sistema operativo móvil de cualquier web, aunque no reconoce los meta tags utilizados en las otras plataformas móviles. Por lo tanto, al ejecutar nuestra WebApp veremos indefectiblemente la barra de direcciones de Internet Explorer Mobile.

Al momento de escribir esta obra, Windows Phone no permite crear iconos personalizados para crear un acceso a la WebApp en su interfaz **Metro**; por lo tanto, al anclar a inicio nuestra WebApp, el sistema realizará una captura de pantalla de la página que estamos navegando en ese momento.

RESUMEN

Este capítulo nos permitió conocer la API para agregar a la pantalla de inicio de Android nuestras aplicaciones web móviles. Aprendimos a redimensionar los iconos de Android, adaptar el meta tag correspondiente y ver el comportamiento de nuestra WebApp una vez que se instaló en el escritorio de Android. También repasamos las opciones que presentan las plataformas BlackBerry 10 y Windows Phone a la hora de agregar una WebApp a sus pantallas de inicio.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Permite Android incorporar WebApps a su pantalla de inicio?
2. ¿Qué diferencias existen entre los **meta tags** de Android y los de iOS?
3. ¿Qué navegador se requiere para agregar una WebApp a la pantalla de inicio de Android?
4. ¿Cuáles son las dimensiones para los iconos que debemos crear para Android?
5. ¿Android identifica las imágenes del tipo **splash screen**?
6. ¿Qué diferencia encontramos entre una WebApp instalada en la pantalla de iOS y una WebApp instalada en la pantalla de Android?
7. ¿Dispone de meta tags propios el browser de BlackBerry para agregar una WebApp a su pantalla de inicio?
8. ¿Qué diferencia al navegador web de BlackBerry 10 del resto de los navegadores web móviles?
9. ¿Permite Windows Phone agregar WebApps a la pantalla de inicio desde su navegador Internet Explorer?
10. ¿Existen meta tags específicos para Windows Phone que permitan realizar esta última tarea?

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

*10

PhoneGap

En este capítulo, nos acercaremos a PhoneGap, conocido también como Apache Cordova, un framework mobile multiplataforma que nos permite convertir nuestro sitio web para móviles en una aplicación instalable en teléfonos inteligentes y tablets. También haremos un repaso por las principales características de esta librería que nos permite fácilmente aprovechar diversos recursos de hardware de los dispositivos móviles.

Introducción

PhoneGap nació con la premisa de permitir el desarrollo de aplicaciones móviles para múltiples plataformas partiendo desde un único desarrollo englobado por las tecnologías HTML5, CSS y JavaScript. La propagación de la tecnología en materia móvil hizo que un programador o empresa deba adaptarse a las necesidades de su cliente, que –globalización mediante– puede estar situado en cualquier rincón del mundo.

Sumada a esta variable, la tendencia de uso de dispositivos móviles en el rincón donde este cliente se encuentre puede ser completamente distinta a la tendencia marcada en el país donde reside el programador; por lo tanto, este último tendrá que contar con los conocimientos necesarios que lo habiliten para poder desarrollar una aplicación móvil, ya sea para Android, como para iOS, Windows Phone o BlackBerry, entre otras plataformas populares.

PhoneGap busca, desde las bases del desarrollo web, acortar lo más posible las diferencias que existen entre las distintas plataformas diseminadas por los mercados de todo el mundo. A través de una librería creada en JavaScript, propone acceder a las características principales de cada equipo móvil, como ser el GPS, acelerómetro, cámara o almacenamiento local, entre otras funcionalidades.

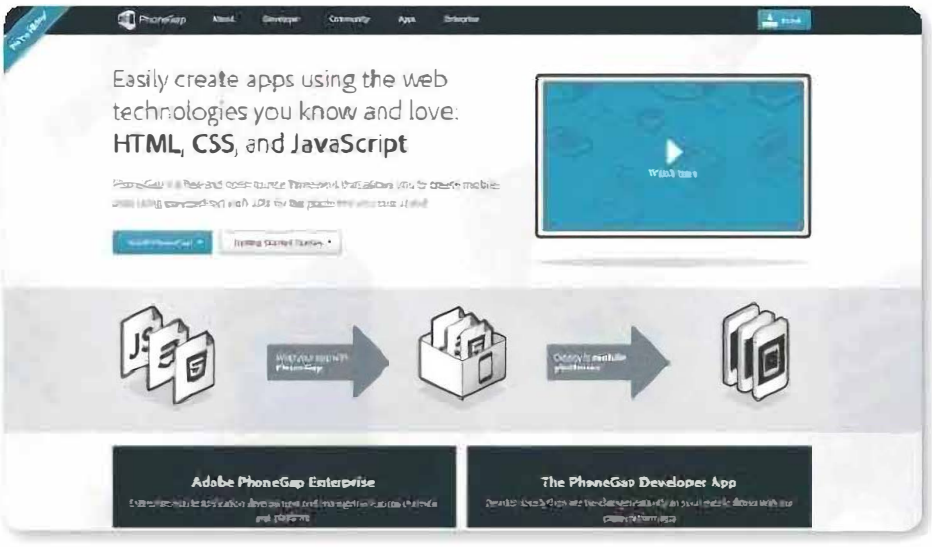


Figura 1. Desde el sitio web **www.phonegap.com** podemos acceder a toda la información referente a este framework multiplataforma que nos permite crear una app móvil desde el código fuente de una WebApp.

Así, con simples conocimientos web, podremos crear en tiempo récord aplicaciones móviles sin la necesidad de tener que empaparnos de conocimiento de cada una de las plataformas existentes en el bolsillo de los usuarios.

Esto nos ayuda a acortar la curva de aprendizaje y, a su vez, a conseguir, mediante un único esfuerzo, abarcar una integración en múltiples plataformas con una interfaz unificada.

Arquitectura

PhoneGap aprovecha la arquitectura implementada en el navegador web de los teléfonos móviles para crear una aplicación que se visualiza en un contenedor web (más conocido como "el motor del browser"), se ejecuta a pantalla completa y explota las características locales del sistema operativo desde su **API JavaScript**.

> PHONEGAP CREA UNA APLICACIÓN QUE EXPLOTA LAS CARACTERÍSTICAS LOCALES DEL S.O.

La interfaz de nuestra aplicación móvil es creada por completo en HTML, CSS y frameworks como jQuery Mobile y JavaScript, y utiliza el contenedor PhoneGap para poder hacer correctamente el despliegue de nuestra aplicación en los diferentes sistemas operativos.

Cuando creamos una aplicación PhoneGap, comúnmente llamada **compilación**, esta se transforma en un archivo binario que podrá ser distribuido en las tiendas de aplicaciones más conocidas (**Google Play**, **iTunes**, **Amazon Market** y **Windows Phone Marketplace**, entre otros), respetando, por supuesto, los requisitos que cada tienda dicta.

Internamente, los archivos web que componen nuestra app de PhoneGap se encuentran almacenados de manera local y se cargan en el contenedor web. Así, PhoneGap nos permite desarrollar una app desde los archivos base de una WebApp, utilizando las múltiples páginas HTML que hayamos creado. Sin embargo, debemos tener cuidado cuando desarrollamos una app PhoneGap de múltiples páginas con las variables creadas en JavaScript, ya que estos valores pueden perderse entre la invocación de una página y otra. En estos casos, debemos contemplar la opción de almacenar los valores de variables de forma local, como vimos en el **Capítulo 7** de esta obra, para poder, así, mantener la consistencia de una WebApp que tiene cierta complejidad en su función.

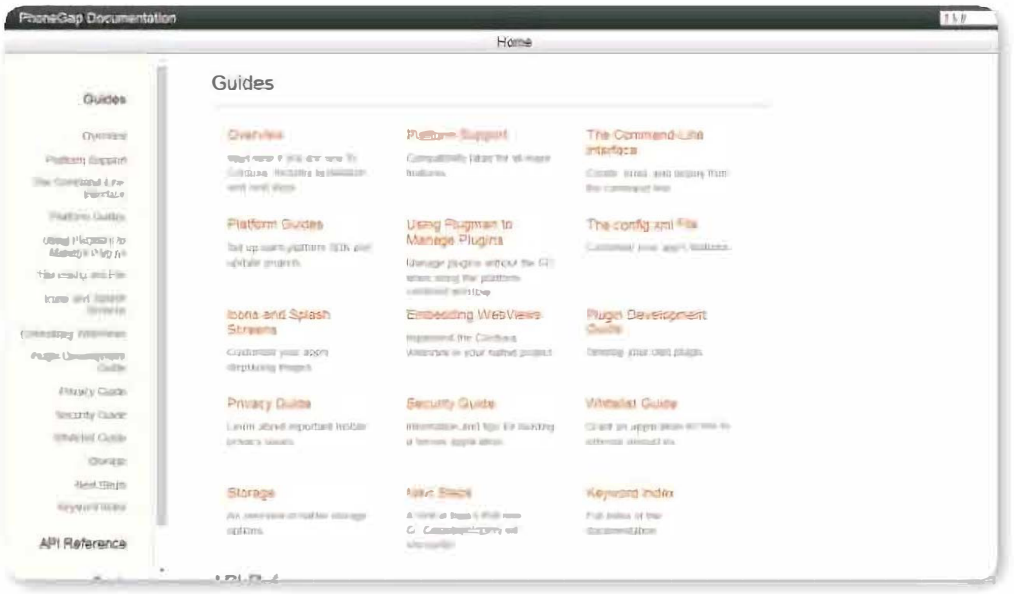


Figura 2. Desde el sitio web **http://docs.phonegap.com** podremos acceder a toda la documentación de este framework, para conocer en detalle las opciones que nos brinda.

Dreamweaver y PhoneGap

PhoneGap nació como un proyecto independiente que permitía encapsular sitios web en el motor **WebKit** del navegador Safari, pensado originalmente para la plataforma iOS. En base a las buenas críticas que recibió el equipo de desarrollo por este sensacional framework, con el tiempo fue mejorado y trasladado a las diferentes plataformas que actualmente cubre.

Algunos años más tarde –y tal vez porque **Adobe Flash**, que era el caballo de batallas, quedó relegado de las plataformas móviles–, la firma Adobe decidió adquirir PhoneGap e integrarlo en su producto **Dreamweaver**, que hoy por hoy sigue siendo la aplicación más utilizada para la construcción de sitios web en diferentes lenguajes.

EL FRAMEWORK PHONEGAP

PhoneGap no solo nos permite compilar una WebApp hacia las plataformas móviles; también nos brinda una serie de herramientas, mediante la API, que utilizando JavaScript nos permitirán explotar las características principales del hardware de los dispositivos móviles.

Asi fue como, desde la versión 5.5 de esta herramienta, Dreamweaver comenzó a dar soporte para PhoneGap. Aun así, **Nitobi**, el creador de este maravilloso framework, puso como condición que PhoneGap fuera de código libre: por esta razón, se desprendió una versión libre bajo licencia Apache, comúnmente conocida como **Apache Cordova**.

Adobe se quedó con el derecho de PhoneGap y PhoneGap Build; Apache Cordova, por otro lado, es distribuido de forma gratuita entre quienes quieren desarrollar aplicaciones móviles híbridas, manteniendo la filosofía original con que Nitobi creó PhoneGap.

Cómo transformar una WebApp en híbrida

A continuación, repasaremos los conceptos básicos que necesitamos conocer para poder adaptar rápidamente una WebApp desarrollada para que pueda funcionar como una app instalable en nuestros dispositivos móviles. Utilizaremos las bases de PhoneGap Build, lo cual nos evita tener que contar con una versión de Dreamweaver en nuestro equipo.

Lo primero que debemos hacer es obtener una cuenta en PhoneGap Build. Para ello, nos dirigimos a **https://build.PhoneGap.com** y seleccionamos **Register**. Esta opción nos permitirá elegir entre los diferentes planes para registrarnos en el servicio de PhoneGap Build.

El primero de ellos es gratuito y nos habilita a poder crear solo una aplicación privada. El resto de las aplicaciones que creemos estará disponible para cualquiera que pueda navegar por ella y descargar el código fuente. Si deseamos que esto último no ocurra, debemos elegir algún plan pago.

APACHE CORDOVA

La firma Nitobi, creadora del framework PhoneGap, vendió en 2012 la plataforma PhoneGap y PhoneGap Build a Adobe, con la condición de poder mantener una versión del framework bajo licencia Apache (open source) destinada a la comunidad que colaboró en el desarrollo del framework.

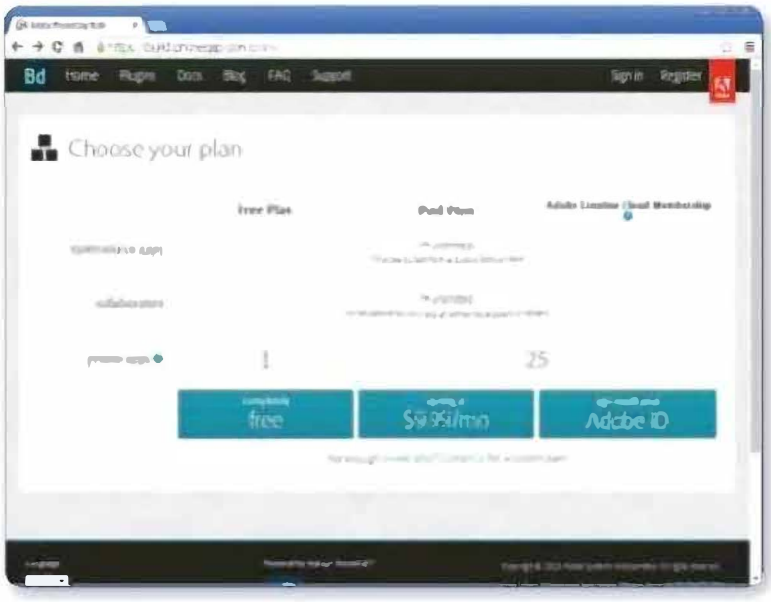


Figura 3. Desde **https://build.PhoneGap.com/plans** podemos crear nuestra cuenta o utilizar una cuenta existente de GitHub o Adobe ID y elegir el plan que más nos convenga.

Si ya disponemos de una cuenta de servicios pertenecientes a Adobe, como, por ejemplo, **Adobe ID** o **GitHub ID**, también podremos utilizarlas para identificarnos en el servicio de PhoneGap Build y así evitar tener que crear una cuenta desde cero.

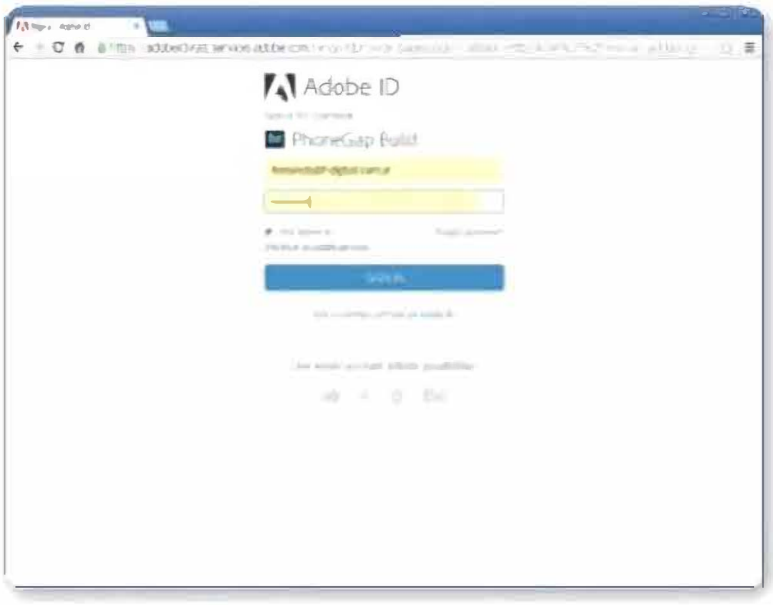


Figura 4. Ya creada la cuenta y validada mediante un e-mail enviado a nuestra casilla de usuario, podemos iniciar sesión en el servicio y comenzar a utilizar PhoneGap Build.

Una vez que ingresamos al servicio de PhoneGap Build, veremos allí, en forma de listado, todas las aplicaciones PhoneGap que hemos ido creando. Para cada una de las aplicaciones que creamos, obtendremos un app ID que las identifica dentro del servicio de PhoneGap Build. Además, accederemos a otros datos, como los del propietario de la app, donde figurará una dirección de correo electrónico correspondiente al usuario que la subió, la versión de PhoneGap utilizada para compilarla y, a través de una serie de iconos, podremos ver también para qué plataformas fue compilada exitosamente nuestra app.

> AL INGRESAR A PHONEGAP BUILD VEREMOS TODAS LAS APLICACIONES QUE HEMOS IDO CREANDO

Las plataformas para las cuales se compiló de manera exitosa se verán en **azul**, mientras que en **rojo** se verán las plataformas para las cuales no se haya podido realizar esta acción.

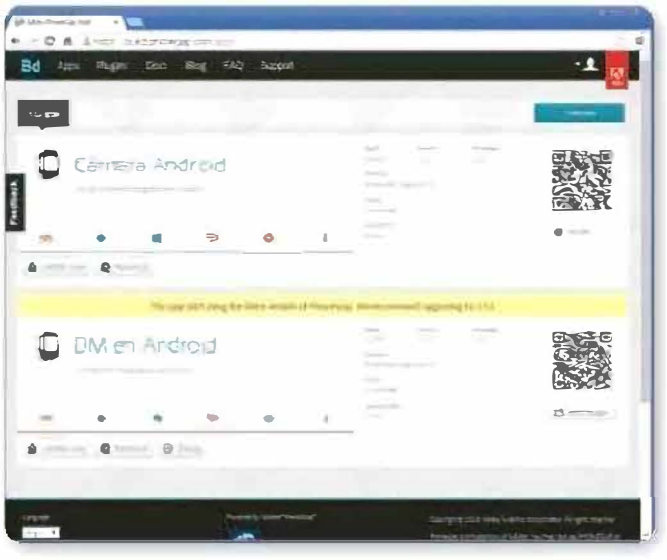


Figura 5. Desde nuestro perfil de PhoneGap Build podemos administrar las apps creadas con este framework, agregar nuevos proyectos y recompilar los existentes si estos requirieran un cambio.

Esta diferencia radica en que, en algunas plataformas, se requiere adicionar archivos que nos identifiquen a nosotros como desarrolladores certificados. Por ejemplo, en iOS necesitamos registrarnos como desarrolladores, obtener un **Apple Developer ID** y un conjunto de archivos que se suman a nuestra app para poder certificar que esta le corresponde a un desarrollador certificado.

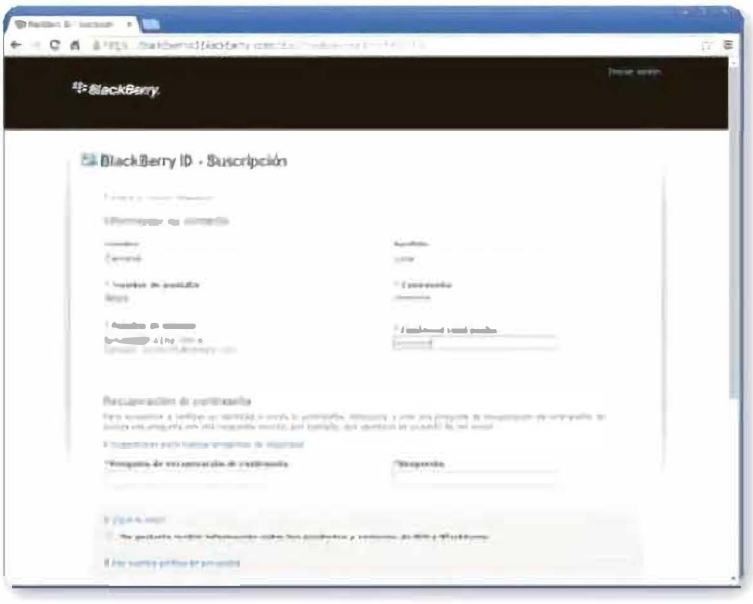


Figura 6. Todos los portales que permiten la creación y distribución de apps para dispositivos móviles poseen un apartado para que los desarrolladores creen su cuenta de usuario.

De no contar con esto, nunca podremos distribuir apps dentro de la plataforma iTunes. En el caso de Android, Google nos permite compilar aplicaciones sin registrarnos como desarrolladores y probarlas en su plataforma; en ese caso, es el usuario quien asume el riesgo de lo que pueda llegar a pasar al momento de instalar y ejecutar una app que no ha sido validada por Google y que, por supuesto, no está disponible en su tienda de aplicaciones Google Play. Por último, BlackBerry también solicita el registro de desarrolladores, al igual que Microsoft para su plataforma Windows Phone.

Ya estamos en condiciones de compilar una de las WebApps que desarrollamos a lo largo de este libro en PhoneGap, para poder distribuirla en los equipos móviles que deseemos. La prueba que mostraremos a continuación podemos realizarla tanto en un dispositivo físico como en un emulador. Tomaremos el código fuente de la aplicación **DOCTOR ASSISTANCE**, desarrollada en el **Capítulo 5** de este libro.

Modificación de index.html

Al código de la aplicación sólo debemos realizarle una simple modificación en su archivo principal, **index.html**. Para esto, lo editamos

en nuestro editor web favorito y, en el **<header>** de dicho archivo, agregamos la siguiente línea de código:

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

Esto habilitará a PhoneGap Build, cuando procese el código fuente de nuestro proyecto, a detectar que este debe compilarse bajo esta plataforma. No importa si agregamos o no el archivo **.JS** correspondiente a PhoneGap.

Al momento de compilar para cada una de las plataformas existentes, PhoneGap Build ubicará el archivo correspondiente a estas plataformas antes de realizar la compilación de nuestro proyecto.

> PHONEGAP PROVEE EL ARCHIVO PHONEGAP.JS PARA CADA SISTEMA OPERATIVO MÓVIL

Actualmente PhoneGap dispone de un archivo **PhoneGap.js** para cada sistema operativo móvil para el cual permite compilar una aplicación. Esto lo hará la plataforma de manera transparente para el desarrollador, evitando así varios dolores de cabeza, como también la necesidad de subir una copia de nuestro código fuente por cada plataforma a la que deseamos compilar nuestro proyecto.

Veamos, a continuación, cómo queda el código fuente de la página principal:

```
<!DOCTYPE html>
<html>
<head>
   <title>Doctor Assistance</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.
mobile-1.3.2.min.css" />
   <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
   <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.
js"></script>
   <script src="Phonegap.js" ></script>
</script>
</head>
...
```

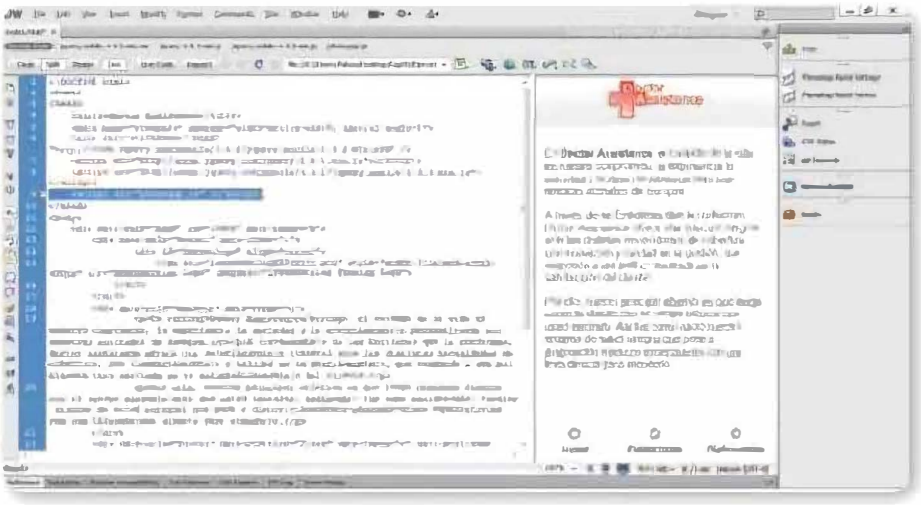


Figura 7. Con una simple línea en el archivo **`index.html`** de nuestra WebApp ya podremos subirla al portal PhoneGap Build para transformarla en una app para las diferentes plataformas móviles.

Creación del icono de nuestra app

La siguiente tarea que debemos realizar es crear el icono que compondrá nuestra aplicación móvil. El icono debe basarse en el **isologo** de nuestra WebApp. Descargando el código fuente que corresponde a este proyecto desde **https://premium.redusers.com**, encontraremos un icono ya creado en la carpeta raíz. Este icono debe mantener una medida estándar de **57x57 pixeles**, exigida por PhoneGap Build.

Figura 8. Desde nuestro editor de imágenes favorito podremos crear rápidamente el icono de nuestra aplicación, para luego subirlo a PhoneGap Build y empaquetar nuestro proyecto como app.

Empaquetar nuestro proyecto

A continuación, debemos crear un archivo **.ZIP** que subiremos a los servidores de PhoneGap Build. Este deberá contener todos los archivos y la estructura de carpetas que le corresponden. Si, por ejemplo, en nuestro proyecto tenemos definido el uso de archivos remotos jQuery Mobile, estos podrán ser reemplazados por las subcarpetas que contengan en forma local los archivos. Esto evitará que con cada inicio de nuestra aplicación se deba ir a buscar el contenido remoto de jQuery Mobile. Como beneficio, se optimizará nuestra app y no se utilizará un consumo de datos de internet en el dispositivo del cliente. Como desventaja, podemos indicar que el peso en kilobytes de nuestra app dependerá de cuánto ocupen los archivos y dependencias de jQuery Mobile.

Una vez que ya tenemos modificado el archivo **index.html** de nuestro proyecto, creado el archivo **.ZIP** y definido el icono de nuestra app, podemos dar de alta este proyecto en PhoneGap Build. Para ello, tenemos que ir a la página principal de PhoneGap Build y presionar el botón **+new app**. Enseguida se abrirá el apartado para la creación de una nueva app, en el cual encontraremos diferentes maneras de incluir los archivos de dependencia a nuestro perfil de PhoneGap.

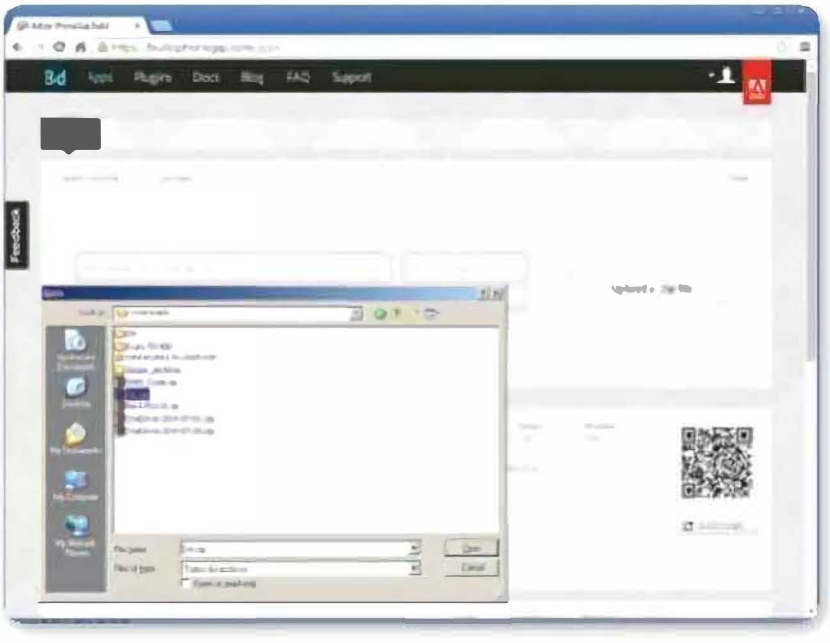


Figura 9. En el proceso de creación de un nuevo proyecto PhoneGap Build, podremos seleccionar desde nuestra computadora el archivo Zip que agrupe la estructura de nuestra aplicación.

Una de las opciones es poner la ruta desde nuestro perfil de GitHub; la otra, subir un archivo Zip desde nuestra computadora. Aquí utilizaremos, a modo de ejemplo, esta última opción. Para comenzar, presionamos el botón **Upload ZIP file**, y, desde la ventana de diálogo que se abre en nuestro equipo, ubicamos el archivo Zip, lo seleccionamos y presionamos el botón **Abrir**. De esta forma, será cargado a nuestro perfil de PhoneGap.

Ya subido el archivo al servidor de PhoneGap Build, presionamos sobre el título de la nueva aplicación creada para ingresar al detalle de esta, en donde podremos especificar la información adicional pertinente. Lo primero que haremos es ubicar el apartado **Configuration**, en donde ingresaremos, en el campo **app title**, el nombre de nuestra app, **Doctor Assistance**. Luego seleccionamos el icono de esta, presionando el botón **Seleccionar archivo** y dirigiéndonos hasta la carpeta que contiene dicho archivo en formato **PNG**.

Figura 10. El icono creado para identificar nuestra app también puede ser subido desde el portal PhoneGap Build en el momento de crear el proyecto.

CONFIGURACIÓN DETALLADA DEL PROYECTO

PhoneGap dispone de un archivo denominado **config.xml**, en donde se podrán configurar, de manera detallada, las pantallas de inicio, iconos y servicios del móvil que nuestra app utilizará. El archivo puede sumarse en la carpeta raíz de nuestro proyecto HTML, o podemos dejar que PhoneGap lo genere automáticamente.

En el campo **Package** debemos declarar un nombre descriptivo para que lleve el proceso que ejecuta el sistema operativo móvil, con el fin de identificar nuestra app. En nuestro caso, debemos ingresar un nombre que identifique a nuestra app y el nombre de nuestra compañía: **com.doctorassistance.vidamobile.**

En el campo **Version**, incluimos la versión de nuestro proyecto. Si es la primera de ellas, solo debemos ingresar **1.0**. A medida que agreguemos funcionalidades o mejoras, iremos incrementando el dígito verificador para las actualizaciones, o el dígito principal para una nueva versión.

Luego nos queda seleccionar la versión de PhoneGap con la cual compilaremos nuestro proyecto y agregar, en el campo **Description**, un comentario que explique de forma clara el objetivo de nuestra app.

Con estos datos ya completados, solo nos queda presionar el botón **Save** para guardar nuestra nueva app en el perfil de PhoneGap Build y ya poder recompilar y descargar los archivos binarios de cada plataforma. PhoneGap Build nos avisará, mediante un mensaje de alerta en la barra superior del sitio web, que la app agregada ya se encuentra en cola de compilación para las distintas plataformas.

Finalizado este proceso, encontraremos en cada una de las plataformas un botón **azul** que identifica que el archivo **binario** de nuestra app ya se encuentra listo para ser descargado, o un botón **rojo** que indica que falta información para poder compilar nuestra aplicación para esa plataforma.

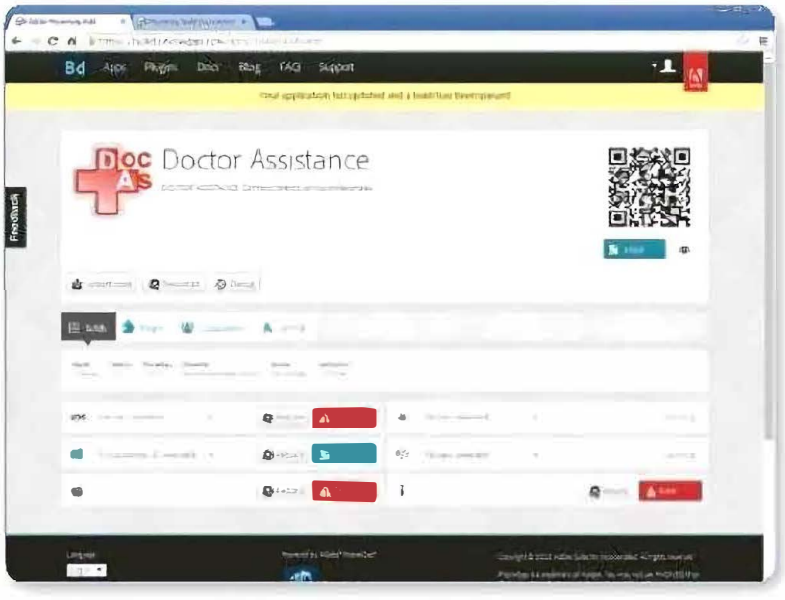


Figura 11. Ya creado nuestro proyecto, al presionar **Save**, PhoneGap Build comenzará el proceso de compilación hacia las distintas plataformas móviles.

De las plataformas disponibles para compilar nuestra app en PhoneGap Build, debemos destacar que solo Windows Phone y Android permiten compilar nuestra app sin los certificados de desarrollador correspondientes. Si bien dicha app no se podrá subir a la tienda de aplicaciones de la plataforma, al menos podremos testearla en un simulador o dispositivo físico en modo **debug**.

Incluir certificados de desarrollador

Para incluir certificados de desarrollador de aplicaciones para las diferentes plataformas, debemos suscribirnos a estas de acuerdo a lo que cada fabricante de sistemas operativos móviles indica. Ser desarrollador de determinadas plataformas como iOS, Android y Windows Phone tiene un costo anual.

Figura 12. Desde el perfil de las distintas plataformas podemos incorporar los correspondientes certificados obtenidos desde el portal de desarrollador de cada fabricante.

Las plataformas BlackBerry y Nokia (esta última, solo para la línea **Symbian** o **S40**) no requieren un pago anual, sino que poseen suscripciones gratuitas. A continuación, en la tabla, detallamos el portal de suscripción para cada comunidad de desarrolladores de aplicaciones móviles:

PORTALES PARA SUSCRIPCIÓN DE DESARROLLADORES

PLATAFORMA	SITIO WEB DE SUSCRIPCIÓN
iOS	https://developer.apple.com/programs/ios
Android	http://developer.android.com/intl/es/index.html
Windows 8 / Windows Phone	http://dev.windows.com/en-us/join
BlackBerry	https://developer.blackberry.com
Nokia (plataformas diferentes a WPhone)	http://developer.nokia.com/community/wiki/Publish_and_Subscribe

Tabla 1. Detalle de los sitios web que nos permiten suscribirnos como desarrolladores móviles de las distintas plataformas.

Cómo testear nuestra app compilada

Por último, nos queda testear nuestra aplicación ya compilada para las diferentes plataformas. Para ello, podremos proceder de dos formas distintas. Una opción es descargar el archivo binario a nuestra computadora y luego subirlo al dispositivo o simulador. La otra opción es aprovechar, desde un dispositivo físico, con una app que lea códigos QR, la lectura del código correspondiente ubicado a la derecha de nuestro proyecto en PhoneGap Build.

Veamos a continuación un ejemplo de cómo instalar nuestra app compilada en PhoneGap Build desde un dispositivo Android, utilizando una app de **código QR**. Primero abriremos la aplicación lectora de códigos QR. Si no disponemos de esta aplicación, podemos descargar, desde la tienda Google Play, **QR Droid**.

CERTIFICADOS DE DESARROLLADOR

Los distintos vendedores de las plataformas móviles implementan un certificado de desarrollador personalizado según la plataforma. En Nokia e iOS, por ejemplo, debemos gestionar un ID único para cada app que creamos, mientras que Microsoft sólo valida a través de un **Developer ID** único.

Una vez iniciado QR Droid (o nuestro lector de códigos QR favorito), deberemos enfocar el correspondiente código QR visualizado en nuestro perfil de PhoneGap Build hasta que este pueda ser leído por el programa. A continuación, la aplicación lectora de códigos QR abrirá el navegador web por defecto instalado en el teléfono móvil o tablet e iniciará la descarga del archivo **APK** correspondiente a nuestra app que deseamos instalar.

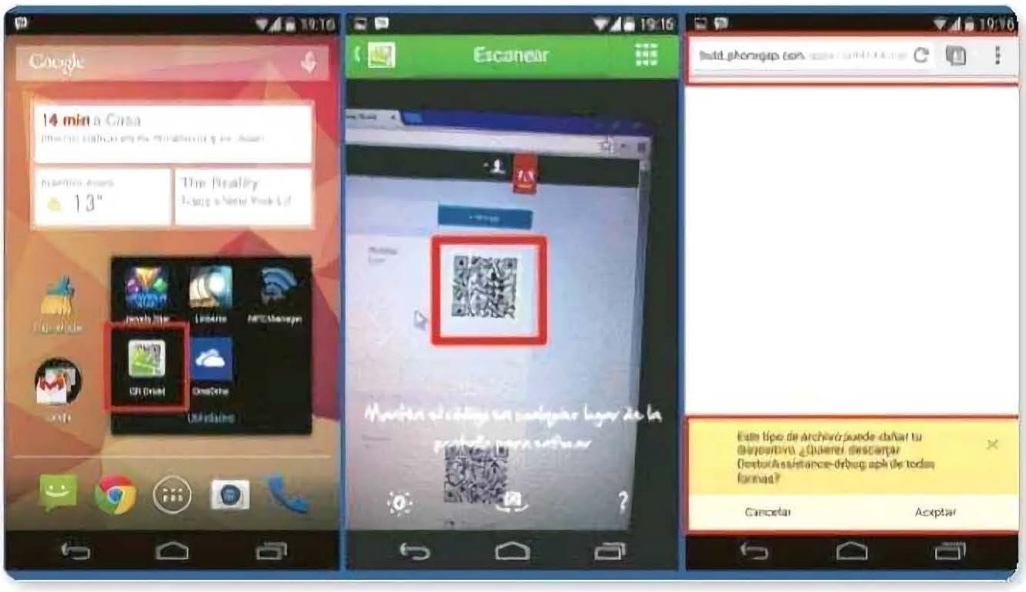


Figura 13. A través de cualquier aplicación lectora de códigos QR podemos iniciar la descarga e instalación de nuestro proyecto compilado desde PhoneGap Build.

SI EL NAVEGADOR NOS PREGUNTA SI QUEREMOS DESCARGAR EL ARCHIVO APK, ACEPTAMOS

Es probable que el navegador web del dispositivo móvil nos pregunte si deseamos descargar el archivo APK a nuestro dispositivo, por cuestiones de seguridad. Simplemente debemos aceptar dicha descarga. Luego, desde la barra de tareas de Android, seleccionaremos el archivo descargado al dispositivo para poder iniciar la instalación de nuestra app en modo **debug**.

A continuación, visualizaremos los pasos de instalación de cualquier app que descargamos en Android: debemos aceptar las condiciones y proceder con la instalación de nuestra app. Al finalizar dicha instalación, el proceso

nos ofrecerá abrir la aplicación. Presionamos el correspondiente botón, y listo: ya podremos verificar el funcionamiento de nuestra primera WebApp convertida a aplicación mediante PhoneGap Build.

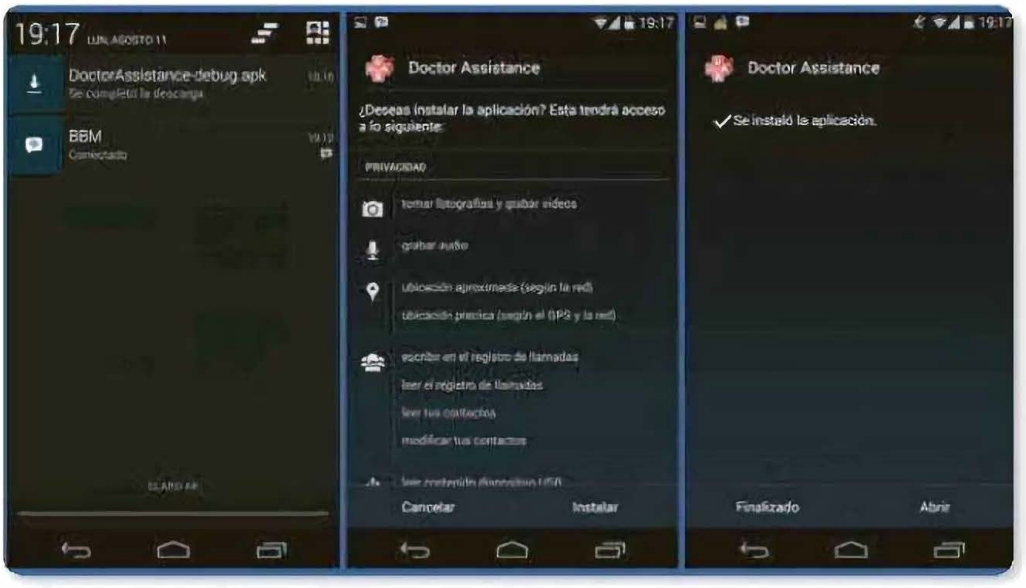


Figura 14. Luego de pasar por el proceso de instalación, nuestra app podrá ser testeada en el dispositivo móvil.

RESUMEN

PhoneGap/Apache Cordova nos permite rápidamente transformar un sitio web en una app para dispositivos móviles, con muy poco esfuerzo. Combinando este framework con el servicio de PhoneGap Build lograremos, en tiempo récord, portar cualquier WebApp hacia el mundo de las aplicaciones instalables en Android, iOS, BlackBerry, Windows Phone o los dispositivos Nokia basados en Symbian y S40.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es **PhoneGap**?
2. ¿Qué diferencia hay entre PhoneGap y **Apache Cordova**?
3. ¿Qué diferencia hay entre PhoneGap y **PhoneGap Build**?
4. ¿Podemos utilizar PhoneGap sólo para compilar WebApps a apps híbridas?
5. Investigue, en el primer capítulo de este libro, la diferencia entre app nativa y app híbrida.
6. ¿Cómo trabaja una app híbrida?
7. ¿Hacia qué plataformas podemos compilar en PhoneGap Build?
8. ¿Para qué sirven los certificados de desarrollador?
9. ¿De qué forma podemos instalar una app compilada en PhoneGap Build?
10. ¿Qué plataformas compiladas en PhoneGap Build nos permiten testear las aplicaciones sin el certificado de desarrolladores?

PROFESOR EN LÍNEA

Si tiene alguna consulta técnica relacionada con el contenido, puede contactarse con nuestros expertos: profesor@redusers.com

BB10, Windows Phone y emuladores web

Este capítulo nos brindará un panorama sobre las herramientas para el desarrollo nativo de aplicaciones móviles para las plataformas BlackBerry 10 y Windows Phone. Conoceremos cuáles son los SDK de cada uno de estos sistemas operativos móviles, qué requisitos debemos cumplir en estos terrenos y cómo podremos probar nuestros desarrollos a través de los simuladores móviles que cada plataforma pone a nuestra disposición.

Programación nativa para BlackBerry 10

Si bien iPhone es el producto más destacado en el terreno mobile y fue el que cambió rotundamente la manera de utilizar un celular, debemos reconocer que gran parte del mérito en los avances de navegadores y sistemas de mensajería avanzados, dentro del terreno móvil, lo tiene, desde mucho tiempo antes, la plataforma BlackBerry.

Esta plataforma realizó cambios significativos en el terreno móvil, dado que adoptó las bondades que la empresa **Palm** había puesto en el mercado de la telefonía móvil y la computación personal portátil desde fines de la década del 90. Así fue como BlackBerry condecoró esta plataforma con un sistema de navegación por web y actualización de las noticias presentadas en nuestros sitios o blogs más reconocidos mediante el sistema **RSS**.

BlackBerry también inició, en parte, el terreno de la mensajería instantánea móvil desde su plataforma BBM (**BlackBerry Messenger**). Incorporó un sistema de chat similar al que nos presentó **ICQ** en los inicios de la Web de escritorio, que evitaba que tuviéramos que darles nuestro número de teléfono a los interlocutores con los que chateábamos.

El sistema de mensajería BBM es, hasta hoy, muy querido y deseado por miles de usuarios en todo el mundo. Por esto, BlackBerry decidió, a partir de 2014, llevar su sistema BBM a las plataformas iOS y Android, a pesar de que los laureles en el terreno los tenga ahora **WhatsApp**.

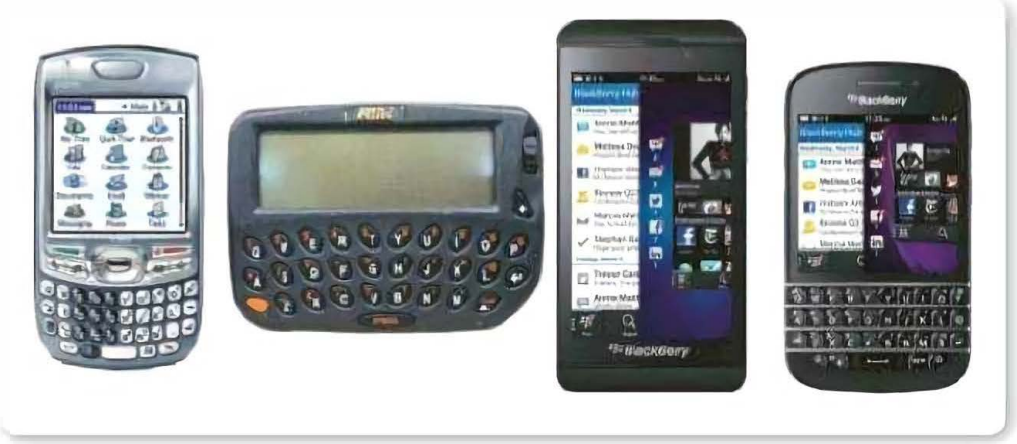


Figura 1. Una de las últimas versiones de teléfonos **Palm**, junto con el primer **RIM** (solo localizador) y la nueva generación de esta plataforma.

La generación BB10

> CON BLACKBERRY 10, BB PROPUSO UN NUEVO S.O. CON MULTITAREA EN TIEMPO REAL

La llegada de BlackBerry 10 al mercado mundial le dio un nuevo respiro a una plataforma que pasó de ser líder exclusivo en el terreno móvil a casi desaparecer por completo de este. **BlackBerry 10** retornó con artillería pesada. Propuso un nuevo sistema operativo con multitarea en tiempo real y lanzó teléfonos inteligentes que respetan las características clásicas de los modelos **QWERTY**: **Q10** y **Q5**, y también otros modelos táctiles, **Z10** y **Z30**, que pelean palmo a palmo con iOS y Android, entre otros competidores. Y, a la par de estos nuevos modelos en el mercado, BlackBerry lanzó también un nuevo conjunto de herramientas para desarrolladores de esta plataforma.

Herramientas de desarrollo para BB10

Dentro de las herramientas de desarrollo de aplicaciones para la plataforma BlackBerry, tenemos la posibilidad de seleccionar el **IDE** y **SDK** para la vieja generación de equipos, para la nueva generación BB10 y para BlackBerry PlayBook. A su vez, el mercado se segmenta en el desarrollo de aplicaciones nativas, el desarrollo de aplicaciones basadas en HTML5 y aplicaciones creadas con **Adobe Air**. Además, en este último tiempo, BB10 incorporó también un **runtime** que le permite correr **aplicaciones Android** dentro de su plataforma.

Desarrollo nativo

Las herramientas de desarrollo nativo permiten crear apps para BB10 de dos maneras: directamente, utilizando código C++, o combinando

DESARROLLO PARA BLACKBERRY

En estos últimos meses, BlackBerry decidió complementar el desarrollo nativo, basado en **WebWorks**, directamente con el compilador **Apache Cordova**, porque este último resuelve casi todo el acceso al hardware que los equipos BB exigen.

este código con el lenguaje QML y JavaScript, aprovechando estos dos para la construcción UI de la app y dejando el código C para el acceso al hardware y otras funcionalidades más complejas.

El IDE en primer plano

Momentics IDE es el ambiente de desarrollo que propone BlackBerry para la codificación de apps nativas. Este entorno de desarrollo fue construido desde el código fuente de **Eclipse**; por lo tanto, quienes hayan programado o programen en esta plataforma, se encontrarán en un ambiente familiar, dado que mantiene las mismas características que Eclipse.

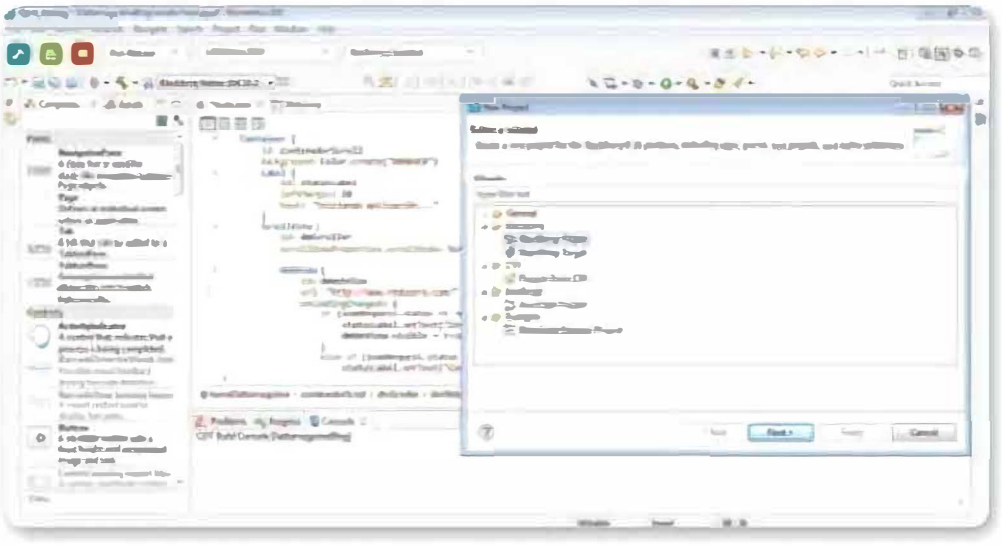


Figura 2. QNX Momentics IDE provee, en un entorno fácil y amigable, todas las herramientas para programar, de forma nativa, aplicaciones BB10.

Desarrollo basado en HTML5

Este último tiempo, BlackBerry comprendió, al igual que Microsoft, que el desarrollo de aplicaciones móviles basadas en la tecnología HTML5 no es malo, y que, a su vez, le permite que más desarrolladores se acerquen a la plataforma, llevando consigo sus aplicaciones móviles.

Así fue que decidieron incorporar **Apache Cordova** (aún conocido como **PhoneGap**) como la base principal de desarrollo de apps híbridas con HTML5. Por eso, hoy Apache Cordova está integrado como plugin al ambiente de desarrollo WebWorks de BB.

Figura 3. Apache Cordova fue integrado como plugin oficial a la interfaz WebWorks para desarrollar apps híbridas para Android.

La interfaz BB en HTML5

Para poder desarrollar aplicaciones híbridas que luzcan igual que la interfaz nativa de BB10, también contamos con una versión alternativa para integrarse con **jQuery Mobile** y así fácilmente crear la interfaz de BB desde CSS y JavaScript: **BBUI.js**.

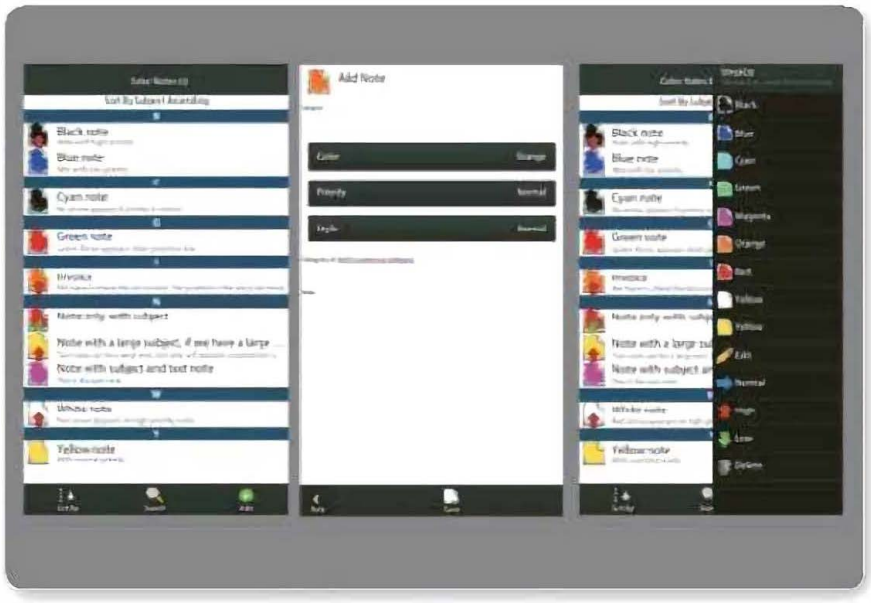


Figura 4. Aplicaciones como **Color Notes 10** demuestran que **BBUI.js** es una gran apuesta para crear interfaces BB10 para los desarrolladores WebWorks o PhoneGap.

El gran simulador

Todo gran entorno de desarrollo debe tener un gran simulador para completar el ambiente perfecto de código y **testing**. BB10 no es la excepción, ya que desde sus inicios el ambiente de desarrollo de esta plataforma puso a disposición de los programadores un excelente ambiente de testing. Este está compuesto por **VMWare Player** (**www.vmware.com/products/player**), el cliente que permite correr máquinas virtuales de diferentes sistemas operativos y las máquinas virtuales en sí de cada plataforma de BB. Estas se descargan desde el sitio web oficial de desarrolladores BB: **http://developer.blackberry.com/devzone/develop/simulator/bb10allsimversions.html**.

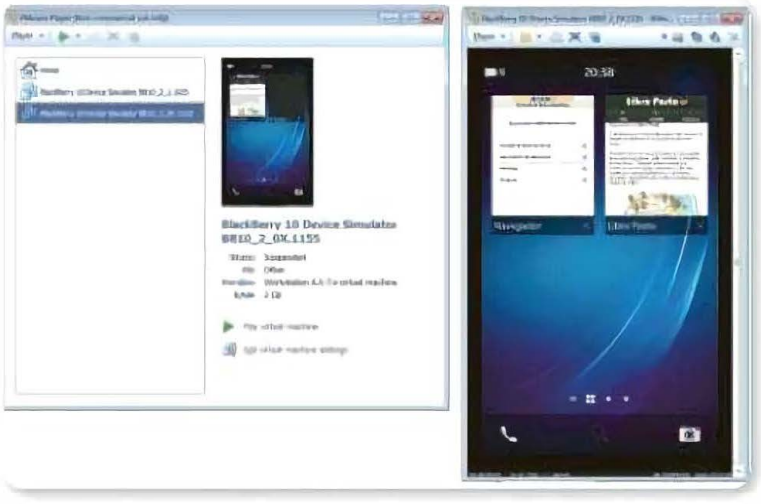


Figura 5. VMWare Player con dos máquinas virtuales que emulan a BB10 y BB10.2.

El uso de las máquinas virtuales BlackBerry nos permitirá, tanto en el desarrollo nativo como en el desarrollo web, testear nuestras aplicaciones en el entorno BlackBerry, si este es uno de los destinos donde nuestra WebApp correrá.

EMULADORES VÍA VMWARE PLAYER

El cliente de emulación **VMWare Player** no solo es utilizado para emular los diferentes sistemas operativos de BlackBerry, sino que también permite crear máquinas virtuales para instalar, por ejemplo, el sistema operativo Windows 8, entre otros.

Desarrollo mobile para Windows Phone

Otra de las plataformas más importantes en el mundo del desarrollo móvil es Windows Phone. Este sistema operativo móvil fue la apuesta de Microsoft para volver a ganar una cuota de mercado en un ecosistema que no ha dejado de crecer en la última década y en el cual la firma de **Redmond** tuvo una participación previa con Windows Mobile.

Junto con la llegada de Windows Phone, un sistema operativo móvil completamente renovado en cuanto a funcionalidades e interfaz gráfica, llegó la adaptación de sus herramientas basadas en **Visual Studio**. Visual Studio posee, desde su versión 2010, un soporte para desarrollo de aplicaciones nativas para Windows Phone. Este desarrollo se basa, en parte, en la tecnología **XAML** y, la otra parte, en la tecnología de sus lenguajes nativos: **C#** y **Visual Basic**.

> DESDE SU VERSIÓN 2010, VISUAL STUDIO POSEE SOPORTE PARA WINDOWS PHONE

Windows Phone 7.0 fue la primera versión de este renovado sistema operativo y, como browser mobile, incluyó una versión reducida de **Internet Explorer**. Luego llegó la actualización a Windows Phone **7.5** y, finalmente, la versión **7.8**, con mejoras constantes en la funcionalidad del sistema operativo. Por último, llegó al mercado Windows Phone 8, que incluyó como novedad principal el soporte de procesadores de doble núcleo.

Windows Phone 7 y 8

Microsoft firmó un convenio con **Nokia** para el lanzamiento masivo de la plataforma Windows Phone. Este acuerdo traía una serie de ventajas para ambas empresas, ya que Nokia fue, durante muchos años, pionera en materia de equipos celulares por su gran calidad de hardware y software, y Microsoft –si bien seguía gozando de la popularidad mundial del sistema operativo Windows en los escritorios de más del 90 % de los equipos– necesitaba recuperar rápidamente el terreno móvil cedido a sus principales competidores.

Cuando se firmó este acuerdo, Nokia llevaba varios años perdiendo mercado de un modo grosero: entre el 11 y el 23 % anual de usuarios de la plataforma Nokia se pasaba a otras tecnologías más modernas. Este problema surgió porque Nokia tenía tres sistemas operativos diferentes para su línea de telefonía celular, mientras que los grandes jugadores del momento –LG y Samsung, entre otros– habían concentrado las fuerzas en renovar su línea de equipos solo con Android.

Así fue como llegó al mercado la marca **Lumia**, con la que Nokia y Microsoft presentaron la plataforma Windows Phone como la novedad que haría resurgir de las cenizas a ambas empresas. Pero el tiempo trajo otro revés más para ambas firmas: Windows Phone 8 llegó muy rápido al mercado y vino optimizado de fábrica para aprovechar las bondades de los procesadores de dos y cuatro núcleos. Esto hizo que la primera línea de teléfonos con Windows Phone 7.x no pudiera migrar nunca hacia el nuevo sistema operativo móvil de Microsoft, porque todos esos equipos poseían un procesador mononúcleo.

Si bien este temprano lanzamiento de Windows Phone 8 no fue un gran acierto, el afán de la gente por tener lo último en materia de teléfonos inteligentes hizo que la falta de actualización para los equipos con Windows Phone 7.x no afectara en gran parte a ambas firmas.

Instalación de herramientas Microsoft

Visual Studio 2010 fue el primer IDE preparado para desarrollar aplicaciones nativas para Windows Phone 7.x. Este programa brinda soporte para desarrolladores mobile, tanto en su versión paga como en su versión express. Incluye un emulador de la plataforma Windows Phone que nos permite instalar y testear nuestras aplicaciones nativas; también posee otras funciones algo más avanzadas, que ofrecen la posibilidad de cambiar los temas de la interfaz entre oscuro y claro,

SDK PARA WINDOWS PHONE 8

Si utilizamos la última versión del **IDE Visual Studio 2013** en su versión profesional, necesitaremos contar con un sistema operativo Windows 8 profesional o superior. Windows 7 no soporta los emuladores WP de la versión 2013 de Visual Studio.

y de modificar la combinación de colores de los iconos, etiquetas y botones de la plataforma. Además, incluye el navegador Internet Explorer Mobile, desde el cual podremos probar nuestras WebApps sin problema.

Al momento de escribir esta obra, Visual Studio se encuentra en su versión **2013 update 2**. También posee la versión completa, paga, y su versión express, gratuita pero con ciertas limitaciones.

Descargar SDK de Windows Phone

Lo primero que debemos hacer para poder acceder al SDK de Windows Phone es suscribirnos como desarrolladores de la plataforma Microsoft. Si ya disponemos de una cuenta en alguno de los servicios de Microsoft, como ser **Hotmail**, **Live.com** u **Outlook.com**, esta nos servirá perfectamente para acceder a la plataforma de desarrolladores mobile de la firma de Redmond.

Para poder descargar las herramientas de Windows Phone debemos acceder a la siguiente URL: **https://dev.windowsphone.com/es-es/downloadsdk?logged_in=1**. Desde allí, podremos acceder a los instaladores del SDK para Windows Phone 8.1 (aún en beta al momento de escribir esta obra), al emulador de Windows Phone 8.1, al SDK para Windows Phone 8.0 y a los emuladores de Windows Phone 8.0, que también incluyen el emulador para la plataforma Windows Phone 7.5 o superior.

Figura 6. El **centro de desarrolladores de Microsoft** provee todas las herramientas para crear aplicaciones nativas o HTML5 para Windows Phone.

Visual Studio IDE para Windows Phone

El entorno de desarrollo para la creación de aplicaciones nativas para Windows Phone nos provee de todo lo necesario para poder crear nuestras propias aplicaciones nativas para esta plataforma.

Los desarrollos de aplicaciones WP segmentan su creación a través de diferentes técnicas. La premisa de Microsoft para la creación de aplicaciones WP es que estas se utilicen con el teléfono en **modo vertical**, de la misma forma que BlackBerry propone la creación de sus aplicaciones. Esto se debe a que la mayoría de los usuarios de teléfonos móviles lo utilizan mucho más tiempo en modo vertical que en modo horizontal.

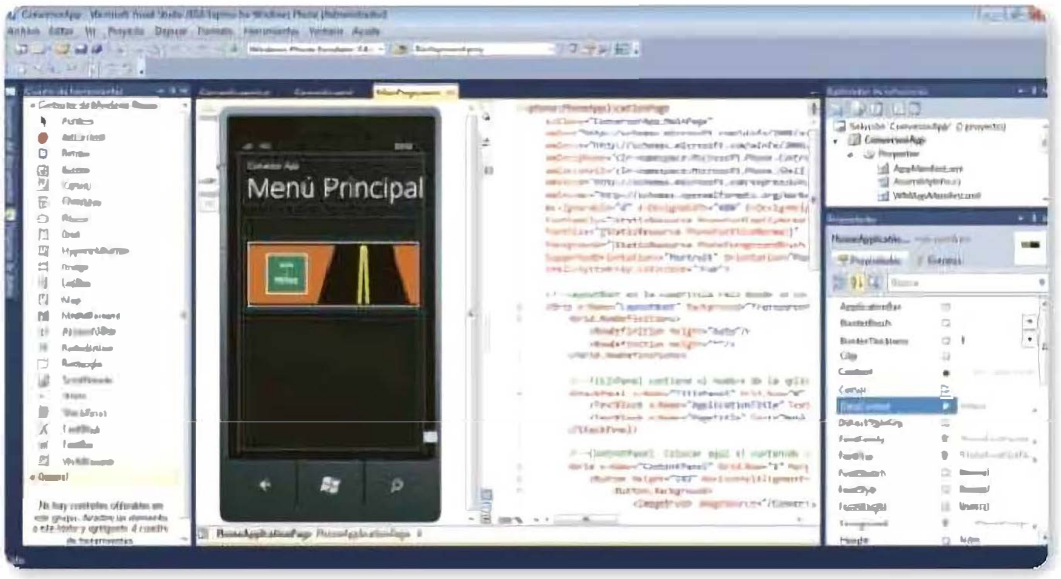


Figura 7. Visual Studio para WP nos permite tener una vista previa de nuestro desarrollo y, a su vez, crear los componentes tanto en modo código como arrastrándolos desde la barra de herramientas.

El lenguaje de programación para crear aplicaciones nativas para Windows Phone puede ser tanto Visual C# como Visual Basic. A su vez, la interfaz gráfica de este se realiza mediante los componentes de su barra de herramientas, arrastrándolos y soltándolos a la pantalla llamada **PhoneApplicationPage**, o a través del código **XAML**. Este último es un lenguaje que utiliza tags, similar al utilizado por CSS, XML y HTML para la creación de páginas web o componentes donde se configuran diferentes atributos.

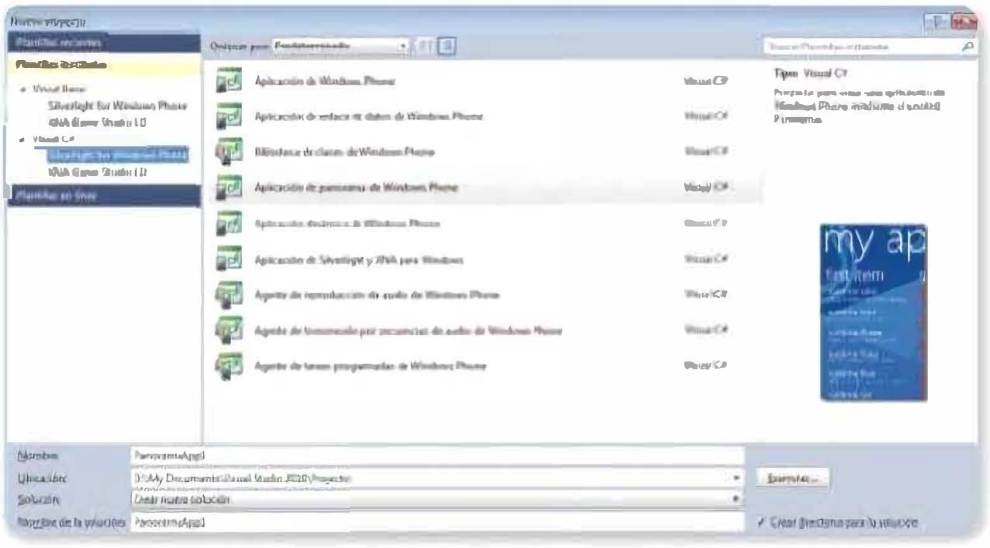


Figura 8. En la creación de un nuevo proyecto para WP, podemos elegir el lenguaje y el tipo de diseño que nuestra app tendrá.

Windows Phone nos propone crear aplicaciones con una ventana simple, **Standalone App**, cuyo proyecto se conoce simplemente como **Aplicación de Windows Phone**. Otra opción es crear una **Aplicación de enlace de datos de Windows Phone** que nos permite generar un array dinámico de ítems a partir de, por ejemplo, un código **JSon** o **RSS**.

También disponemos de la creación de una **Aplicación Panorama**, la cual extiende su pantalla hacia los laterales del equipo, permitiendo su visualización a través de hipervínculos que conectan cada sección o directamente navegándola a través del deslizamiento horizontal de la pantalla hacia la izquierda o derecha.

Figura 9. El tipo de proyecto **Aplicación Panorama** nos permite conectarnos a fuentes de noticias RSS y crear rápidamente aplicaciones desde blogs de noticias con la estética propia del sistema operativo WP.

Emular WebApps en la computadora

Existen también complementos para navegadores web que sirven para emular aplicaciones web dentro de un browser instalado en nuestra computadora. Desde hace un tiempo, Google incluye en su portal de aplicaciones y extensiones para Chrome un complemento llamado **Ripple Emulator**, el cual nos ayuda a emular fácilmente WebApps y así poder ver cómo lucen estas en las pantallas de dispositivos móviles, ya sea tablets o smartphones.

EL EMULADOR DE WEBAPPS RIPPLE EMULATOR ES UNA DE LAS EXTENSIONES PARA CHROME

Desde nuestro navegador Chrome o desde **Chromium**, podemos instalar Ripple mediante la siguiente URL: **https://chrome.google.com/webstore/detail/ripple-emulator-beta/geelfhphabnejjhdalkjhgipohgpdnoc?hl=es-419**. Ripple Emulator es actualmente un proyecto open source bajo la licencia Apache. Simplemente debemos agregarlo como complemento a nuestro browser y ya estaremos listos para sacarle todo el provecho.

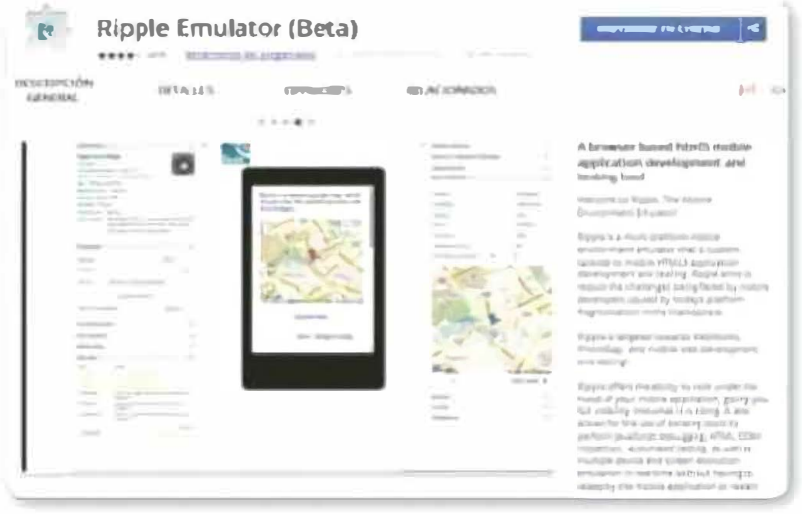


Figura 10. **Ripple Emulator** es un verdadero aliado a la hora de testear WebApps bajo el motor WebKit, así como aplicaciones híbridas desarrolladas con PhoneGap.

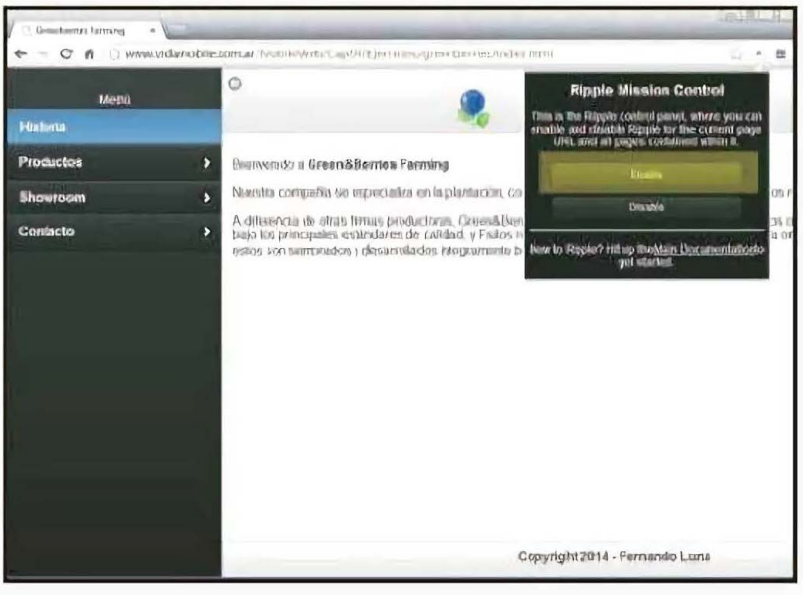
Utilización de Ripple Emulator

Una vez instalado Ripple Emulator, se creará un icono en la barra de herramientas, justo después de la barra de direcciones de Chrome. A su vez, desde **chrome://extensions/** podemos tener un control cuando deseamos activarlo o desactivarlo de los complementos de Google Chrome o Chromium.

Su utilización es muy fácil. Suponiendo que deseamos ver el formato web de una página creada para plataformas móviles, lo primero que debemos hacer es cargar la página en una pestaña del navegador web. Luego ejecutamos el complemento desde la barra de herramientas; este nos preguntará si queremos visualizar una WebApp (y podremos seleccionar el dispositivo en donde esta se verá) o si estamos construyendo e intentando visualizar un desarrollo realizado en PhoneGap o Apache Cordova.

PAP: ACTIVAR Y DESACTIVAR RIPPLE EMULATOR

01 Primero, cargue la página web a visualizar. Esta se debe encontrar en un servidor web original o webserver local. No puede visualizarse ninguna WebApp desde el disco local de la computadora.



02 A continuación, Ripple Emulator le consultará qué tipo de aplicación web desea emular. Entre ellas, podemos ver WebApps, aplicaciones PhoneGap y desarrollos WebWorks 1.0, 2.0 y para tablets de la línea BlackBerry.

03 Al cargarse la interfaz Ripple Emulator, desde el apartado `Devices` podrá seleccionar el dispositivo correcto que permita emular en la pantalla del emulador nuestra WebApp. Podrá intercambiar diferentes dispositivos desde la lista extensa que ofrece Ripple.

04 Para volver todo a la normalidad, debe seleccionar nuevamente el botón `Ripple Emulator` de la barra de herramientas y presionar `Disable`. De no desactivarlo, cada vez que vuelva a cargar esta URL se visualizará dentro de Ripple Emulator con la última configuración seleccionada.

El aporte de Firefox

El navegador web **Mozilla Firefox** de escritorio incluye, desde sus últimas versiones, la posibilidad de adaptar una web a un tamaño de pantalla predeterminado. Para visualizar esta opción, primero debemos cargar en una pestaña la página deseada y luego presionar **CTRL** + **SHIFT** + **M**, o ir a **Menú principal/Desarrolladores/Vista de diseño adaptado**.

Desde el menú superior de la vista de diseño adaptada, podemos recargar la página, emular los toques de pantalla como si fuera un dispositivo móvil, capturar la pantalla y guardarla como imagen

EMULADORES IOS

Lamentablemente, Apple nunca puso a disposición emuladores de su sistema operativo móvil para la plataforma Windows o Linux, ya que las herramientas que ellos exigen utilizar para desarrollo sólo disponen de una versión que puede instalarse y ejecutarse en OS-X.

para propósitos de documentación, y hasta cambiar la posición de la pantalla a modo horizontal. También, desde el combo desplegable podemos seleccionar diferentes medidas de pantalla y hasta agregar nuestras medidas predeterminadas.

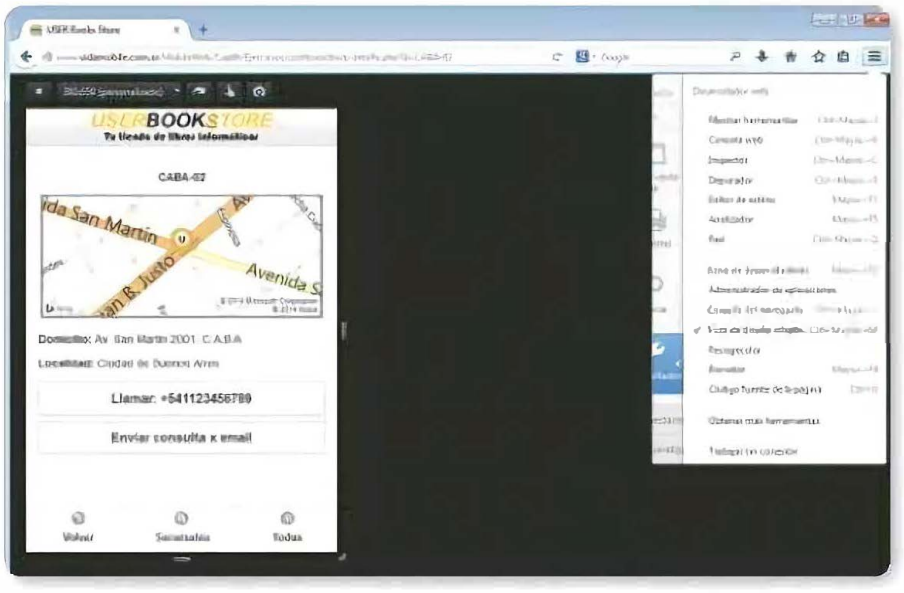


Figura 11. El browser de escritorio Firefox incorporó la funcionalidad **Vista de diseño adaptado**, que nos permite emular una web en un tamaño de página específica.

ALTERNATIVAS PARA DESARROLLO EN OS-X

Para desarrollar o emular con herramientas Apple oficiales, podemos contratar el servicio denominado **Mac In Cloud** (**www.macincloud.com**) que, por un pequeño costo, nos permite alquilar una computadora Apple con OS-X y acceder a esta por escritorio remoto.

Firefox OS

Este apéndice nos brinda una rápida introducción al mundo de Firefox OS, un nuevo jugador en el terreno mobile que apunta a acercar la telefonía móvil de los mercados emergentes al paradigma del desarrollo web, basando toda su interfaz en la tecnología HTML5. Esto permitirá que cualquier desarrollador o diseñador web pueda integrar sus productos rápidamente en esta plataforma.

Un nuevo jugador en el ecosistema móvil

La **Fundación Mozilla** lleva más de una década desarrollando diversas aplicaciones bajo la filosofía open source. Su principal misión es mantener una selección de productos que ponen a disposición del usuario una nueva alternativa en el ecosistema de aplicaciones de escritorio y –ahora– móviles.

Dentro de sus principales estrellas, podemos destacar a **Firefox** y **Thunderbird**, navegador web y cliente de correo electrónico, respectivamente. Toda la línea de productos de Mozilla está disponible de manera gratuita, ya que son de código abierto y multiplataforma, lo que nos permite utilizar el mismo producto en **Windows**, **Linux**, **BSD** y **OSX**, sin existir diferencias entre estas plataformas.

Con los años, el fuerte crecimiento de las plataformas móviles hizo que Mozilla lanzara una versión móvil de su navegador web para casi todas las plataformas existentes. Hoy podemos usar Firefox tanto en las computadoras de escritorio como en las tablets y celulares más populares.

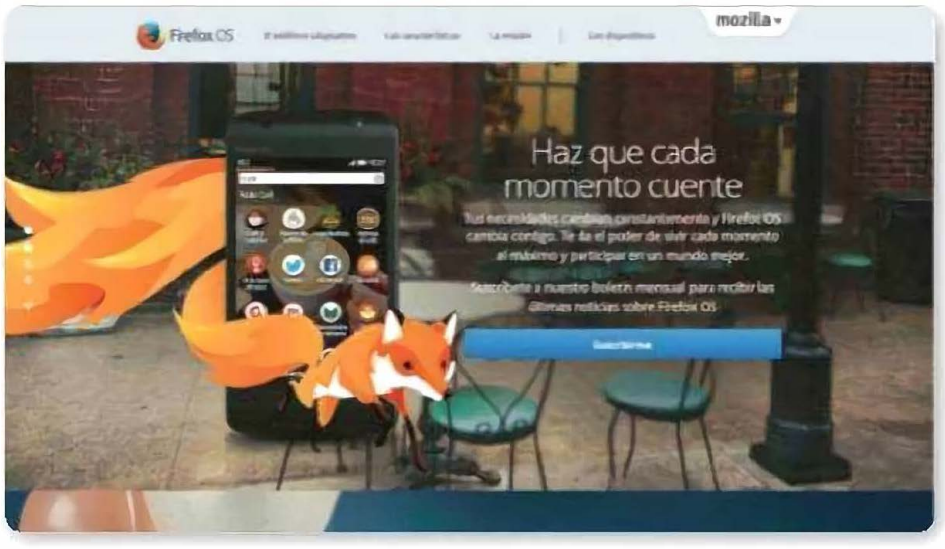


Figura 1. En la web oficial de Firefox OS (**www.mozilla.org/es-ES/firefox/os**) encontramos toda la información referente a la plataforma.

Durante mucho tiempo, Mozilla recibió apoyo económico de Google, su principal interesado en el correcto desarrollo de productos abiertos. Cuando Google decidió seguir su camino por el mundo web, lanzando

Gmail como correo electrónico basado en web y, más tarde, Chrome como navegador web, Mozilla dejó de recibir apoyo de esta compañía. Aun así, con la financiación de otros interesados, pudo seguir manteniendo hasta hoy su línea de productos.

La llegada en 2012 de **Telefónica de España** como financiador del proyecto Mozilla ayudó a esta fundación a desarrollar más sistemas y aplicaciones para el mundo móvil. Así fue como se potenció la idea de crear un sistema operativo propio, de código abierto y con un costo casi nulo para los fabricantes de hardware móvil que eligieran integrar en sus productos este sistema operativo. De allí nació **Firefox OS**.

> FIREFOX OS ES UN SISTEMA OPERATIVO DE CÓDIGO ABIERTO DE LA FUNDACIÓN MOZILLA

El objetivo de Firefox OS es abaratar los costos de los fabricantes de hardware y operadores de telefonía móvil, para que competir en las grandes ligas, con equipos de un costo económico, no sea un impedimento. Así fue como, luego de la presentación de un sistema operativo de código abierto –bautizado originalmente **Seabird** e inspirado en Android–, Mozilla pudo llevar a cabo su desarrollo completo.

Arquitectura de Firefox OS

Firefox OS divide su arquitectura en tres componentes:

- **Gonk**: oficia de capa de bajo nivel de **B2G** con su kernel basado en Linux y une este a una segunda capa que se ocupa de la interacción con el hardware.
- **Gecko**: cumple la función de entorno de ejecución de las aplicaciones en general.
- **Gaia**: oficia de interfaz gráfica del sistema operativo. Una vez que la capa B2G se ejecutó, todo lo referente a la gráfica que aparece en la pantalla de Firefox OS es responsabilidad de esta capa. Las aplicaciones de bloqueo, marcador telefónico, mensajes de texto y menú de aplicaciones, entre otras funcionalidades más, forman parte de la capa Gaia.

Las aplicaciones que corren bajo Firefox OS están implementadas bajo el estándar HTML5, CSS3 y JavaScript. Estas alcanzan, así, una interfaz que se ejecuta correctamente en este sistema operativo.

Características del S.O.

Firefox OS posee una serie de características que lo diferencian por completo del resto de los sistemas operativos móviles.
Entre ellas, destacamos las siguientes:

- **WebApps**: todas las aplicaciones de este S.O. están basadas en web (HTML5, CSS, JS). No existe hoy un framework que permita codificar y compilar las aplicaciones de manera nativa. Aun así, las aplicaciones son ejecutadas de manera nativa, sin depender del navegador web. A través de un archivo específico, el S.O. lee las características que tiene cada app y las ejecuta en pantalla completa.
- **Storage**: el almacenamiento de datos de estas apps y también los que guarda el S.O. utilizan **SQLite** como base de datos.
- **UI**: la interfaz gráfica de las aplicaciones es adaptable a cualquier tipo de pantalla y resolución, sea o no táctil. Esto es posible gracias a la capa Gaia. Utiliza la fuente tipográfica por defecto, **Fira Sans**, y los iconos del sistema operativo poseen bordes redondeados. Igualmente, la creación de estos queda a criterio del autor de cada app.
- **Geolocalización**: el GPS de Firefox OS y el sistema de geolocalización de usuario utiliza **HERE**, de Nokia, para visualizar mapas y orientar a los conductores hacia su destino deseado.

Figura 2. Firefox OS nos pone a disposición todas las funciones de cualquier smartphone y ejecuta tanto apps nativas como WebApps sin complicaciones.

Las notificaciones, la reproducción de audio y video y la toma de fotografías también se pueden llevar a cabo en este sistema operativo. Y, dado el mercado elegido para desplegar su masificación, Firefox OS cuenta con una **Radio FM** dentro del sistema operativo, obviando así la popularidad de la competencia, que promueve la sintonía radial mediante **streaming**.

Figura 3. El simulador de Firefox OS pone a disposición todas las características del S.O. y funciona en Windows, Mac y Linux por igual.

Codificando apps

Como bien dijimos en el inicio de este apéndice, Firefox OS promueve el desarrollo de aplicaciones íntegramente basadas en tecnologías web, lo que permite que cualquier persona con conocimientos básicos en estas tecnologías pueda rápidamente crear una app compatible para este sistema operativo. Esto nos permite

PANEL DE ADMINISTRACIÓN

Firefox browser nos pone a disposición **App Manager**, el panel de administración principal de Firefox OS, desde donde podemos administrar las aplicaciones del dispositivo, detectar un teléfono físico, ejecutar el simulador e instalar aplicaciones, entre otras cosas. Se accede desde esta URL: **about:app-manager**

garantizar que todas las aplicaciones web que fueron desarrolladas a lo largo de este libro son totalmente compatibles con Firefox OS.

Y dado que Firefox OS no posee un framework o IDE específico para el desarrollo de aplicaciones, apostando por el estándar web, la Fundación Mozilla pudo centralizar su esfuerzo en el desarrollo de un simulador que se integra con el navegador web, y que nos permite, así, probar todos nuestros desarrollos tal como si lo hiciéramos sobre un dispositivo físico.

El simulador

Para poder instalar el simulador Firefox OS, solo debemos contar con el navegador web Firefox. El simulador se instalará como un complemento de este y se ejecutará de manera fácil desde la interfaz del navegador.

> PARA INSTALAR EL SIMULADOR FIREFOX OS SOLO NECESITAMOS EL NAVEGADOR WEB FIREFOX

Para instalarlo, debemos acceder al siguiente link desde Firefox browser: **www.addons.mozilla.org/es/firefox/addon/firefox-os-simulator**. Luego, debemos presionar el botón **Agregar a Firefox** y esperar que finalice el proceso de instalación. Se descargará el simulador y luego se agregará como complemento de Firefox.

Finalizado este proceso, podremos acceder al simulador desde una nueva pestaña, dirigiéndonos al menú **Firefox /Desarrollador Web/ Firefox OS Simulator**. Se cargará el **Addon** en esta pestaña y, sobre el lateral izquierdo, veremos la

FIREFOX OS KNOWLEDGE DATABASE

Para seguir al pie de la letra la correcta instalación de Firefox OS y conocer con mayor profundidad sus características, podemos ingresar al link: **www.developer.mozilla.org/es/docs/Mozilla/Firefox_OS/Using_the_App_Manager**, donde encontraremos un instructivo completo y detallado.

opción **Simulator Stopped**. Presionando sobre esta, el simulador se iniciará en una ventana emergente visualizando la pantalla inicial del sistema operativo móvil.

Ajustes

Desde el menú de aplicaciones, podemos acceder a los ajustes generales del sistema operativo, donde podremos configurar opciones como: idioma, país, horario, fecha, etcétera. Una vez configuradas las opciones por defecto de nuestro simulador, podremos acceder a la descarga de aplicaciones desde **Firefox Marketplace** o comenzar a adaptar nuestras WebApps para distribuirlas en esta plataforma.

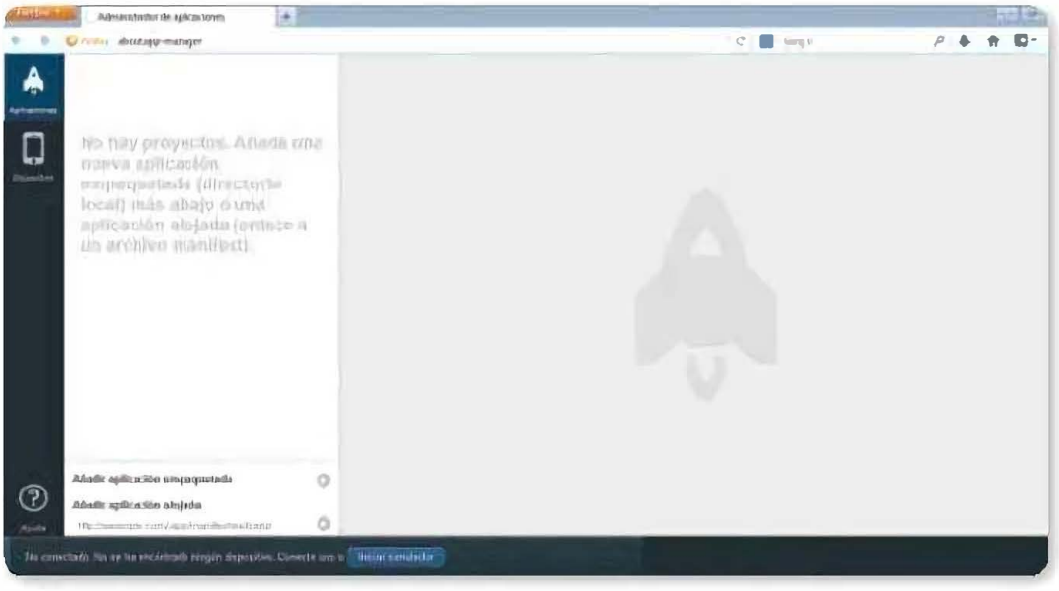


Figura 4. El panel de administración será el centro de pruebas de nuestras aplicaciones sobre el simulador o dispositivo físico Firefox OS.

Requisitos de una app

Las métricas para el desarrollo de una app que se ejecute de manera nativa en Firefox OS no poseen requisitos complejos como puede tener una app de Android o iOS. Si deseamos cumplimentar el estándar UI, podemos utilizar las librerías nativas basadas en CSS bautizadas bajo el nombre **FFOS UI Style**, las cuales son distribuidas a través del portal **GitHub** de la Fundación Mozilla: **www.github.com/mozilla-b2g/gaia/tree/master/shared**.

Firefox OS UI Style

De este repositorio debemos tomar los archivos ubicados en la carpeta **/JS/** y en la carpeta **/STYLE/**. El resto de las carpetas posee otros componentes, aunque aún estos se encuentran en modo borrador y no se recomienda utilizarlos.

Los archivos CSS poseen clases definidas para esta interfaz, similares a las de jQuery Mobile, con lo cual al escribir un archivo HTML, podremos invocar tags como **<data-role="dialog" data-type="confirm">**, y así obtendremos una ventana de diálogo que visualizará los botones de confirmación para ejecutar otras acciones y/o cerrarse.

Dentro de los **skins** o **temas** que Firefox OS nos pone a disposición, se incluyen actualmente dos temas específicos, a los cuales accedemos a través de las siguientes líneas de código:

```
<class="skin-organic">
//nos devolverá el tema claro

<class="skin-dark">
//nos devolverá el tema oscuro
```

Firefox OS propone, a través de estos archivos, lograr la misma simpleza que jQuery Mobile o **Sencha Touch** ponen a disposición del desarrollador.

Diseñar los iconos de nuestra app

Para poder diseñar iconos que cumplan con la estética requerida por Firefox OS, podemos acceder a la siguiente guía: **www.mozilla.org/en-US/styleguide/products/firefox-os/icons** que nos permitirá conocer a fondo los requerimientos principales en cuanto a tamaño, forma, profundidad y otras características propias del gráfico que distinguirá nuestra aplicación del resto dentro del sistema operativo.

UI alternativo

Igualmente, si no queremos involucrarnos de lleno con el desarrollo de aplicaciones que utilicen el estilo nativo de Firefox OS, podemos recurrir a la librería de jQuery Mobile, sin que esto sea un impedimento para distribuir apps en Mozilla Marketplace.

Manos a la obra

Veamos a continuación cómo podemos adaptar nuestra app **Green&Berries Farming** para que corra de manera nativa en Firefox OS. Para ello, copiemos la carpeta del proyecto a una nueva ubicación y, una vez realizado esto, renombremos la carpeta raíz del proyecto como **FFOS – GREEN&BERRIES FARMING**.

Realizado esto, el siguiente paso será crear un archivo **Manifiesto**, al igual que el desarrollado en el capítulo **PhoneGap**, que nos permitirá indicarle al sistema operativo las principales características de nuestra app.

Manifest.webapp

El archivo **Manifest.webapp** es el principal objeto al que Firefox OS apunta cuando ejecuta una aplicación. En este, el sistema operativo móvil encontrará toda la información referente a la aplicación, para poder instalarla en el sistema operativo, mostrar su correspondiente icono e identificar, entre otras cosas, el título y qué funcionalidades del software o hardware utilizará al momento de ejecutarse.

Para ello, dentro del archivo en cuestión, debemos volcar el siguiente contenido, que incluye los comandos a través de los cuales identificará las características mencionadas de nuestra app. Antes crearemos, dentro del directorio raíz de nuestro proyecto, un archivo con este nombre. A continuación, escribiremos el siguiente código:

```
{
 "version": "1.0",
 "name": "Green&Berries Farming",
 "description": "Ejemplo funcional de App para Firefox OS – by Fernando Luna",
 "launch_path": "/index.html",
 "icons": {
  "48": "/imagenes/gb-48.png",
  "60": "/imagenes/gb-60.png",
  "128": "/imagenes/gb-128.png"
 },
 "developer": {
  "name": "Fernando Luna",
  "url": "http://www.vidamobile.com.ar"
```

```
    },
    "installs_allowed_from": [
     "*"
    ],
    "locales": {
     "en": {
      "description": " Ejemplo funcional de App para Firefox OS – by Fernando
Luna",
      "developer": {
       "name": "Fernando Omar Luna",
       "url": "http://www.vidamobile.com.ar"
      }
     }
    },
    "default_locale": "es",
    "permissions": {},
    "fullscreen": "true"
   }
```

Como podemos ver, este código es de muy fácil interpretación. En él, indicamos el nombre de la app, su versión, el nombre del desarrollador y la URL oficial. También podemos especificar una descripción de la app y volver a detallar todas sus características en todos los idiomas que deseemos hacerlo, a través del apartado **locales** y del ID del idioma **es**, **en**, etcétera.

Default_locale nos permite establecer el idioma principal con el que la app se ejecutará, y, en el apartado **permissions**, podremos establecer la funcionalidad de hardware o sistema operativo que la app utilizará.

Si este apartado queda en blanco, significa que se utilizarán todas las características del sistema operativo y del hardware.

DASHBOARD DE FIREFOX OS

Desde el **Panel de control** de Firefox OS podemos no solo cargar apps desde su carpeta contenedora, sino también, a través de una URL, validarla, editar el archivo **manifest.webapp**, y actualizar el package del simulador si cambiamos alguno en el proyecto, entre otras cosas.

Por último, el apartado **fullscreen** nos indica si la app se ejecutará o no a pantalla completa.

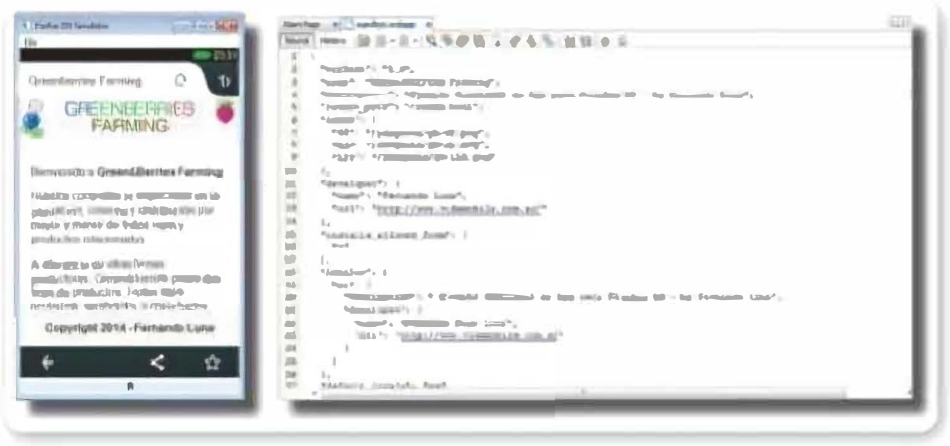


Figura 5. Una aplicación web como **Green&Berries** y `manifest.webapp` transformarán esta WebApp en una app nativa de Firefox OS.

Testear nuestro desarrollo

Solo nos queda testear la app en el simulador. Para ello, debemos ingresar a este y, desde la sección **Dashboard**, presionar el botón **Añadir aplicación empaquetada**. Buscamos, a continuación, la ruta de la carpeta de nuestra WebApp y presionamos **Seleccionar**.

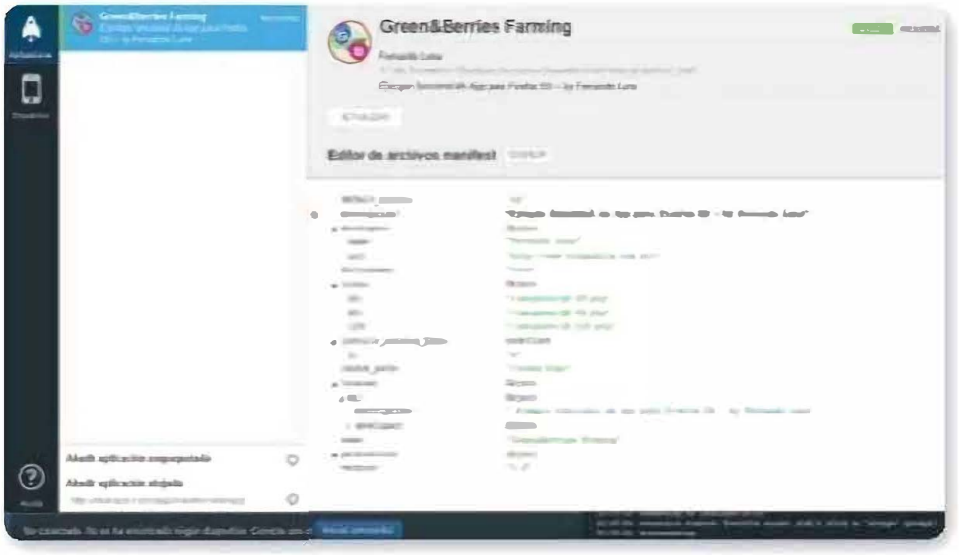


Figura 6. Si creamos correctamente el archivo `manifest.webapp`, nuestra app aparecerá en el **dashboard** de Firefox y se instalará en el simulador o dispositivo de desarrollo.

Nuestra app se instalará en Firefox OS y se ejecutará sin problema, tal como se muestra en la siguiente figura.

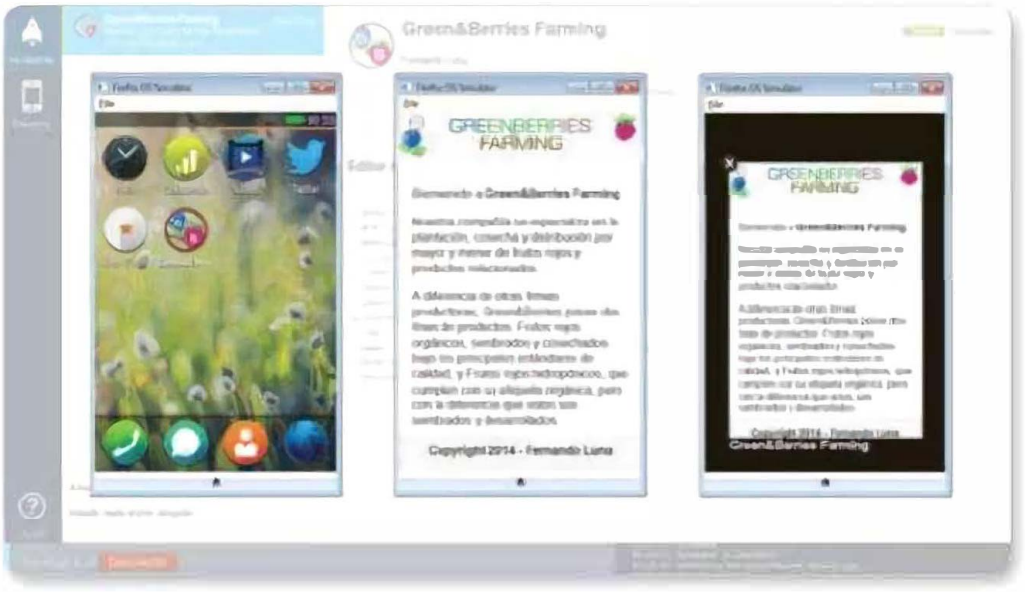


Figura 7. Nuestra aplicación ya se encuentra instalada en Firefox OS y corre de manera independiente al navegador web.

Distribución de aplicaciones

Por último, para distribuir nuestras aplicaciones en Mozilla Marketplace, debemos inscribirnos como desarrolladores de la plataforma. Y para subir nuestras apps debemos crear, previamente, un archivo Zip con la carpeta que contiene todo el proyecto.

Mozilla enviará nuestra app para su correspondiente certificación y, al cabo de unos días, recibiremos el OK de la implementación de esta dentro del Marketplace, o el correspondiente KO con los comentarios que indican por qué fue rechazada nuestra app.

Si queremos distribuir aplicaciones por primera vez, Mozilla Marketplace es la mejor opción para inducirnos en el fascinante mundo del desarrollo de las aplicaciones móviles, tanto por su simpleza y costo cero, como por otras tantas bondades propias de la plataforma.

DESARROLLO WEB PARA DISPOSITIVOS MÓVILES

Este libro está dirigido a quienes buscan acercarse a la programación de soluciones para el creciente ecosistema de tablets y smartphones. En cada capítulo se aborda un desarrollo web con acceso a características propias de los sistemas operativos móviles iOS, Android, Windows Phone, BlackBerry y Firefox OS. Entre otros proyectos, invocaremos llamados telefónicos y videoconferencias desde una web, y aprovecharemos las capacidades de geolocalización y soluciones que interactúen con redes sociales. Finalmente, aprenderemos a convertir una web móvil en una aplicación nativa.

> **La integración de las apps con el hardware de los equipos es casi infinita. El único límite es nuestra imaginación como desarrolladores.**

EN ESTE LIBRO APRENDERÁ:

- **Plataformas y tecnologías móviles:** sistemas operativos móviles. Diferencias entre WebApp, App nativa y App híbrida. Tecnologías de la web actual.
- **HTML5:** navegadores, sistemas operativos y motores de renderizado. Declaraciones y metatags. HTML5 para aplicaciones móviles. Geolocalización.
- **jQuery Mobile:** definición. Uso local o remoto. Configuración de una WebApp. Componentes y estructura.
- **WebApps para iOS y Android:** diseño. Prestaciones de Safari y de Google Chrome mobile. Visualización de una WebApp como nativa.
- **Almacenamiento local y aplicaciones offline:** Local Storage y Session Storage. Creación de un formulario. Bases de datos Web SQL.
- **PhoneGap:** arquitectura. Integración con Dreamweaver. Transformación de una WebApp en híbrida.
- **Programación nativa para BlackBerry 10 y Windows Phone:** herramientas de desarrollo. Emulación de WebApps en la computadora con Ripple Emulator.

SOBRE EL AUTOR

Fernando Luna es analista funcional de sistemas y cuenta con una diplomatura en desarrollo de aplicaciones para dispositivos móviles por la UTN. Es colaborador de las revistas *Power* y *Users* y autor de *Visual Basic 2010*, publicado por esta editorial.

NIVEL DE USUARIO

Intermedio

CATEGORÍA

Desarrollo / Internet / Mobile

ISBN: 978-987-1949-83-0

REDUSERS.com
En nuestro sitio podrá encontrar noticias relacionadas y también participar de la comunidad de tecnología más importante de América Latina.

PROFESOR EN LÍNEA
Ante cualquier consulta técnica relacionada con el libro, puede contactarse con nuestros expertos: profesor@redusers.com.