Empleo de Tecnologías Web de última Generación para la realización de Aplicaciones Web Adaptables a Múltiples Dispositivos

Por

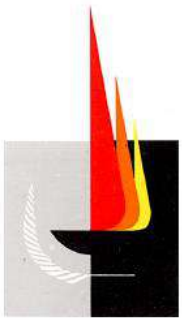
Carlos Ariel Schiavoni

Tesis presentada como requisito para obtener el título de

Ingeniero en Sistemas

en la

Universidad Nacional de La Pampa



Universidad Nacional de La Pampa
Facultad de Ingeniería

Director: Dr. Luis Olsina

General Pico, La Pampa, Argentina 2013

Prefacio

Esta Tesina se presenta como parte de los requisitos para optar al grado Académico de Ingeniería en Sistemas, de la Universidad Nacional de La Pampa y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otra. La misma contiene los resultados obtenidos durante el período comprendido entre el 20/04/2012 y el 12/05/2013, bajo la dirección del Dr. Luis Olsina.

Carlos Ariel Schiavoni

Facultad de Ingeniería

Universidad Nacional de La Pampa

Mayo de 2013

Agradecimientos

A Luis Olsina por sus consejos y por haberme guiado de manera clara y precisa en la realización de este trabajo.

A mi familia por brindarme la posibilidad de cumplir con mis estudios y hacer realidad este sueño, su empuje y motivación me alentaron para cumplir este objetivo tan importante para mí.

A mis compañeros de trabajo, facultad y amigos por compartir sus experiencias, consejos y conocimientos que me nutrieron durante el desarrollo de esta carrera.

Índice de contenidos

Índice de figuras

Índice de tablas

Resumen

La diversidad de dispositivos electrónicos que hallamos hoy en día es indiscutible, Smartphones, Tablets, PCs, Laptops, Netbooks, Consolas de juegos, etc. En el momento que una empresa decide desarrollar una aplicación intenta llegar a la mayor cantidad de consumidores posibles, por lo cual no sólo debe desarrollar su aplicación para los diferentes dispositivos sino también para las diferentes plataformas que estos conllevan (Android, iOS, Windows, Web, etc.). Esta enorme segmentación incurre grandes costos en las empresas que deben contratar personal calificado en numerosas tecnologías o conformase con los usuarios de las plataformas predominantes.

El presente trabajo expone un conjunto de metodologías que aprovechan el avance de HTML5 y permiten crear una única aplicación web adaptable a múltiples dispositivos. También se presenta un prototipo implementado bajo el nombre WASt que exhibe el marco tecnológico utilizado y el resultado del trabajo.

1 Introducción

La web se ha convertido en la plataforma dominante para la distribución de aplicaciones a cientos de millones de usuarios, hoy en día no basta considerar que una aplicación web (Webapp) solo será accedida desde una computadora. Si no se piensa en la amplia diversidad de usuarios accediendo desde dispositivos disímiles, la aplicación no estará preparada para responder a nuevas experiencias de usuario. Esto pondrá de manifiesto una percepción de baja calidad que llevará a los usuarios a dejar de utilizarla.

Se espera que las nuevas Webapps escalen fácilmente desde algunos usuarios a millones con el mínimo esfuerzo, al mismo tiempo se exige adaptabilidad al dispositivo o plataforma desde la cual se accede para explotar al máximo la experiencia de usuario en ese contexto de ejecución específico. Este problema no es novedoso y la solución que prevalece es construir una aplicación nativa para cada plataforma y dispositivo.

La actualidad muestra una gran segmentación del mercado en cuanto a los dispositivos existentes (Smartphone, Tablet, Notebook, Netbook, PC, etc.) y las plataformas que los gobiernan en sus diferentes versiones (Android, iOS, WebOS, Windows Phone, etc.) por lo cual los desarrolladores optan por implementar la aplicación en las plataformas de mayor popularidad, entre ellas Android y iOS, con el objetivo de alcanzar la mayor cantidad de usuarios con el mínimo esfuerzo.

1.1 Motivación

La chispa que enciende la motivación de este trabajo se origina en el hecho de que la mayor parte de las aplicaciones implementan funcionalidades que pueden ser alcanzadas con tecnologías estándar como HTML5, JavaScript y CSS3 sin degradar notablemente la performance y experiencia de usuario. De esta manera no es necesario re-implementar la aplicación en los distintos SDKs de cada plataforma y se alcanza una solución única para cubrir todos los dispositivos.

Una de las mayores ventajas de las aplicaciones web es que se ejecutan en cualquier browser, entre ellos los presentes en los dispositivos móviles, y también pueden ser re-empaquetadas como aplicaciones nativas en las diferentes plataformas. La amplia gama de plataformas a soportar, junto con la potencia que los browsers van adquiriendo con el correr del tiempo conduce a optar por HTML5 como la solución “write once, run many” [1].

HTML5 no está solo, el surgimiento y avance de las tecnologías web, entre ellas JavaScript, CSS3, NodeJS junto al nacimiento de numerosos frameworks y herramientas, ponen de manifiesto nuevos patrones de desarrollo que contribuyen a mitigar varios problemas que actualmente acarrean las aplicaciones web avanzadas (escalabilidad, notificaciones en tiempo real, modularización, reutilización, testing, etc.).

La motivación principal del trabajo radica en lograr, mediante el empleo de las tecnologías previamente mencionadas, la implementación de un prototipo de aplicación web con una base de código compartida que adecúe la interfaz y experiencia de usuario en tres dispositivos diferentes: PC/Laptop, Tablet y Smartphone.

El prototipo pretende:

- Demostrar la viabilidad de HTML5 como plataforma para la construcción de una única Webapp adaptable a múltiples dispositivos, frente a la construcción de diversas aplicaciones en cada una de las plataformas nativas. Así, los desarrolladores y diseñadores codifican la aplicación solo una vez orientada a un browser moderno y evitan la necesidad de realizar trabajo extra para otros dispositivos o plataformas.
- Explorar y exponer las principales metodologías (Mobile First, Responsive Web Design, Adaptive Web Design) utilizadas actualmente por las Webapps para lograr adaptabilidad y flexibilidad al contexto con la misma base de código. Para observar la flexibilidad el usuario del prototipo será capaz de acceder desde distintos dispositivos o redimensionar el tamaño de la ventana del browser y ver como el texto y los diferentes elementos pasan a tener una fisonomía acorde al dispositivo o tamaño de la pantalla.
- Exhibir herramientas de desarrollo, frameworks y librerías que aportan enorme valor a la construcción de Webapps complejas. Éstas mueven mucha de la lógica al cliente y es necesario todo un nuevo conjunto de herramientas y librerías que agilicen la implementación en este nuevo entorno. Además la testeabilidad y modularización de la aplicación son conceptos primordiales a la hora de encarar el desarrollo de una Webapp de estas características.

1.2 Enfoque tradicional de adaptabilidad a múltiples dispositivos

En los últimos años los avances tecnológicos a nivel hardware han sido abrumadores y se han expandido hacia otros tipos de dispositivos agrupados en el dominio mobile-aware.

Las capacidades de hardware de estos dispositivos han crecido notablemente, procesadores de dos y cuatro núcleos a frecuencias mayores a 1GHz, memoria RAM en el orden de los gigabytes, chips para aceleración gráfica, pantallas de altas resoluciones y soporte para pantallas táctiles, capacidades de conexión inalámbrica con alta tasa de transferencia de datos, etc.

Debido a toda esta nueva gama de dispositivos y formas de interacción surgen muchos problemas a resolver en el mundo del software, ¿cómo construir una aplicación que se adapte a la UI que quiero para cada dispositivo?,¿cómo proveer optimizaciones para el uso de la aplicación mediante la interacción con pantallas táctiles?, ¿cómo reaccionar ante los cambios de orientación?, si la aplicación tiene varias plataformas como objetivo, ¿en necesario reescribirla?

El enfoque tradicional opta por crear una aplicación web para el mundo “desktop” la cual generalmente está diseñada para soportar múltiples browsers (Chrome, Internet Explorer, Firefox, Opera, etc.). Al mismo tiempo se diseña e implementa una aplicación nativa paracada dispositivo (Smartphone, Tablet) y plataforma (Android, iOS), aprovechando todas las características nativas que provee.

1.3 Enfoque propuesto

La eclosión de dispositivos y plataformas hace replantear la idea de mantener varias instancias de desarrollo para lograr el soporte a cada combinación. La idea que persigue el desarrollo de este trabajo es ¿por qué no extender las aplicaciones web que actualmente se desenvuelven en el mundo desktop a cualquier tipo de dispositivo? Hace algunos años esta idea existía pero no era viable debido a las pocas capacidades de los dispositivos y la carencia de un estándar consolidado que aportara muchas de las funcionalidades que hoy nos provee HTML5, JavaScript y CSS3.

El enfoque propuesto persigue esta idea alternativa a las aplicaciones nativas, y hace hincapié en el aprovechamiento de la evolución constante del estándar HTML5 y las tecnologías web actuales sacrificando algunas características nativas (cada vez menores gracias al avance de HTML5) en pos de la mantenibilidad, escalabilidad y reusabilidad.

Algunas personas pueden pensar que las aplicaciones web móviles no son tan atractivas como sus contrapartes nativas para Android o iOS. Sin embargo, la mayor ventaja de las

aplicaciones web móviles es que pueden ser construidas independientemente de la plataforma y sin la necesidad de toolkits y SDKs específicos.

1.4 Alcance del Trabajo

En esta sección se desea resaltar las consideraciones principales respecto al alcance del trabajo.

El alcance del presente trabajo radica en:

- Los adaptabilidad de las aplicaciones web a múltiples dispositivos usando técnicas de Responsive Web Design (RWD) [2]: Aquí se hace referencia a los cambios en la interfaz y experiencia de usuario (UI & UX) que la aplicación deberá experimentar al momento de ejecutarse en ambientes diferentes (presentación del contenido, acciones visible, soporte para uso táctil, etc.).
- Construir un prototipo fundado en las tecnologías web. Se utilizará el lenguaje de programación JavaScript tanto en la implementación del Frontend como en el Backend. Asimismo se emplearán plataformas, frameworks y librerías existentes tales como Node.js, RequireJS, Backbone.js, Handlebars, etc. Todas estas tecnologías permitirán el desarrollo del prototipo “End to End” en un solo lenguaje de amplio dominio y uso muy difundido.
- Evaluar el uso de RWD y contrastar la creación de aplicaciones web con esta técnica frente a las aplicaciones nativa e híbridas.

Queda fuera de alcance de este trabajo la adaptabilidad de la aplicación según perfiles de usuario y presentación de contenido o recomendaciones en base al uso o las preferencias de la aplicación.

1.5 Organización del trabajo final

El trabajo se organiza comenzando con la introducción actual que exhibe principalmente la motivación del trabajo y alcance del mismo.

En el segundo capítulo se comenzará investigando el estado del arte de las aplicaciones web, su incorporación al ecosistema de dispositivos móviles y la competencia con las aplicaciones nativas. Por otro lado se analizará la evolución hacia el estándar HTML5 y las nuevas características que aporta para la creación de Webapps cada vez más asombrosas.

El tercer capítulo introducirá y profundizará los conceptos de Mobile First [3] y Responsive Web Design como solución a los problemas de adaptabilidad a múltiples dispositivos que enfrentan las aplicaciones web. Asimismo se expondrán distintos frameworks que implementan esta técnica.

El cuarto capítulo del trabajo se centrará en la definición y diseño de un prototipo – como prueba de concepto de una aplicación web con las características desarrolladas en el capítulo anterior. Se especificarán los requerimientos funcionales y no funcionales teniendo en consideración la adaptabilidad a múltiples dispositivos. Así como también requerimientos que hacen a una aplicación web avanzada relacionados a notificaciones en tiempo real, renderización en el cliente o el servidor, etc.

El quinto capítulo especificará los frameworks, librerías y tecnologías usadas en el Backend y Frontend del prototipo. Además se expondrán los módulos principales y las dificultades encontradas durante el diseño y la implementación. Finalmente se detallarán aspectos relacionados a los ambientes de desarrollo y despliegue.

El capítulo final expone las conclusiones respecto a los resultados obtenidos.

2 Aplicaciones web: Estado del arte

En este capítulo se presenta el estado del arte de las Webapps y las Webapps móviles, además se desarrollan las novedades incorporadas por el estándar HTML5 y cómo pueden contribuir a la implementación de una aplicación web avanzada.

La sección 2.1 hace una breve revisión del concepto de aplicación web y su evolución. También plantea sus ventajas y desventajas respecto a las aplicaciones tradicionales. En la sección 2.2 se expone la propagación de las aplicaciones web al ecosistema de dispositivos móviles y la complejidad incorporada por este nuevo ámbito de acción. La sección 2.3 presenta la competencia actual entre las aplicaciones móviles nativas y las Webapps, además resalta el concepto de aplicación híbrida como alternativa válida. Finalmente, la sección 2.4 refiere a la evolución y características más novedosas que HTML5 aporta al desarrollo de Webapps.

2.1 Evolución de las aplicaciones web

En los comienzos de internet la web consistía solo de sitios web (Websites) que eran esencialmente repositorios de información conteniendo documentos estáticos. Lo navegadores web o browsers fueron inventados en ese momento como un medio para recuperar y mostrar esos documentos en un formato adecuado. Además el flujo de información ocurría en un solo sentido, desde el servidor al browser. La mayoría de los sitios no seguían un proceso de autenticación porque no era necesario (a cada usuario se lo trataba de la misma forma y presentaba la misma información).

Hoy la web es casi irreconocible respecto de sus inicios. La mayoría de los sitios en la web son de hecho aplicaciones de software compejas. Estas son altamente funcionales y dependen para funcionar de un flujo de información bidireccional entre el cliente (browser) y el servidor. Permiten registración y login, transacciones financieras, búsquedas, creación de contenido por parte de los usuarios, etc. El contenido presentado es generado dinámicamente y casi siempre es dependiente del usuario que lo requiere. Mucha de la información procesada es privada y sensible por lo que la seguridad se torna en un tema que requiere consideración.

Una definición potencial de Webapp puede ser la siguiente:

"*Una aplicación web es una aplicación que es accedida por usuarios sobre una red tal como Internet*" [4].

La pregunta que podría surgir es, ¿una aplicación web es un sitio web?

Tal como se precisa en [5] el primer paso para diferenciar una aplicación web de un sitio web es enfocarse en la palabra "aplicación". Una aplicación es (entre otras cosas) un programa computacional diseñado para resolver una tarea específica.

El término aplicación web está relacionado al surgimiento del término Web 2.0. Esta expresión fue introducida en 1999 para describir sitios web que usan tecnologías más allá de las usadas por los primeros sitios web creados.

Un sitio Web 2.0 permite a los usuarios interactuar y colaborar entre sí como creadores de contenido en una comunidad virtual, a diferencia de los sitios web tradicionales donde los usuarios se limitan a la observación pasiva del contenido. Algunos ejemplos de la Web 2.0 son las redes sociales, blogs, wikis, etc. [6]

Más allá de esta definición, Web 2.0 tiene un significado más amplio, en palabras de su autor:

"*Web 2.0 es la revolución empresarial en la industria de la computación causada por el uso de Internet como una plataforma, y un intento de entender las reglas para el éxito en esta nueva plataforma. La principal regla entre todas es: Construir aplicaciones que aprovechen los efectos de la red para mejorar a medida que más gente las usa.* "

—Tim O'Reilly

El propósito fundamental de todas las aplicaciones web es facilitar la completitud de una o más tareas. A diferencia de los usuarios que acostumbran a acceder a Websites centrados en contenido (solo lectura) los usuarios de las aplicaciones web acceden con metas y tareas específicas en mente.

Una de las implicancias más importantes de su orientación basada en tareas es el grado en que la aplicación debería llamar la atención hacia sí misma. Los diseñadores tienen que ser lo suficientemente astutos para: no desviar la atención del contenido mostrado en

la aplicación pero, al mismo tiempo deben resaltar acciones y sucesos para brindar una experiencia de usuario agradable y transparente.

Otra de las diferencias importantes de una aplicación web frente a un Website es que las primeras deben notificar al usuario del estado de las tareas que están llevando a cabo mientras que los Websites no enfrentan este tipo de problemas.

En una aplicación web (Webapp) se pueden identificar las siguientes características:

- Más herramienta que libro: su naturaleza está orientada en las acciones que un usuario puede llevar acabo más que en la información que se le presenta.
- Relación uno a uno: establecen una sesión única con cada uno de los visitantes
- Capacidad para persistir/cambiar datos: permiten a los usuarios crear, manipular y almacenar datos.

Otra propiedad que se deduce de las anteriores es que las Webapps poseen cierta lógica de aplicación de la que carecen los Websites estáticos. Principalmente, requerida para lograr avanzados niveles de interacción. En esencia intentan parecerse y comportarse de forma similar a las aplicaciones desktop (ver Figura 2-1).

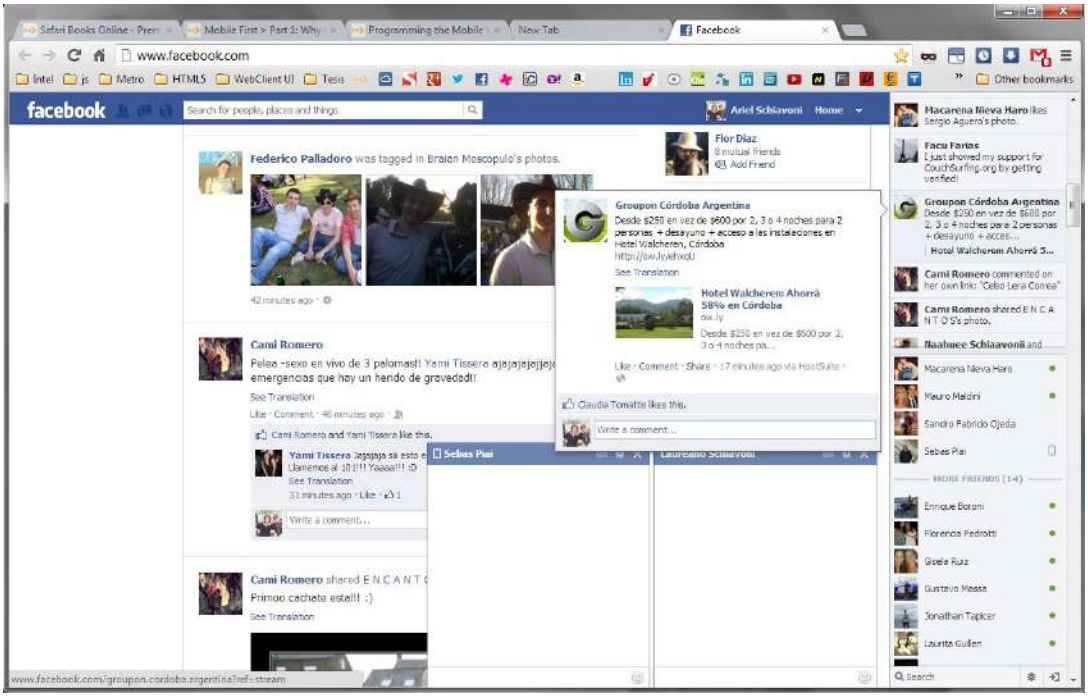


Figura 2-1: Webapp de Facebook ejecutándose en Chrome. El nivel de interacción que presenta es comparable a cualquier aplicación desktop.

Las aplicaciones web se han expandido para realizar cualquier tarea que puede ser implementada on-line tanto a nivel público como intra organizacional. Algunos ejemplos:

- Redes sociales: Facebook, Twitter, Linkedin.
- Shopping: Amazon, eBay, Mercadolibre, Walmart
- Banking: Santander, Banelco.
- Buscadores y clientes de email: Google, Yahoo.
- Apuestas: Bwin, Bet365.
- Herramientas Organizacionales: Jira, Bamboo, Wiki.

Numerosas aplicaciones preexistentes a la era de las aplicaciones web fueron migradas a esta tecnología y la tendencia continua en alza.

Las ventajas de las Webapps surgen de la concentración de factores técnicos sumados a los incentivos comerciales derivados de la forma en que la sociedad se relaciona mediante internet.

- HTTP, el protocolo usado por la WWW se centró en ser liviano y sin estado. Esto proporcionó capacidad de recuperación en caso de errores de comunicación y también evita que el servidor mantenga una conexión abierta para cada usuario como existía previamente
- Cada usuario cuenta con un browser ya instalado en su computadora en el cual la aplicación web se ejecuta. Así se evita la necesidad de distribuir y manejar instaladores o actualizaciones por parte de los desarrolladores. Asimismo los usuarios no necesitan llevar a cabo tediosos procesos de instalación o actualización.
- Los browsers son muy avanzados, permitiendo la construcción de aplicaciones con interfaces de usuario atractivas y con un alto grado de interactividad gracias a la ejecución de JavaScript en el cliente.
- Las tecnologías y lenguajes utilizados para desarrollar Webapps son relativamente simples, además existe un sinfín de herramientas de desarrollo y proyectos open source que facilitan enormemente la tarea del desarrollador.
- Acceso desde cualquier lugar con Internet.
- Los datos necesarios para el funcionamiento de la aplicación son almacenados remotamente por lo cual se requiere menos espacio en disco y los requisitos de sistema son menores.

Desde el punto de vista de las desventajas o limitaciones se pueden mencionar:

- Están desarrolladas para ejecutarse en el contexto de un browser en lugar del sistema operativo directamente. Esta característica hace que se encuentren limitadas por las capacidades que el browser expone (ver Figura 2-2 y Figura 2-3).
- Requieren acceso a internet para funcionar (esta desventaja comienza a desaparecer, HTML5 posibilita la construcción de aplicaciones web que funcionan offline)
- Los servidores acaparan demasiada información sensible del usuario que puede ser comprometida.
- Desde el punto de vista del desarrollador el mismo debe manejar un abanico de nuevas tecnologías o lenguajes que evolucionan rápidamente – JavaScript, CSS3, HTML5 – a diferencia de las ya bien preestablecidas – Java, C++ –.

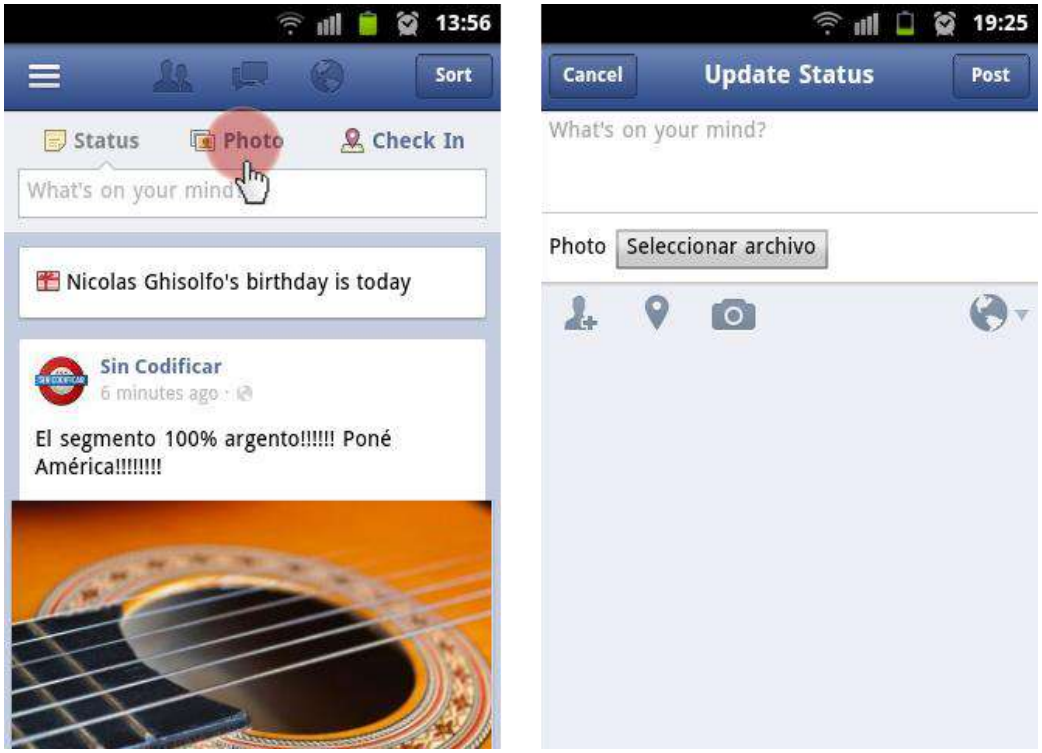


Figura 2-2: Webapp de Facebook ejecutándose en el browser de un dispositivo mobile Android. Se puede observar la imposibilidad para acceder a la cámara de fotos. En su lugar existe un control para cargar archivos desde un medio de almacenamiento.

Figura 2-3: Aplicación nativa Android. Luce muy similar a la Webapp pero permite acceso a mayores características del dispositivo como la agenda telefónica o la cámara en este caso.

2.2 Expansión de las aplicaciones web al ecosistema de dispositivos móviles

Antes de comenzar es conveniente resaltar las características que convierten a un dispositivo en “móvil”.

- **Portable**: se puede llevar a cualquier lugar en cualquier momento sin ninguna consideración especial (gimnasio, trabajo, universidad, etc.).
- **Personal**: es absolutamente personal, cada cual elige su ringtone, aplicaciones, tema visual, etc.
- **Acompañante**: esta con el usuario todo el tiempo (incluso en el baño). Una persona puede olvidar varias cosas al salir en la mañana pero de seguro nunca olvidará la billetera, las llaves y su dispositivo móvil.
- **Fácil de usar**: Una laptop es portable, puede llevarse a cualquier lugar y tiene conexión de red. Sin embargo si se tiene que usar es necesario sentarse y quizás encontrar una mesa para empezar. Un dispositivo móvil tiene que ser fácil y rápido de usar. El usuario no quiere esperar a que inicie el sistema operativo y tener que sentarse. Si está caminando por la ciudad solo quiere ser capaz de averiguar cuando sale el próximo tren o donde se encuentra una parada de colectivo.

- **Conectado**: tienen que tener la capacidad de conectarse a internet en todo momento ya sea por redes Wi-fi o 3G/4G. En este sentido las notebooks o el iPod por ejemplo quedan fuera de la clasificación.

2.2.1 Importancia de las aplicaciones web móviles

La web se está convirtiendo en la principal plataforma de desarrollo y fuente de información y software. Hasta hace pocos años las aplicaciones web eran propiedad del mundo desktop y contar con este tipo de aplicaciones en los dispositivos móviles era un requisito opcional debido a su escasa preponderancia. Sin embargo esta tendencia ha ido virando y actualmente representa una clara obligación. Las aplicaciones web se vuelven cada día más complejas e interesantes y están reemplazando a gran ritmo a las aplicaciones desktop o nativas que los usuarios acostumbraban a utilizar con mayor frecuencia.

A medida que las aplicaciones web se fueron expandiendo hacia los dispositivos móviles, uno de los mitos que surgió es “la gente no usa los browsers de los dispositivos móviles”. Al principio este enunciado tenía un alto grado de veracidad, los primeros dispositivos contaban con un browser muy básico y las capacidades de procesamiento y red eran muy limitadas. Sumado al hecho de que la experiencia de usuario que brindaban las aplicaciones web en un dispositivo móvil era muy mala. Sin más, se puede ver como lucía Facebook en sus primeras versiones en el browser de un Smartphone (ver Figura 2-4)

Figura 2-4: Aplicación web de Facebook accedida desde un dispositivo móvil en 2010.

Con el correr del tiempo el avance a nivel software (mejores sistemas operativos, browsers, aplicaciones, etc.), hardware (mayor capacidad de procesamiento, pantallas de mayores resoluciones y calidad) y también de las velocidades de conexión disponibles han cambiado en gran medida este panorama (ver Figura 2-5). Hoy en día según las estadísticas provistas por [7] y [8], a comienzos de 2012 había en el mundo:

- 2.405.510.036 conexiones de internet (34.3% de la población).
- 6.000.000.000 suscripciones a servicios de telefonía celular (86% de la población).

Además según estudios desarrollados por [9] se pone de manifiesto que el 10% del tráfico web proviene dedispositivos móviles y está creciendo a tasas muy importantes (ver Figura 2-6).

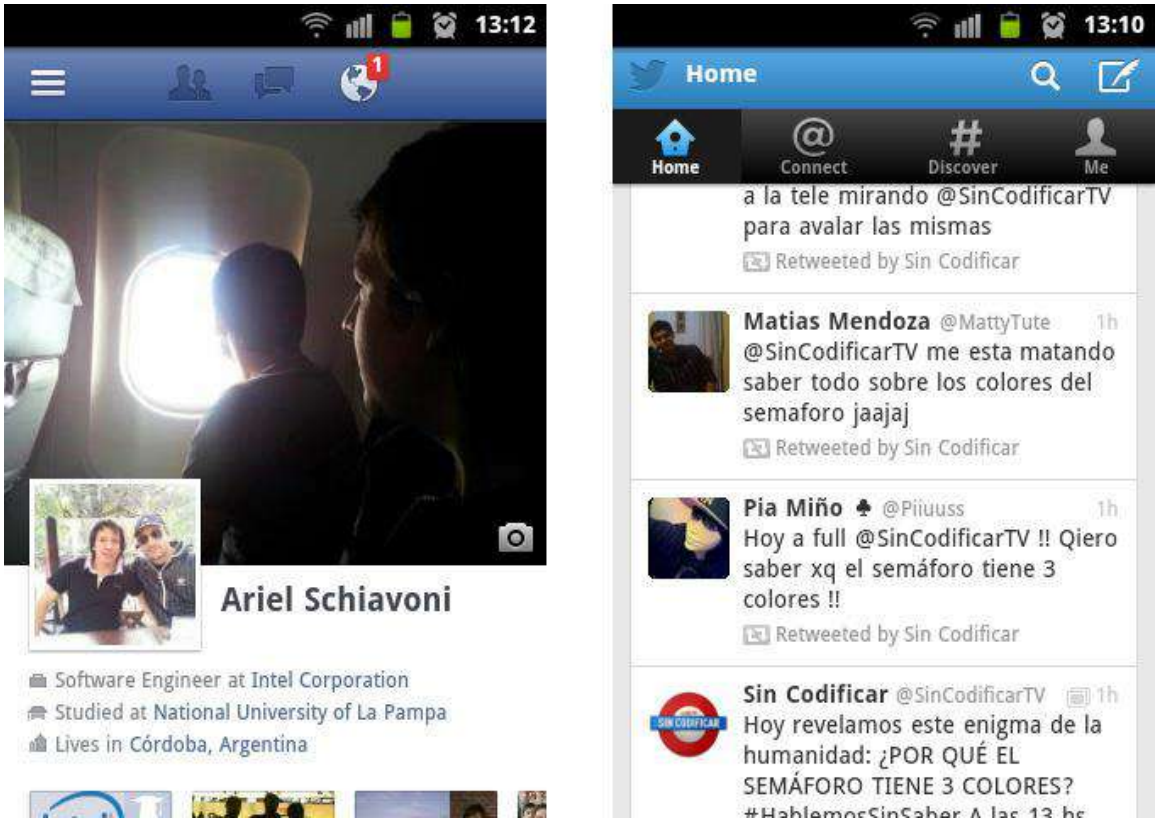


Figura 2-5: Aplicación web de Facebook y Twitter accedida desde un dispositivo móvil en 2012.

Mobile share of web traffic			
	2010	2012	Increase 2010-2012
Africa	5.81%	14.85%	155.59%
Asia	**6.1%**	**17.84%**	**192.46%**
Europe	1.81%	5.13%	183.43%
North America	4.71%	7.96%	69.00%
Oceania	2.88%	7.55%	162.15%
South America	1.46%	2.86%	95.89%
Worldwide	3.81%	10.01%	162.73%

Figura 2-6: Crecimiento de la cuota de tráfico web proveniente de dispositivos web móviles.

Como se observa en las estadísticas anteriores el uso de la web desde los dispositivos móviles esta “explotando” y pronto superará al de las PCs. Frente a este hecho de la realidad es necesario desarrollar una estrategia mobile con el objetivo de acaparar la mayor cantidad de usuarios para el negocio o no perderlos. Un artículo que claramente demuestra esta tendencia es “*Why Mobile Matters*” de Luke Wroblewski [10]. En el mismo se hace una comparación entre el número de nacimientos mundiales que se producen por día y la cantidad de dispositivos móviles (smartphones y tablets) que venden las compañías. Los resultados arrojan que el total de dispositivos móviles que ingresan al mundo por día es aproximadamente 1.45 millones frente a los cerca de 320.000 nacimientos diarios.

Semejante masa de dispositivos móviles ingresando en el mercado condujo al cambio de tendencias preestablecidas en el mercado de la computación que casi siempre fue dominado por la alianza WINTEL. Tanto Apple con los dispositivos iPhone/iPad como Google con sus Androids empezaron a sustraer un cuota importante en el mercado de la computación en los últimos años (ver Figura 2-7).

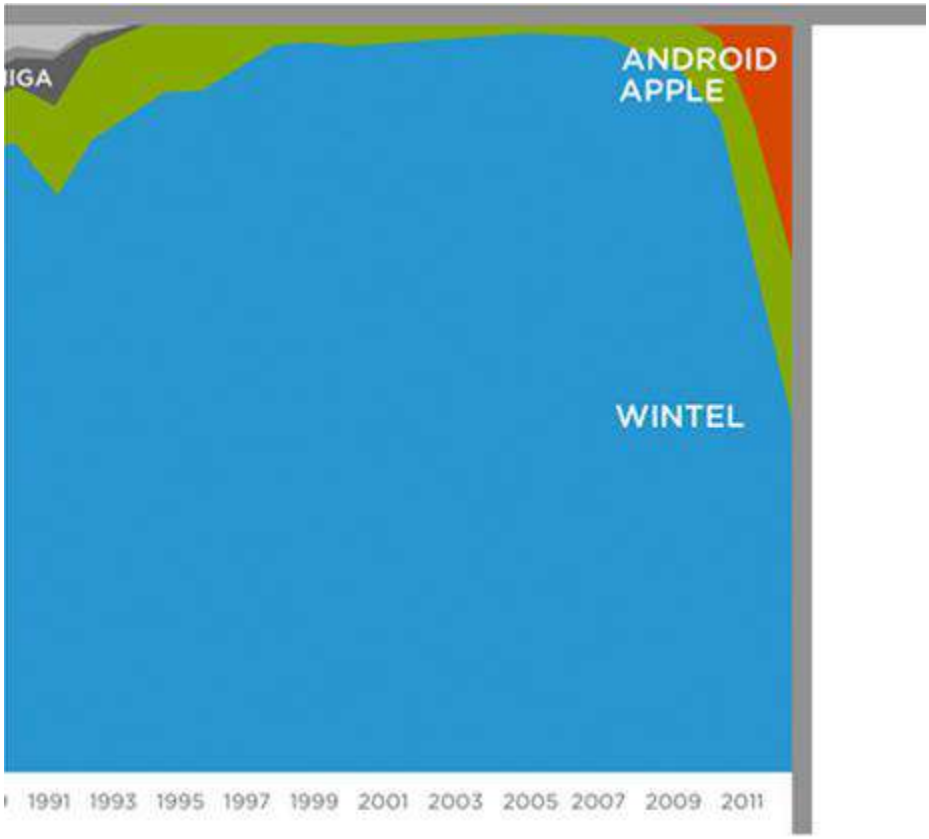


Figura 2-7: Participación en el mercado de la computación.

A medida que los dispositivos móviles van ganando terreno en el mercado de la computación surgen muchas oportunidades para las empresas de software y servicios. Considérese el caso de los pagos mediante dispositivos móviles en PayPal que pasaron de 141 millones de dólares en 2009 a 4.000 millones hacia finales de 2011 [11]. De forma similar las ventas de eBay pasaron de 600 millones de dólares a 5.000 millones en el mismo periodo [12].

2.2.2 Complejidad para alcanzar adaptabilidad a múltiples dispositivos

Cada dispositivo que surge suma nuevas formas de interacción que va cambiando la experiencia de usuario. La idea de adaptabilidad que se plantea es dotar a la misma aplicación con la capacidad de brindar la mejor experiencia de usuario según el dispositivo o plataforma en que se ejecuta. En el transcurso del trabajo se pondrá énfasis en el desarrollo de aplicaciones web móviles y las estrategias a seguir para incluir la mayor cantidad de dispositivos posibles.

La integración de las aplicaciones web al ecosistema mobile también contribuyó una gran cuota de complejidad en el desarrollo web. La misma puede desglosarse en dos factores.

- La complejidad intrínseca de la aplicación que se quiere desarrollar: aumenta de forma constante, por un lado con el surgimiento de dispositivos que ofrecen nuevas funcionalidades tales como acceso a la cámara fotográfica, GPS, acelerómetro, etc. Por otro lado se requieren novedosas features relacionadas a notificaciones en tiempo real, integración a redes sociales, funcionamiento offline, etc.
- La complejidad "multi-device": las aplicaciones web ya no forman parte de un único dispositivo y plataforma sino que se expanden a muchos ámbitos de la vida en el uso de la tecnología; desde tablets y smartphones a televisores, electrodomésticos y otros. Cada nuevo tipo de dispositivo representa una nueva forma de interacción que debe ser tenida en cuenta.

A medida que la fisonomía de la web se vuelve más compleja, es muy importante entregar una experiencia web sólida a un creciente número de contextos. Afortunadamente, fueron surgiendo nuevas técnicas como RWD que dan a los desarrolladores de Webapps, herramientas para construir layouts sensibles a cualquier tamaño de pantalla. Al mismo tiempo se usan "fluid grids, flexible images y media queries" para dotarlo de una destacada apariencia a pesar de las diferentes dimensiones de pantalla de los dispositivos.

El contexto móvil es mucho más que solo el tamaño de la pantalla. Los dispositivos móviles están presentes en todos lados, desbloqueando nuevos casos de uso. Debido a que se vive apegado a estos dispositivos, la conectividad es un tema importante a considerar, fluctuando entre señales Wi-fi muy buenas cuando alguien se halla en su casa a 3G/EDGE cuando sale. Asimismo las pantallas táctiles abren nuevas oportunidades para interactuar con el contenido mediante gestos, los cuales son puestos en consideración al momento de diseñar la estructura y la funcionalidad de una aplicación web.

Con el fin de crear una aplicación web que esté verdaderamente diseñada para funcionar en un contexto móvil y no sólo para pantallas pequeñas, se desea asegurar que se están atacando los principales desafíos del desarrollo móvil. Las restricciones en este contexto

fuerzan a enfocarse en definir qué contenido es esencial y cómo presentarlo tan pronto como sea posible.

2.3 Desarrollo de aplicaciones móviles nativas, web e híbridas

Muchas organizaciones que comienzan a dar sus primeros pasos para implementar una estrategia “móvil” enfrentan una difícil decisión que influirá en los resultados de sus iniciativas. El proceso de elegir una propuesta de desarrollo para una aplicación móvil, ya sea nativa, web o híbrida implica varios parámetros tales como: presupuesto, tiempos del proyecto, público objetivo y funcionalidades de la aplicación que se intenta construir. Cada enfoque conlleva sus propias ventajas y limitaciones, y encontrar el que mejor se adapta a las necesidades de la organización puede ser una tarea desafiante.

2.3.1 Aplicaciones Nativas

Las aplicaciones nativas se componen de archivos binarios ejecutables que se descargan directamente al dispositivo y se almacenan localmente. La forma más habitual de descargar una aplicación nativa es a través de los stores o tiendas de aplicaciones tales como Apple App’s Store, Android’s Marketplace o BlackBerry’s App World.

El proceso de instalación es lanzado por el usuario, una vez finalizado el usuario ejecuta la aplicación como cualquier otra. Durante la inicialización la aplicación nativa interactúa directamente con el SO móvil sin ningún intermediario o contenedor. La aplicación nativa es libre de acceder a todas las APIs que el SO pone a disposición y, en muchos casos, tienen funcionalidades únicas provistas por el SO en cuestión.

Para crear una aplicación de estas, los desarrolladores deben escribir el código fuente y recursos adicionales tales como imágenes o segmentos de audio. Luego, usando las herramientas provistas por la plataforma, el código fuente es compilado para crear un archivo ejecutable en formato binario que posteriormente es empaquetado con el resto de los recursos para formar el archivo final distribuible (ver Figura 2-8).

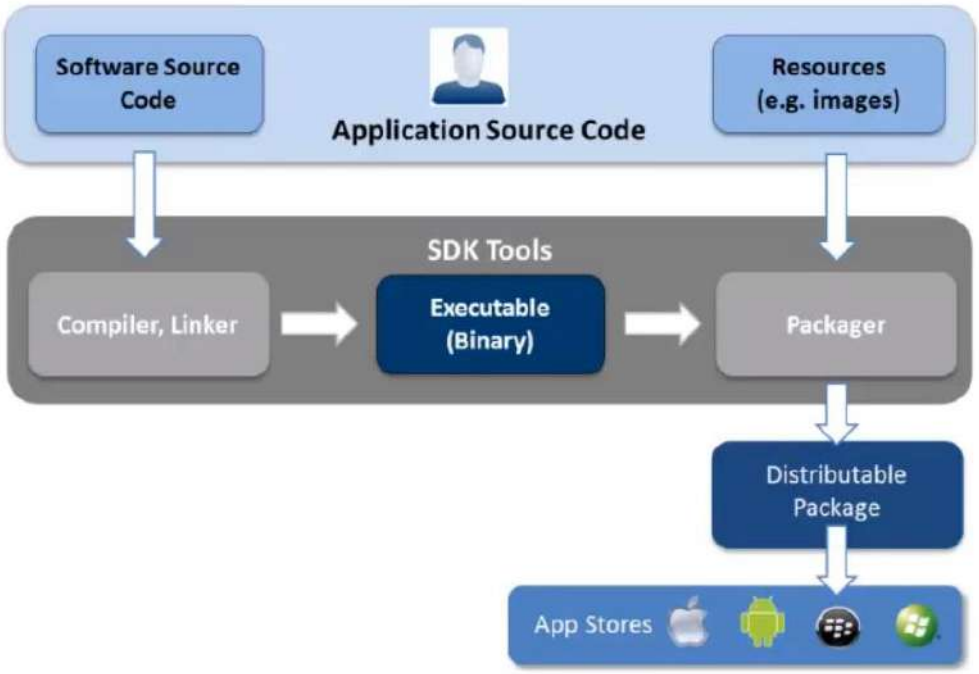


Figura 2-8: Desarrollo de aplicaciones nativas.

Las herramientas provistas por la plataforma así como otras utilidades y archivos es lo que se conoce como Software Development Kit (SDK) del SO mobile. Aunque el proceso de desarrollo es similar para los diferentes SO, el SDK es particular a la plataforma y cada SO mobile viene con sus herramientas o tools únicas (ver Tabla 1 y Figura 2-9).

Tabla 1: Características del desarrollo de aplicaciones móviles según el SO.

	Apple iOS	**Android**	**BlackBerry OS**	**Windows Phone**
Lenguaje	Objective-C, C, C++	Java (some C, C++)	Java	C#, VB.NET and more
Herramientas	Xcode	Android SDK	BB Java Eclipse Plug-in	Visual Studio, Windows Phone development tools
Formato del paquete	.app	.apk	.cod	.xap
Tienda de Aplicaciones	Apple App Store	Google Play	BlackBerry App World	Windows Phone Marketplace

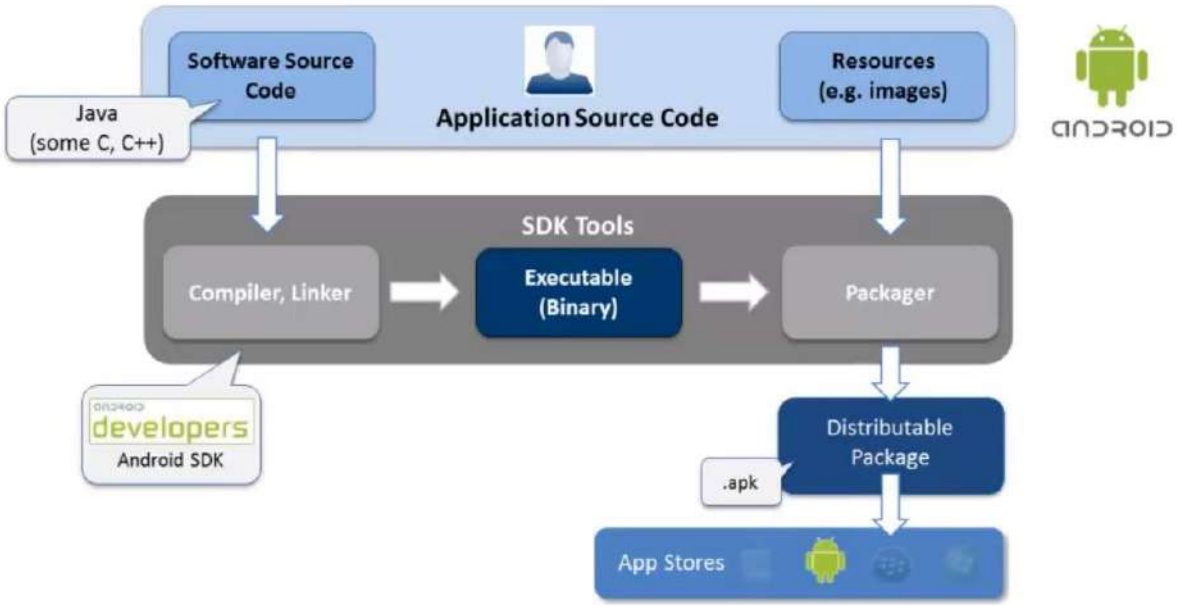


Figura 2-9: Desarrollo de aplicaciones Android.

Las diferencias entre plataformas que evidencia la tabla anterior es una de las desventajas más críticas que presenta el enfoque de desarrollo orientado a las aplicaciones nativas. El código escrito para una plataforma no puede ser usado por la otra, convirtiendo el desarrollo y mantenimiento de las aplicaciones nativas para múltiples plataformas en una tarea muy larga y costosa.

El principal beneficio de las aplicaciones nativas radica por un lado en la facilidad de búsqueda y monetización mediante la tienda de aplicaciones y por otro en la performance y acceso a todas las APIs que provee el SO. Estas APIs están divididas en:

- **APIs de Bajo Nivel**: Proveen servicios de acceso al hardware del dispositivo como cámara, GPS, micrófono, parlantes, almacenamiento, aceleración gráfica, etc.
- **APIs de Alto Nivel**: Proveen servicios tales como browser, calendario, agenda, manejo de llamadas, mensajes, emails, etc. Estas APIs son utilizadas por las aplicaciones nativas que suelen venir por defecto en los SO móvil pero también pueden ser accedidas por nuestra propia aplicación.
- **GUI Toolkit**: Provee un conjunto de componentes de interfaz de usuario tales como buttons, sliders, input fields, menus, tabs, dialogs, etc. Estos componentes están optimizados para ofrecer una experiencia de usuario fluida y placentera.

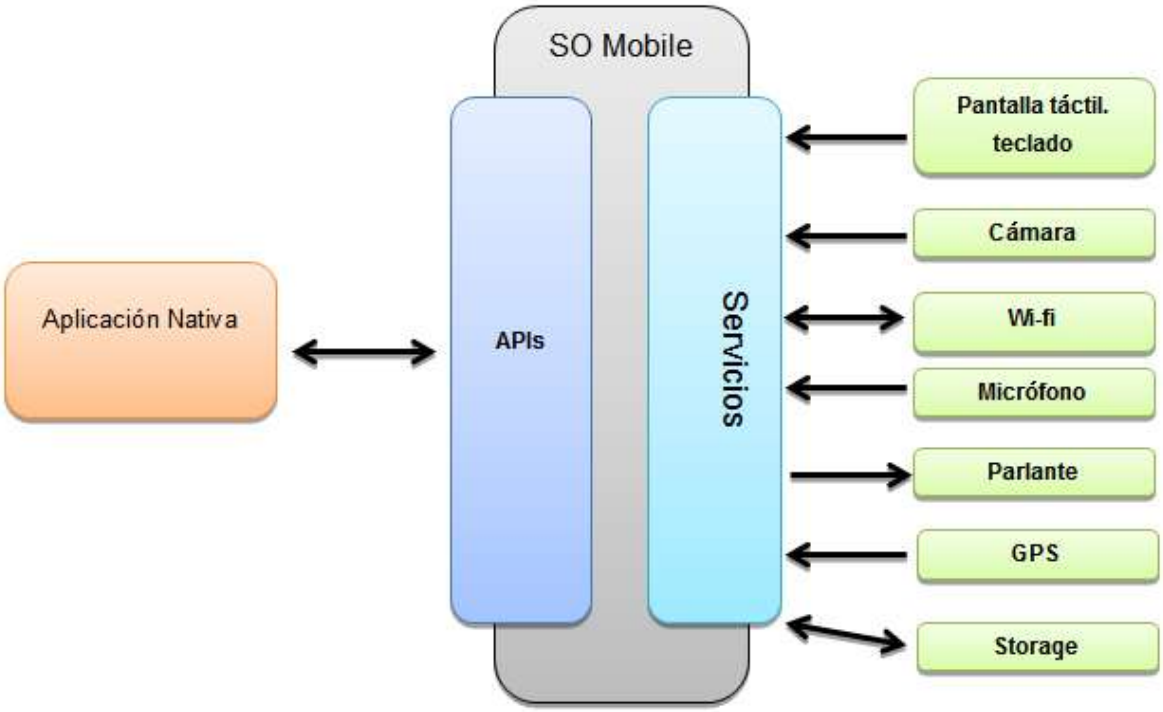


Figura 2-10: Interacción de una aplicación nativa con el dispositivo.

2.3.2 Aplicaciones Web

Los dispositivos móviles actuales cuentan con browsers muy potentes que soportan varias de las nuevas capacidades de HTML5, CSS3 y JavaScript avanzado. Con los recientes avances en este frente, HTML5 marca la transición de esta tecnología desde un simple lenguaje para la "definición de páginas" a un completo estándar de desarrollo para la creación aplicaciones basadas en el browser.

Algunos ejemplos que demuestran el potencial de HTML5 son:

- Avanzados componentes de interfaz de usuario
- Acceso a contenidos audiovisuales
- Servicios de geolocalización
- Disponibilidad offline

Usando estas características y muchas otras más que están en progreso, los desarrolladores son capaces de construir avanzadas aplicaciones usando solo tecnologías web.

Es importante distinguir entre dos enfoques extremos de aplicaciones web móviles. La primera dirección que las empresas tomaron fue la creación de sitios web optimizados para el funcionamiento en dispositivos móviles. Estos sitios son fácilmente reconocibles cuando son accedidos por un smartphone, proveen páginas HTML completas que han sido diseñadas para proveer una experiencia “touch” placentera sobre pantallas pequeñas.

En la Figura 2-11 y la Figura 2-12 se observan claramente las optimizaciones en la versión mobile. La tipografía y los iconos son de mayor tamaño, solo se muestran tres noticias destacadas y luego se agrega un botón de “ver más”. Así se disminuye notablemente el tiempo de carga de la página completa. Además se agregan iconos fácilmente accesibles y operables para el aprovechamiento de la pantalla táctil.

Figura 2-11: Aplicación web del diario Infobae accedida desde un Smartphone en su versión para móviles.

Figura 2-12: Aplicación web del diario Infobae accedida desde un Smartphone en su versión desktop.

En el segundo enfoque algunas empresas fueron aún más allá en aspectos de mejora sobre la experiencia de usuario creando aplicaciones web que lucen como aplicaciones nativas y pueden ser ejecutadas desde un acceso directo casi indistinguible del resto de las aplicaciones (ver Figura 2-13). Unos de los conceptos fuertemente impulsados en esta metodología es “Single Page Application” (SPA). En una SPA todo el código necesario – HTML, JavaScript y CSS – es recuperado con una única carga de página. Las acciones ejecutadas por los usuarios generan peticiones AJAX al servidor, el cual puede retornar pequeños fragmentos de HTML o solamente datos en formato JSON o XML delegando la creación de la vista HTML al cliente. Los beneficios radican en que se evita la recarga completa de la página, existe menor tráfico de datos, mejor tiempo de respuesta, el usuario puede seguir interactuando con otras partes de la aplicación mientras se carga una determinada región, etc.

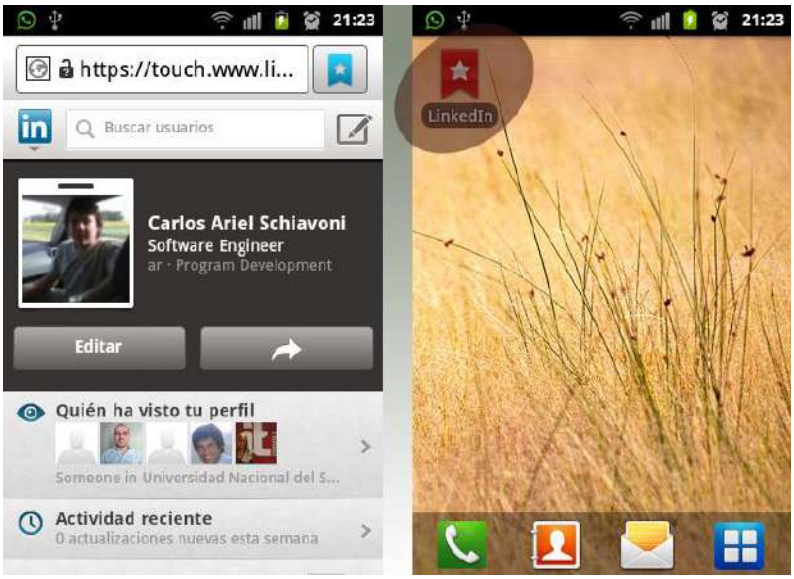


Figura 2-13: Aplicación Web de LinkedIn con acceso directo desde el escritorio de Android.

Una de las ventajas más prominentes de las aplicaciones web es su soporte multiplataforma y el bajo costo de desarrollo. La mayoría de los proveedores mobile utilizan el mismo motor de renderizado en sus browsers, Webkit –un proyecto open source liderado por Google y Apple que ofrece una de las implementaciones de HTML5 más completas y utilizadas por los browsers de la actualidad–. Debido a que el código de la aplicación es escrito en lenguajes web estándar que son compatibles con Webkit, Una sola aplicación proporciona una experiencia uniforme para los distintos dispositivos y sistemas operativos, por lo que es multiplataforma por defecto.

No obstante, a pesar del potencial y lo prometedor que parecen las tecnologías web todavía acarrean algunas limitaciones. A diferencia de las aplicaciones nativas que interactúan directamente con el SO, las Webapps se ejecutan dentro de un browser. El browser es en sí mismo una aplicación nativa que tiene acceso directo a las APIs del SO, pero sólo un conjunto limitado de estas APIs son expuestas a las Webapps que corren dentro de él. Mientras las aplicaciones nativas tienen acceso total al dispositivo, muchas características del mismo están parcialmente disponible para las Webapps o directamente son inaccesibles.

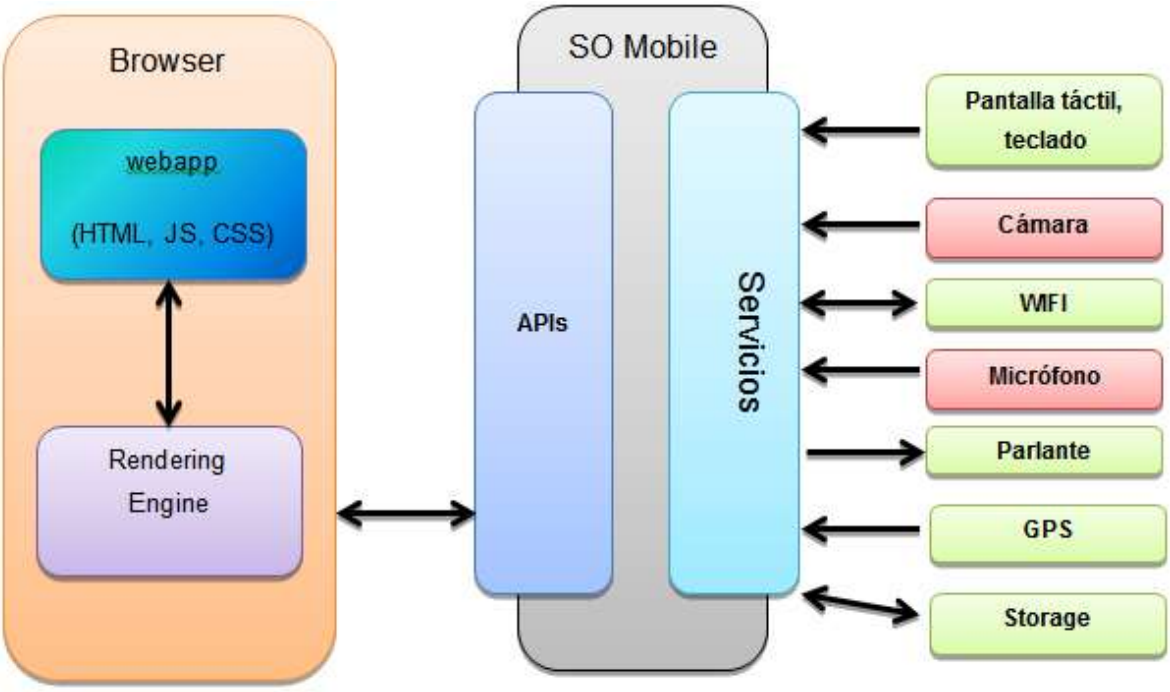


Figura 2-14: Interacción de una aplicación web con el dispositivo.

2.3.3 Aplicaciones Híbridas

Híbrido por definición es algo que deriva de orígenes heterogéneos, o está compuesto de elementos de diferentes clases. Una aplicación híbrida es escrita con las mismas tecnologías usadas por las aplicaciones web. Asimismo está alojada y se ejecuta dentro de un contenedor nativo sobre un dispositivo móvil. Esto representa el "casamiento" entre las tecnologías web y la ejecución nativa. Usando este enfoque, los desarrolladores escriben significantes porciones de código en tecnologías web multiplataforma al mismo tiempo que mantienen acceso directo a las APIs nativas cuando lo necesitan.

La porción nativa de la aplicación utiliza las APIs de SO y sirve como puente para exponer funcionalidades nativas que no son provistas naturalmente por el contenedor web o browser engine. Esta pequeña capa expone los servicios al contenedor web en un espacio de nombres accesible desde JavaScript, de esta forma facilita a las aplicaciones web contenidas el aprovechamiento de todas las características que los dispositivos modernos tienen para ofrecer.

Por otro lado los desarrolladores pueden optar por construir la porción nativa o aprovechar soluciones que han tenido éxito como PhoneGap [13] –librería open source que provee

APIs JavaScript uniformes para las capacidades comunes entre los diferentes sistemas operativos móviles.

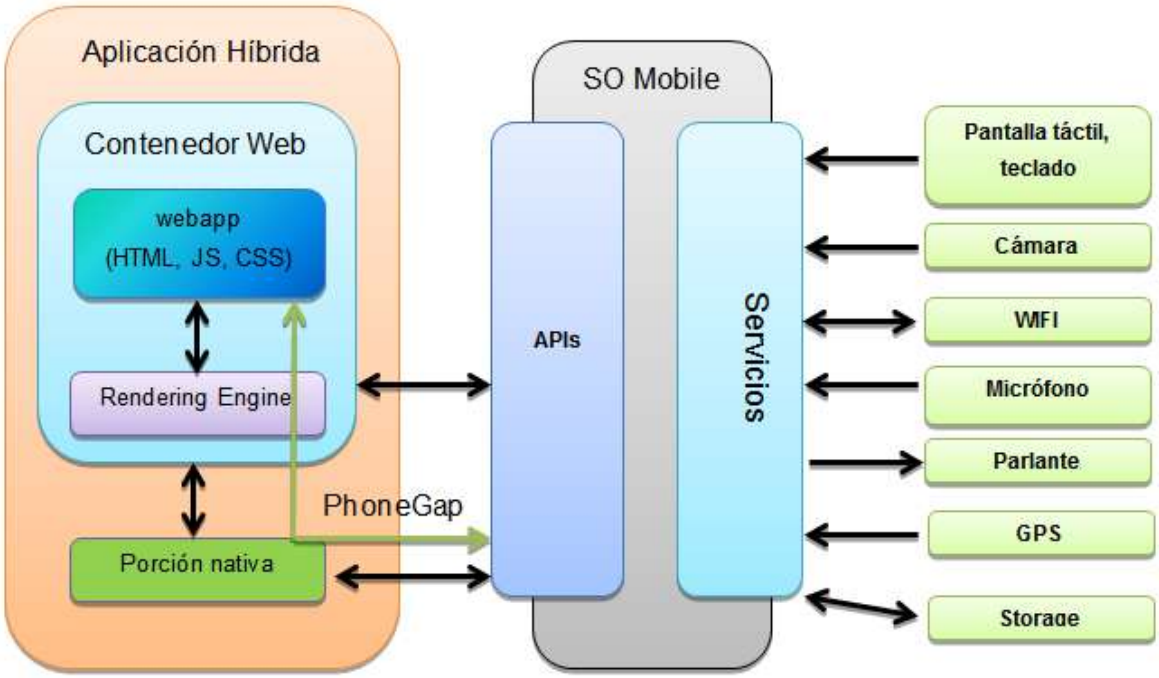


Figura 2-15: Interacción de una aplicación híbrida con el dispositivo.

Es sabido que las aplicaciones nativas son instaladas en el dispositivo mientras que las aplicaciones HTML5 residen en un servidor web. En el caso de las aplicaciones híbridas existen dos implementaciones posibles:

- Local: El código HTML JavaScript y CSS es empaquetado junto con el binario de la aplicación nativa. En este escenario se utilizan APIs REST para sincronizar datos entre el cloud y la aplicación.
- Server: El código de la aplicación web HTML5 reside en el servidor y opcionalmente se utilizan estrategias de cacheo en la aplicación nativa.

Un claro ejemplo de aplicación hibrida que ofrece las mejores características de ambos mundos es el de Netflix. Desarrolló una aplicación [14] que emplea la misma base de código para generar la interfaz de usuario en todos los dispositivos, basada en las tecnologías HTML5, posibilita por ejemplo el cambio “on the fly” de la interface de usuario en los servidores para implementar A/B testing y determinar las interacciones de usuario

óptimas. Así se evitan complicados procesos de actualización de la aplicación mediante el store. Por otro lado, la aplicación cuenta con porciones nativas a las cuales delega la decodificación y streaming de video que optimizan notablemente la performance de la aplicación.

2.3.4 Comparación

¿Aplicación Nativa o Web HTML5 o Híbrida? Depende. No existe una respuesta correcta que aplique a todas las situaciones.

En la Tabla 2 se muestra un resumen entre los diferentes enfoques al mismo tiempo que sirve como ayuda para la toma de decisión de la tecnología adecuada según el contexto de la organización.

Tabla 2: Comparación de los enfoques de desarrollo para aplicaciones móviles

	Nativa	**Web HTML5**	**Hibrida**
Conocimientos y Tecnologías necesarias	Objective-C, Java, C, C++, C#	HTML, CSS, JavaScript	HTML, CSS, JavaScript, C++, PhoneGap
Distribución	App Store	Internet	App Store
Velocidad de desarrollo	Baja	Alta	Media
Aplicaciones necesarias para cubrir las plataformas más importantes	4 (iOS, Android, Windows Phone, BlackBerry)	1	1
Mantenimiento	Alto	Bajo	Medio
Acceso a las capacidades ofrecidas por el dispositivo	Total	Parcial	Total
Acceso Offline	Si	Si	Si
Ventajas	Permite crear aplicaciones con interfaces de usuarios complejas y alto nivel de gráficos.	Ofrece desarrollo rápido, mantenimiento simple y portabilidad	Combina velocidad de desarrollo de la web con el acceso total a las capacidades del dispositivo y la forma de distribución de las aplicaciones nativas
Desventajas	Tiempo y costo de desarrollo. Mantenimiento. No portabilidad.	No puede manejar gráficos o animaciones pesadas. No puede acceder a	No puede manejar gráficos o animaciones pesadas. Requiere conocimiento

		todas la características del dispositivo (cámara, micrófono)	extra de frameworks (PhoneGap)
Mejores para	Juegos. Aplicaciones con requisitos de gráficos elevados.	Aplicaciones generales, sin elevados requisitos de gráficos y que no requieren acceso total al hardware del dispositivo.	Aplicaciones generales, sin elevados requisitos de gráficos. Aplicaciones que necesitan acceso total al hardware del dispositivo.

2.4 HTML5

HTML5 introduce varias características de vanguardia que permiten a los desarrolladores crear aplicaciones web con la funcionalidad, velocidad, rendimiento y experiencia de las aplicaciones desktop. Sin embargo, a diferencia de las aplicaciones desktop, las aplicaciones basadas en la plataforma web pueden llegar a una gama de dispositivos y público más amplio. HTML5 acelera el ritmo de las innovaciones y habilita la puesta en producción de los últimos cambios que el desarrollador incluya en su aplicación a todos los usuarios simultáneamente.

HTML5 no es una entidad única o tecnología monolítica. Es una colección de características, tecnologías y APIs que traen el poder de las aplicaciones desktop y la experiencia multimedia a la web. Además impulsa aún más los conceptos principales de interactividad y conectividad.

HTML5 incluye la quinta versión del lenguaje de marcado HTML, CSS3 y una serie de APIs JavaScript. Juntas estas tecnologías habilitan la creación de aplicaciones web que antes solo podrían haber sido implementadas para la plataforma desktop.

HTML5 no pertenece a una compañía o browser específico. Se ha forjado por una comunidad de personas interesadas en la evolución web y de un consorcio de líderes tecnológicos como Google, Apple, Mozilla, IBM y muchos más. Juntos están colaborando en la definición de estándares y llevando aún más lejos las capacidades del browsers. La próxima generación de aplicaciones web pueden ejecutar gráficos de alta complejidad, funcionar offline, almacenar grandes cantidades de datos en el cliente, hacer cálculos paralelizados, etc.

El primer “borrador” de HTML5 salió a luz en Enero de 2008 y sorprendentemente ya tenía un amplio soporte por parte de los browsers. Sin embargo, al día de hoy, aún no se encuentra completamente implementado y tampoco lo estará por algunos años más. Los dos grupos o comunidades que están a cargo de la evolución del estándar son la W3C y WHATWG (Web Hypertext Application Technology Working Group). WHATWG es un grupo integrado Apple, Google, Mozilla y otros, que surgió como respuesta al lento accionar de la W3C para evolucionar los estándares web.

Algunas de las reglas que estos grupos acordaron para el avance del estándar se resumen en:

- Las nuevas características deberían estar basadas en HTML, CSS, DOM y JavaScript.
- Reducir la necesidad de plugins externos como Flash.
- Mejor manejo de errores.
- Independencia respecto a los dispositivos.
- El proceso de desarrollo y avance del estándar debe ser público.

Las características más importantes que ofrece HTML5 se resumen en la siguiente tabla:

Tabla 3: Características de HTML5.

 **Semántica**	La semántica es una de las características más distintivas entre la plataforma web y otras plataformas de aplicaciones. La web es texto y el texto tiene un significado. Las aplicaciones web han sido creadas en un ecosistema donde el contenido basado en texto es “linkeable, searchable and mashable”. Estos beneficios no son gratis. Las herramientas automatizadas y robots que los search engines ejecutan para indexar la web deben reconocer el significado de lo que la aplicación web representa para arrojar cada vez mejores resultados. HTML5 introduce nuevos elementos y atributos que ayudan al desarrollador a mejorar la semántica del contenido que agrega a su aplicación web, y al mismo tiempo, facilita el trabajo de los agentes automatizados que deben reconocer el contenido. Entre las incorporaciones se encuentran: • Elementos estructurales: <header>, <footer>, <nav>, <aside>,

	<section>, <article>. • Elementos multimedia: <canvas>, <video>, <audio>. • Nuevos atributos: autofocus, data-*, placeholder. • Nuevos tipo de relaciones para links: external, license, help, tag, icon. • Nuevos tipos de input: color, url, email, tel, number, range.
 **Offline Storage**	Web y Online son dos términos íntimamente relacionados. Incluso sinónimos para muchas personas. HTML5 presenta una serie de alternativas para hacer realidad las aplicaciones web offline. • Application cache • localStorage • web SQL & indexed databases • online/offline events Estas tecnologías permiten el funcionamiento de la aplicación web con o sin conexión. La aplicación puede incluso descargar datos y almacenarlos localmente para funcionar cuando el usuario se mueva en regiones donde las condición de conectividad son deficientes. Asimismo, otras de las ventajas que ofrece, yace en la optimización de la performance que se alcanza al mantener datos cacheados localmente evitando la necesidad de hacer peticiones al servidor.
 **Acceso al Dispositivo**	HTML5 ofrece varias APIs que actualmente están en estado borrador o “working draft” que posibilitan el acceso a características del dispositivo tales como: • File Access: Esta API facilita la interacción con datos binarios y el sistema de archivos local (leer archivos asíncronamente, crear archivos temporales, drag&drop, lectura recursiva de directorios, etc.) • Geolocation: Es una API que permite acceder a información de localización del usuario con su previo consentimiento. La API es agnóstica del dispositivo y no tiene en cuenta los mecanismos subyacentes que el browser utiliza para obtener las coordenadas (GPS, Wi-Fi, consultándole al usuario que ingrese su ubicación,

	etc.)
 **Conectividad**	Las características que HTML5 trae caen en dos categorías: las que ponen a las aplicaciones web a la par con las aplicaciones nativas, y las que agregan una ventaja o plus. Lograr una mejor y más eficiente conectividad entre las aplicaciones web se traduce a aplicaciones que permiten colaboración en tiempo real como chats, juegos, compartir escritorio, etc. Las APIs que están en desarrollo son: • Web Sockets: Proveen canales de comunicación full-duplex con el servidor. El protocolo es muy eficiente por ser ligero comparado a HTTP. Además, una vez establecida la conexión, la comunicación puede ser iniciada tanto por el cliente como por el servidor y no es necesario utilizar técnicas ineficientes como "long polling" en el cliente para detectar alguna "notificación" que requiere la atención del usuario. • WebRTC: Habilita el soporte para video conferencias y streaming en vivo.
 **Multimedia**	<audio> y <video> son las incorporaciones más importantes que HTML5 aporta en esta sección. A través de las APIs, mediante JavaScript se puede acceder, controlar y manipular la reproducción de los archivos (agregar efectos de sonido y videos, subtítulos, etc.). Aunque el verdadero poder de estos nuevos elementos HTML se destaca cuando se combina con otras tecnologías web como Canvas, SVG, CSS3 o WebGL.
 **Gráficos y 3D**	La web ha sido siempre un medio visual restringido comparado a las aplicaciones nativas. Hasta hace poco tiempo los desarrolladores web estaba limitados a utilizar CSS o JavaScript para crear animaciones y efectos visuales o sino debían confiar en plugins como Flash. Con la incorporación de tecnologías como el elemento Canvas, WebGL, SVG este ya no es el caso. Ninguna de estas nuevas capacidades brindadas por HTML5 sería útil sino es capaz de ejecutarse de manera eficiente. Gracias al enorme avance de los engines JavaScript como V8, SpiderMonkey, JSCore la ejecución de JavaScript en el cliente es lo suficientemente rápida como para correr juegos 3D o manipular video en tiempo real.

 **Performance**	Las aplicaciones web de hoy empiezan a competir a nivel de performance con las aplicaciones nativas. Usando una variedad de técnicas y tecnologías las Webapps pueden ser notablemente optimizadas y responder ágilmente a las acciones del usuario. Las opciones que ofrece HTML5 son: • Cacheo de Datos: Uso de Application Cache o local Storage para mantener datos localmente y evitar peticiones remotas. • Web Workers: Permiten la ejecución de tareas JavaScript de manera paralelizada. • Web Sockets: Habilita eficientes canales de comunicación bidireccionales.
 **CSS3**	Gracias a la introducción de CSS3 es fácil aplicar estilos al HTML para crear interfaces de usuario de muy alta calidad. CSS3 por su parte también va adquiriendo nuevas características o extensiones tales como • Transiciones y Transformaciones 2D/3D • Web Fonts

Más allá de la enorme cantidad de funcionalidades que se van acoplando al compendio de tecnologías agrupadas bajo el nombre de HTML5, es importante resaltar que todos estos avances no están 100% implementados a lo largo de todos los browsers. En el camino para llegar a ser estándar y ser implementada por cada uno de los browsers, una determinada API pasa por los estados "borrador" → "recomendación" → "estándar". Sin embargo no es estrictamente necesario que sea estándar para que los distintos browser implementen tal funcionalidad. De hecho muchas de las APIs nombradas anteriormente se encuentras en estado de borrador o recomendación. Un sitio muy utilizado por los desarrolladores para determinar el alcance de una funcionalidad en los distintos browser y poder decidir su utilización es caniuse.com [15].

3 Adaptabilidad Web

Este capítulo tiene la intención de mostrar los avances en el campo de la adaptabilidad de las aplicaciones web que posteriormente serán aplicados en el desarrollo del prototipo. Se expondrá asimismo la estrategia “Mobile First” [3] vinculada a la metodología RWD. Juntas se emplean a la hora de planificar y desarrollar Webapps adaptables a distintos dispositivos.

3.1 Mobile First

La frase “Mobile First” es un término acuñado por Luke Wroblewski hace unos años y desencadenó un importante cambio en la forma en que se enfrenta la construcción de una aplicación. La idea planteada es que la experiencia móvil de una aplicación por lo general se comienza a construir una vez terminada la versión PC o desktop. No obstante, Luke plantea que hay razones para diseñar la versión móvil en primera instancia.

1. **Expansión:** El crecimiento de los dispositivos móviles es una tremenda oportunidad para llegar a más usuarios que nunca. Construir la experiencia móvil primero, asegura a las compañías el acceso a esta nueva y creciente masa de usuarios. (ver Figura 2-6)
2. **Enfoque:** Las restricciones del medio móvil hacen enfocar al desarrollador en qué es lo que realmente interesa. El contenido y los elementos mostrados tienen que ser priorizados. El resultado final es una experiencia de usuario enfocada en las tareas claves que el usuario desea lograr sin distorsionar la interfaz con elementos superfluos.
3. **Nuevas capacidades**: Las capacidades del medio móvil crean oportunidades para innovar. Los nuevos dispositivos ofrecen información de localización, orientación, entrada de datos mediante gestos táctiles, etc.

Por otra parte, el término puede expresar un pensamiento o idea diferente cuando es utilizado en diferentes contextos:

- **Cultural**: comúnmente “mobile first” en su sentido cultural hace referencia al hecho de que los dispositivos móviles son, para muchas personas (principalmente niños), el primer dispositivo conectado con el cual interactúan y se familiarizan.

- **Estrategia**: muchas empresas actualmente se encolumnan detrás de este nuevo slogan “mobile first”. Normalmente, esto significa hacer que el ambiente móvil sea una prioridad en lugar de una idea de último momento con el fin de capitalizar el crecimiento y la capacidad del medio.
- **Diseño**: El trabajo de los diseñadores se ha visto notablemente alterado. Los dispositivos móviles presentan restricciones que en el mundo desktop no existían. La más importante es la reducción del tamaño de pantalla que conlleva a menor espacio disponible para la ubicación de los elementos requeridos. El desplazamiento en la estrategia de diseño radica en concentrarse sólo en el contenido esencial primero y luego crear prototipos para revelar cómo el contenido principal escala en ambientes con mayores dimensiones.
- **Desarrollo**: El término “mobile first” en el contexto de desarrollo de Webapps está íntimamente relacionado y actúa en conjunto con RWD. La idea básica detrás de este tándem es que en lugar de crear una experiencia web orientada a PC con pantallas grandes y luego agregar código adicional para que funcione en pantallas pequeñas, tiene más sentido crear una base fundacional de código primero, que incluya las características mandatorias de la aplicación para los dispositivos de bajas capacidades,y luego mejorar progresivamente la experiencia de usuario a medida que los dispositivos se vuelvan más poderosos (ver Figura 3-1)

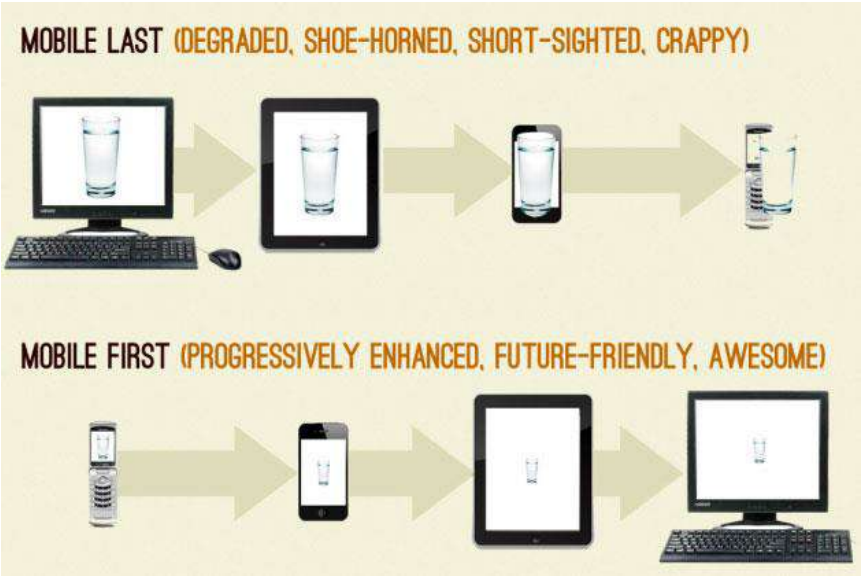


Figura 3-1: “Mobile First” vs “Mobile Last” presentado por Brad Frost en “The Many Faces of Mobile First” [16]

3.2 Responsive Web Design

De igual modo que el término “mobile first” hace unos años comenzó a mencionarse enérgicamente, la frase “responsive web design” ha adquirido bastante fama en el contexto de los desarrolladores y diseñadores web en el último tiempo. Esto se debe principalmente a que uno de los desafíos más importantes del diseño web en estos días es que las Webapps puedan ser visualizadas en un amplio rango de dispositivos que pueden ir desde un Smartphone, PC o Tablet hasta en una consola de juegos o Smart TV.

Cada cliente hoy en día desea tener la versión móvil de su Webapp, después de todo no quieren quedarse fuera de la enorme masa de usuarios que este ambiente representa. Se vuelve prácticamente esencial la creación de un diseño para BlackBerry, iPhone, iPad, Netbook y otros. Considerando además sus diferentes resoluciones de pantalla.

En el campo del diseño y desarrollo web es de suponer que la estrategia anterior se vuelve impracticable. Se torna imposible mantener un diseño para cada nuevo dispositivo y resolución que va surgiendo.

3.2.1 Concepto y ejemplo

RWD ofrece una solución más elegante, se basa en crear una única instancia de desarrollo que cuenta con la inteligencia y tecnología necesaria para responder a cambios en el tamaño de la ventana del browser o su orientación, ajustando dinámicamente las dimensiones de los elementos y su posicionamiento para lograr un mejor aprovechamiento del espacio disponible.

Un ejemplo de sitio web “responsive” que implementa RWD es el de la Figura 3-2.

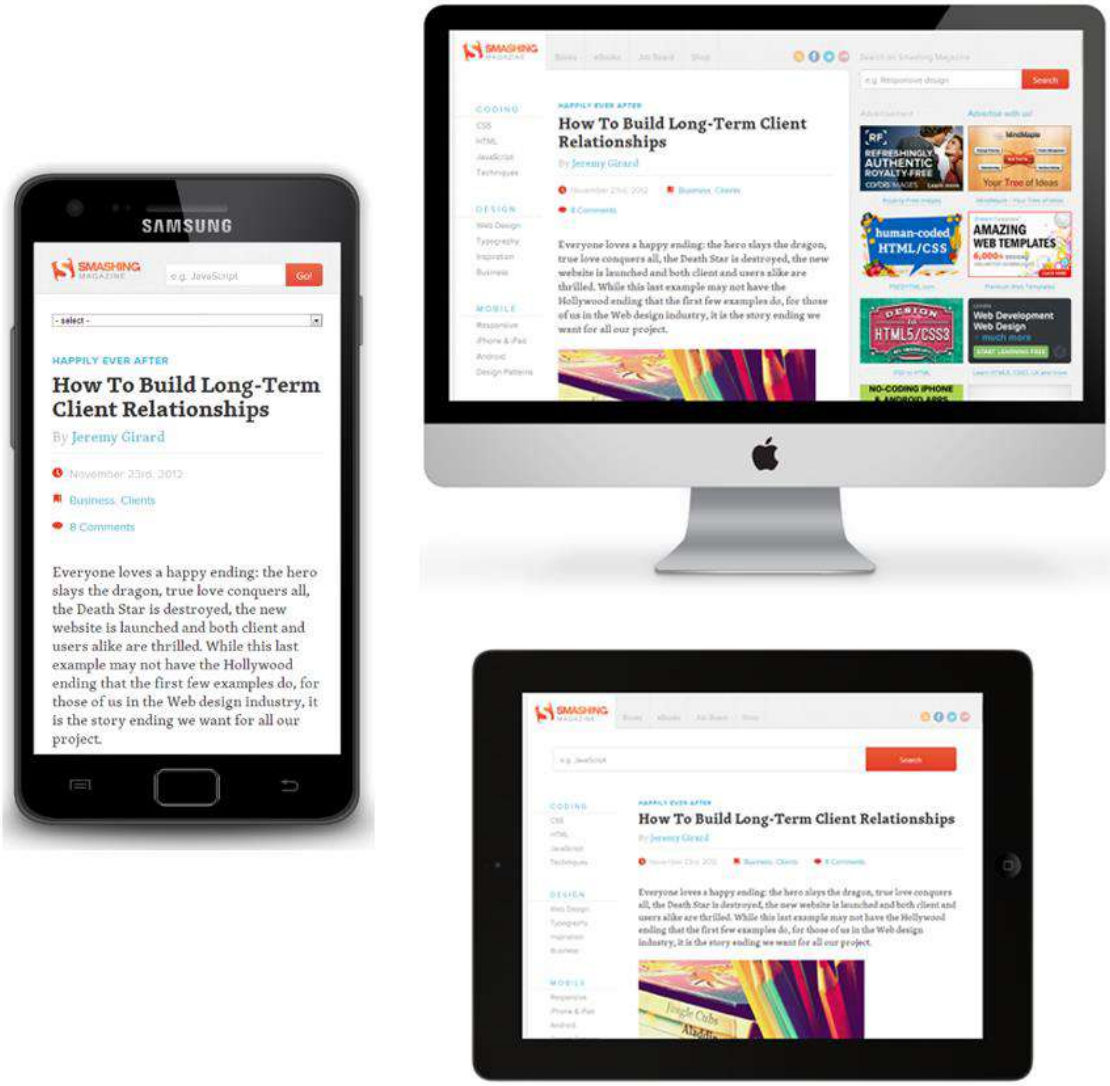


Figura 3-2: RWD a la vista en www.smashingmagazine.com

En este ejemplo se puede apreciar algo de la “magia” que RWD provee. Se observa claramente como el contenido se ajusta en relación al espacio disponible del dispositivo. Siguiendo el enfoque “mobile first”, se empieza por el Smartphone (Samsung Galaxy SII), el contenido esta reducido a una sola columna, tamaños de fuentes adecuados, contenido más relevante y acciones imprescindibles como “search” bien a la vista.

Continuando con el dispositivo Tablet (iPad) se observa que existe nuevo contenido, principalmente navegacional como el menú superior, una columna lateral con secciones y categorías, acciones para compartir, etc.

Por último la PC que cuenta con una resolución ampliamente superior y tiene la oportunidad de mostrar mayor contenido al usuario por lo que agrega otra columna lateral en la derecha con avisos publicitarios y links a recursos de interés.

3.2.2 Origen

El concepto RWD fue introducido por Ethan Marcotte en su artículo “Responsive Web Design” [17]. El origen del término está inspirado en “Responsive Architecture” [18] [19] una disciplina relativamente nueva que desafía el pensamiento tradicional afín al concepto de arquitectura[1].

Hay varios ejemplos para mencionar, salas que se redimensionan a sí mismas ajustándose según la cantidad de ocupantes; el desarrollo de una tecnología denominada “Smart Glass”, que oscurece los vidrios cuando se alcanza un determinado umbral de personas en una sala para ofrecer privacidad.

Lo más interesante de esto es que las disciplinas de arquitectura puras están empezando a ofrecer esta clase de flexibilidad luchando contra su medio natural poco flexible.

Desde la perspectiva del diseño web en los comienzos se combatió la flexibilidad natural que ofrece su medio. Ethan Marcotte expresa en su libro “Responsive Web Design” [2] que, en cualquier disciplina creativa un artista comienza su trabajo eligiendo un “canvas”[2], un pintor elige el lienzo, un escultor la roca o bloque necesario. La elección del canvas es importantísima ya que establece las dimensiones, forma, e impone un límite o alcance al trabajo. En el diseño web se trata de imitar este proceso, el problema radica en que el canvas para este dominio (browser) es cambiante y cuenta con muchas inconsistencias e imperfecciones. El diseño queda a merced de las configuraciones de fuentes, colores, tamaños de ventana, etc.

En medio de tanta incertidumbre se comienzan a establecer restricciones, definir layouts de tamaños fijos en pixeles o asumir resoluciones de pantalla mínimas. Estas restricciones permiten establecer límites y parámetros a partir de los cuales se puede

---

[1] Christopher Wren, uno de los más aclamados arquitectos ingleses escribió “architecture aims at eternity”. Las decisiones creativas de un arquitecto perduran por décadas o incluso siglos.

[2] Escenario, espacio o marco en el cual se va a desarrollar un trabajo.

empezar a trabajar. Sin embargo uno de los problemas de la web, es que tiende a romper fácilmente cualquier restricción impuesta. Por ejemplo si se establece una resolución mínima, determinados links pueden quedar invisibles, secciones mal posicionadas, párrafos cortados, etc. El porqué de esta situación se remonta a los orígenes del diseño web. Es una disciplina relativamente nueva que utiliza conceptos de diseño gráfico y diseño de impresión (masthead, leading, whitespace, fold, etc.). La esencia del problema es resumida en un párrafo de John Allsopp en el artículo “A Dao of Web Design” [20]

> “*El control que los diseñadores tienen en el medio de impresión y que a menudo desean en la web, es simplemente una función de las limitaciones de la página impresa. Deberíamos adoptar el hecho de que la web no tiene las mismas restricciones, y diseñar para su flexibilidad. Pero primero debemos aceptar el flujo y reflujo de las cosas*”
>
> — John Allsopp

En lugar de crear diseños totalmente desconectados cada uno apuntado a un dispositivo en particular, RWD se enfoca en un diseño con varias facetas de una misma experiencia.

3.2.3 Ingredientes

Los tres ingredientes de RWD son:

- layout basado en una grilla flexible
- imágenes flexibles
- media queries

Grilla Flexible: Es el componente fundamental que existe detrás de RWD y el punto de entrada para empezar a crear diseños que escalan según el tamaño de la ventana del browser sin importar la resolución de pantalla.

Originalmente los diseños web se construían con tamaños fijos expresados en pixeles y contenido centrado. En ese momento las resoluciones de pantallas y dispositivos no eran tan diversas, por lo cual un diseño de 960px de ancho era la solución estándar. En la actualidad las resoluciones son abrumadoramente cambiantes por lo cual un diseño de tamaño fijo no aplica.

Hacia mediados del siglo XX Emil Rude y Josef Muller-Brockman entre otros popularizaron el concepto de grilla tipográfica (ver Figura 3-3): un sistema racional de filas y columnas sobre el cual se pueden ubicar módulos de contenido.

La grilla sirve como un armazón o esqueleto sobre el cual un diseñador puede organizar texto e imágenes fácilmente.

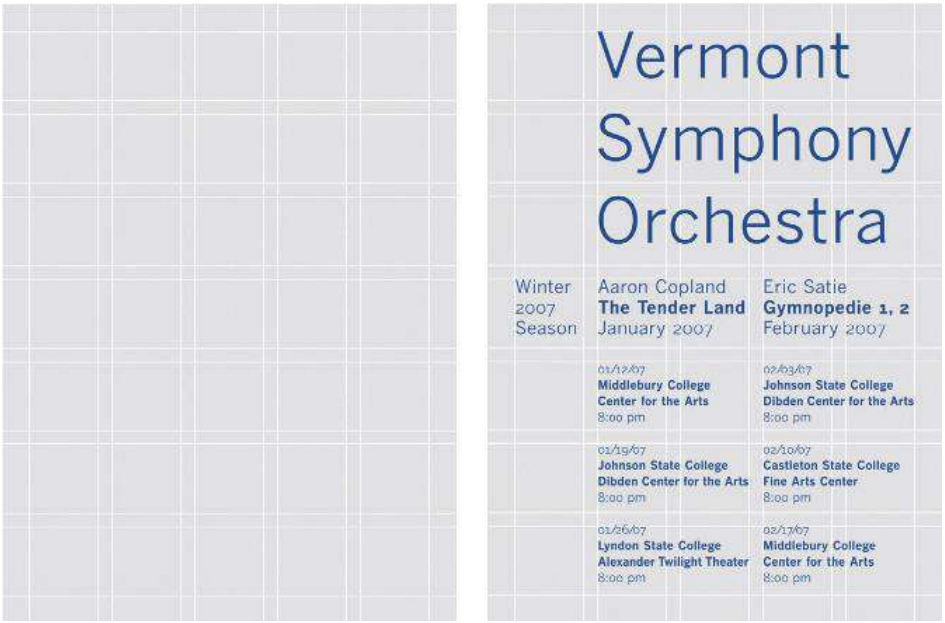


Figura 3-3: Grilla tipográfica, facilita la organización del contenido dentro de una página.

En su libro “Grid Systems in Graphic Design” [21], Müller-Brockmann refiere a este proceso como “creating a typographic space on the page”, adaptando las proporciones de la grilla al tamaño de una pieza de papel. Pero para los diseñadores web falta un componente clave: la presencia de una página. El medio de acción para el diseñador web o canvas es la ventana del browser y esta puede variar en forma y tamaño.

A menudo se comienza creando un elemento HTML y se le da un ancho fijo y se centra en la pantalla. Pero si se piensa en flexibilidad se necesita traducir un diseño creado en Photoshop en algo más fluido y proporcional.

La idea impulsada por la grilla flexible es hacer uso de medidas relativas en lugar de expresar valores absolutos en pixeles. De esta forma no importa la resolución de pantalla, los elementos adoptarán el tamaño relativo correspondiente.

Al basar tamaños de texto, anchos y márgenes en porcentajes o em[3], se puede transformar un tamaño expresado en valores absolutos (pixeles) a valores relativos. Esto significa que es necesario hacer pequeños cálculos para lograr una grilla flexible. La fórmula es muy simple:

$$\frac{objetivo}{contexto} = resultado$$

Basándonos en la Figura 3-4 que parte de un diseño de referencia de 960px se resuelve la ecuación anterior.

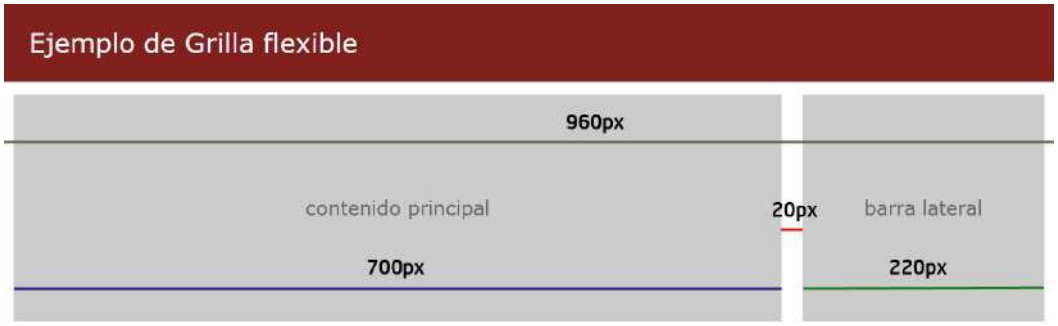


Figura 3-4: Configuración de proporciones en una grilla flexible.

$$\frac{contenido\ principal}{contexto} = proporcion$$

$$\frac{700px}{960px} = proporcion$$

$$\mathbf{0.7291666666666666} = proporcion$$

Una vez calculados los valores para las secciones y los márgenes se pasa a escribir el código CSS que quedaría de la siguiente forma:

```
#main{
display: inline;
float: left;
width: 72.91666666666666%;
margin: 0 1.0416666666666665%;
}
```

[3]Unidad de medida basada en el tamaño de Fuente.

```
#sidebar{
display: inline;
float: left;
width: 22.916666666666664%;
margin: 0 1.0416666666666665%;
}
```

Luego si se redimensiona la ventana del browser podemos observar cómo se mantienen las proporciones anteriores tanto para el ancho de las cajas como para los márgenes.

Figura 3-5: Funcionamiento de la grilla flexible cuando se redimensiona la ventana del browser.

Imágenes Flexibles: El texto se acomoda fácilmente al tamaño del su contenedor flexible, pero ¿qué sucede cuando se introducen imágenes de tamaño fijo en un diseño flexible?

Figura 3-6: Imagen de tamaño fijo en una grilla flexible.

En este escenario el layout no se rompe pero la imagen excede el tamaño de su contenedor. La solución a este problema es configurar a las imágenes para que se comporten de manera flexible. Para esto es necesario escribir una regla que prevenga a las imágenes de exceder el tamaño de su contenedor.

```
img{
max-width: 100%;
}
```

De esta forma los browsers adaptan la imagen proporcionalmente de acuerdo al cambio de tamaño de sus contenedores respetando la relación de aspecto.

Figura 3-7: Imagen flexible en una grilla flexible.

Media Queries: La grilla flexible permite construir aplicaciones web que agregan cierto grado de fluidez a los diseños, pero cuando los cambios son muy pronunciados el layout empieza a deformarse y no aprovecha eficientemente el espacio (ver Figura 3-5). En resoluciones muy por debajo del diseño original el contenido se torna ilegible y amontonado, mientras que en grande resoluciones el espacio sobrante no se aprovecha y quedan al descubierto márgenes desproporcionados.

Las media queries [22] son un mecanismo robusto incorporado por la W3C en el estándar CSS3 que sirven para identificar características físicas de los dispositivos y browsers que muestran el contenido de nuestra aplicación web.

Una media query tiene 2 componentes:

- Empieza con un “media type” (screen, print, braille, handheld, projection, etc)
- Continúa con una consulta o “query” entre paréntesis→ (max-width: 720px). A su vez la query está compuesta por una determinada característica de ese “media type” (max-width) y su valor (720px).

En el siguiente código se crea una media query para cambiar la disposición de los elementos en la grilla flexible y aumentar su ancho cuando la resolución de la pantalla es menor a 720px. Así, se logra un mejor uso del espacio y se mejora la interacción del usuario en dispositivos tales como smartphones.

```
@media screen and (max-width: 720px) {
#main,#sidebar{
display: inline;
float: left;
width: 97.91666666666666%;
margin: 0 1.0416666666666665%;
margin-bottom: 1em;
}
}
```

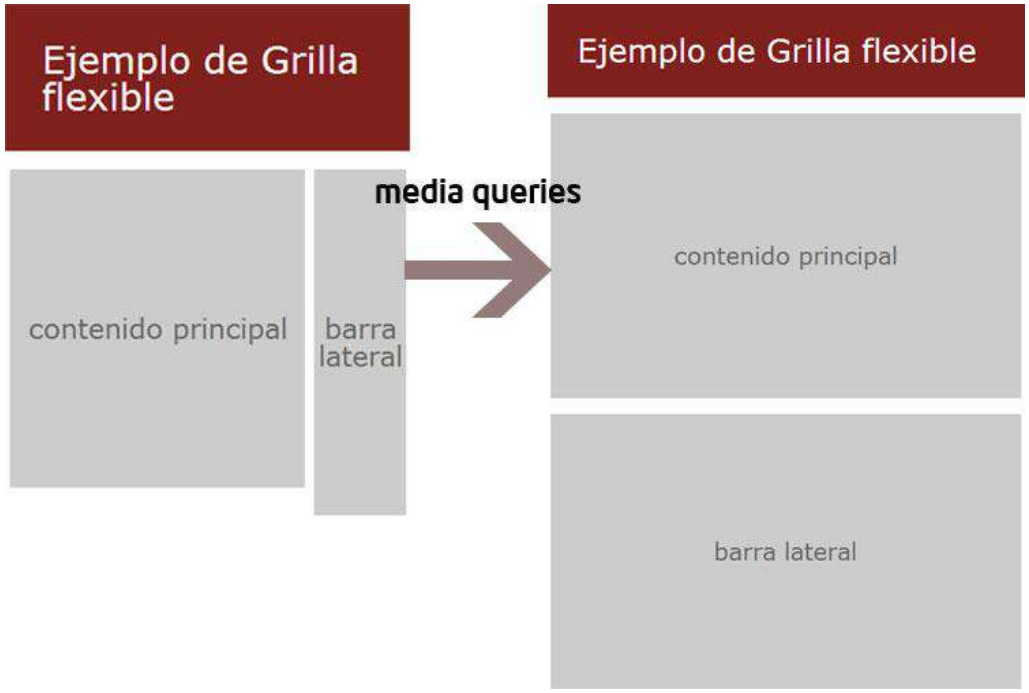


Figura 3-8: Ejemplo de uso de media queries para cambiar la disposición de elementos en la grilla flexible.

3.2.4 Análisis de librerías "Responsive"

En la presente sección se analizan algunas de las librerías CSS que existen en el mercado y se autoproclaman "Responsive".

Twitter Bootstrap [23]: Es mucho más que una librería responsive. De hecho se define como un framework de Frontend para el desarrollo web rápido y fácil.

Ofrece una grilla de 12 columnas y un máximo de 1170px. Tiene múltiples puntos intermedios[4] y la estrategia de responsive consta en reducir el ancho del cuerpo progresivamente, cambiar tipografía y quitar márgenes. Por debajo de 767px, todas las columnas se convierten en filas, en lo que sería una estrategia responsive clásica.

[4] Resoluciones en las que se aplican estilos diferenciados mediante el uso de media queries.

Se destaca la buena y simple documentación y la gran variedad de componentes. Además incluye una serie de plugins de jQuery [24] para los componentes interactivos como menús, barra de navegación, combos, etc.

Está escrita en LESS [25] y es ampliamente personalizable. Permite la creación de una versión personalizada donde se pueden elegir los componentes necesarios y pre configurar las variables de la librería.

Foundation [26]: Se autoproclama “el mejor framework de Frontend del mundo”. Al igual que Bootstrap proporciona una serie de elementos prefabricados como botones, formularios, pestañas, etc. que agilizan el prototipado de sitios web. También incorpora plugins de jQuery [24] para dar funcionalidad a algunos de estos componentes.

Proporciona una grilla flexible de 12 columnas con un ancho fijo configurable. Además permite la construcción de grillas anidadas.

Esta desarrollada con SASS [27].

1140 CSS Grid [28]: Librería responsive básica que sólo provee una grilla flexible de 12 columnas y 1140px de ancho máximo. Los gutters[5] son proporcionales a la resolución, esto puede no gustar en el diseño final.

No incluye ningún tipo de widget prefabricado.

The Semantic Grid System [29]**:** Librería construida en LESS, provee una grilla flexible de 12 columnas y 960px, permite definir la configuración deseada por el usuario en cantidad de columnas, ancho y gutters. Sólo suministra la grilla y a diferencia del resto de las librerías analizadas no requiere que el desarrollador incorpore nombres específicos de clases en los elementos HTML.

Fluid Baseline Grid [30]: Es un kit de desarrollo de HTML5 y CSS3 que provee una base sólida para diseñar sitios web fácilmente. La principal diferencia con todas las librerías anteriores es que fue diseñada siguiendo la filosofía “mobile first”. Los estilos CSS se van personalizando a medida que la resolución crece en los saltos definidos por las media queries en lugar de cuando decrece.

[5]Espacios entre columnas

FBG define por defecto 3 columnas de proporciones iguales, 31.333% de ancho y gutters del 2%. Sin embargo los valores pueden ser cambiados según el proyecto lo requiera.

3.3 Adaptive Web Design

RWD es por definición, como se vio anteriormente, una grilla flexible con imágenes flexibles y media queries que adapta el layout según la resolución. Sin embargo es parte de una filosofía o estrategia mucho más abarcativa. Adaptive Web Design (AWD) [31] [32] es otro término de moda. Se puede considerar que RWD es un subconjunto de AWD (ver Figura 3-9) y que RWD refiere específicamente al layout. Mientras que AWD además, mejora la experiencia de usuario basado en las capacidades que el dispositivo puede ofrecer.

Por ejemplo, si el dispositivo:

- es capaz de realizar llamadas telefónicas → agregar un botón “call now”.
- cuenta con pantalla táctil →agregar funcionalidad al carrusel presente para que sea swipeable además de tener los botones anterior/siguiente
- tiene acceso a la ubicación del usuario →agregar un botón “use mi ubicación” que proporcione la localización automáticamente en lugar de tener que completar un formulario con la misma.

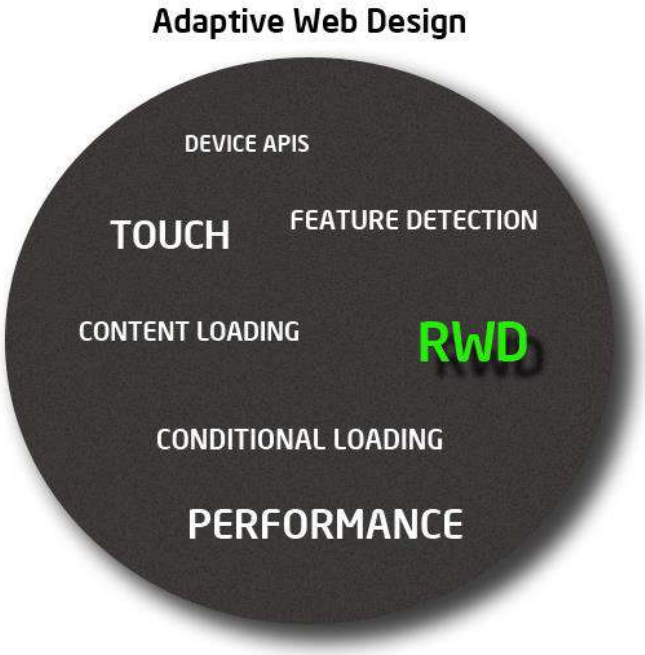


Figura 3-9: RWD es un subconjunto de todo lo que representa AWD.

En un reciente artículo “Beyond Media Queries: Anatomy of an Adaptive Web Design” [33] Brad Frost establece algunos principios relacionados a la adaptabilidad de la web que son importantes considerar:

- **Ubiquidad**: El poder de la web radica en su ubicuidad. Nadie conoce como será el panorama en unos años pero es seguro que cada vez más dispositivos se sumarán. Es necesario preservar y adoptar esta ubicuidad, por lo tanto se deben entregar experiencias web completas independientemente de cómo los usuarios accedan.
- **Flexibilidad**: Tomar ventaja de la fluidez inherente de la web y continuar creando interfaces flexibles agnósticas del dispositivo que puedan adaptarse a cualquier tamaño de pantalla.
- **Performance**: Es unos de los puntos cruciales. Actualmente las actividades relacionadas a mejorar la performance de una Webapp se realizan en la etapa final. Hoy en día el 86% de las Webapps que usan RWD pesan lo mismo en la pantalla más pequeña que en la de mayor tamaño. Hay que cambiar esta condición, priorizar y considerar aspectos de performance desde las fases de diseño. Según sugiere la encuesta “What Users Want From Mobile” [34], el 74% de los usuarios de dispositivos móviles abandonan la Webapp si tarda más de 5 segundos en cargar.
- **Mejora progresiva**: Consiste en definir una funcionalidad “core” que estará disponible en todos los dispositivos y luego ir mejorando la experiencia de usuario según las características disponibles en su contexto (touch, location, phonecall, etc.). Esta estrategia es opuesta al viejo concepto de “graceful degradation” en el que el desarrollador priva al usuario de consumir contenido cuando no posee la tecnología requerida.
- **Futuro amigable**: El aspecto clave del pensamiento “future friendly” es reconocer y aceptar la imprevisibilidad. Es anticiparse y pensar soluciones hoy que sean fácilmente adaptable a los requerimientos futuros.

4 Definición del prototipo

En este capítulo se definen requerimientos funcionales y no funcionales del prototipo con el objetivo de ejercitar la mayor parte de los conceptos y técnicas reveladas en los capítulos anteriores.

El prototipo que se va a construir es un store de aplicaciones web que se ha denominado WASt (Web Application Store). En resumen permitirá al usuario buscar aplicaciones y evaluar sus características en una pantalla de detalles, una vez registrado y autenticado dependiendo del estado de la aplicación podrá simular el proceso descarga e instalación, pausar, reanudar o cancelar la descarga, escribir reviews, etc.

Asimismo la interfaz de usuario se adaptará de acuerdo al contexto en el que se está mostrando, variando la cantidad de aplicaciones y tamaños de las mismas, notificaciones, respuesta a gestos táctiles, etc.

La elección de esta Webapp se fundamenta en que contempla los escenarios suficientes para llevar a la práctica los conceptos de adaptabilidad.

Los objetivos que se abordan con la implementación del prototipo son:

- Hacer uso de la estrategia Mobile First y AWD.
- Utilizar alguna de las funcionalidades avanzadas que HTML5 aporta.
- Ejercitar el proceso de desarrollo de este tipo de aplicaciones, entornos de desarrollo, frameworks, puesta en producción, etc.

El primer punto se logrará definiendo la funcionalidad principal para la versión móvil de WASt y luego se sumarán incrementos en las versiones tablet y desktop. El prototipo definirá tres pantallas principales como home, resultados de búsqueda y detalle de una aplicación y otras secundarias para la registración, login y actualización del perfil del usuario.

Para conseguir el segundo objetivo se pensó en utilizar la tecnología WebSockets que incorpora HTML5. La idea es enviar notificaciones en tiempo real del proceso de descarga e instalación de una aplicación. Por motivos de tiempo este proceso será simulado, es decir, el servidor enviará notificaciones que permitirán ejercitar el mecanismo de notificación, los valores de progreso no representarán la descarga real o instalación

debido a que se necesitaría tener acceso a APIs del sistema operativo que no están disponible en los browsers. Como trabajo futuro se considerará embeber la aplicación HTML5 en un contenedor web para crear una aplicación híbrida que elimine estas restricciones.

Finalmente el último punto refiere a la adquisición de experiencia y resolución de conflictos en el desarrollo de este tipo de aplicaciones. Involucra la selección de herramientas, entorno de desarrollo, análisis de librerías y frameworks, sistema de control de versiones, puesta en producción en el cloud, etc.

La ejecución del prototipo se hará siguiendo una metodología de desarrollo de proyectos Agile [35]. Específicamente se usará Scrum [36]. Aunque suene irrazonable ya que una sola persona está involucrada en el desarrollo y deberá cumplir varios roles (PO[6] y Delivery Team) es importante ejercitar esta práctica tan común en el desarrollo de software actual. Asimismo la metodología servirá como guía para la especificación de los requerimientos mediante “User Stories” [37].

Los requerimientos quedarán definidos mediante User Stories. Cada una expondrá la funcionalidad requerida por el PO. Por otro lado se presentarán los wireframes [38] que mostrarán el diseño de la interfaz de usuario y sus modificaciones desde las pantallas de menor a mayor resolución (“Mobile First”).

Para generar los wireframes se utilizará la herramienta Balsamiq Mockups [39].

4.1 User Stories

Como PO, se desea poder registrar un usuario con información mínima tal como user name, password, email, teléfono, dirección y una foto de perfil (ver Figura 4-1).

Como PO, se desea permitir la autenticación de usuarios mediante dos formatos diferentes. Por un lado accediendo a una página encargada de esta tarea exclusivamente. Por otro lado se desea incorporar una ventana de login que deberá aparecer al momento de ejecutar una acción que requiera autenticación tales como comprar una aplicación o escribir reviews. En este último caso es necesario recordar la última acción y reintentarla automáticamente si el usuario se autenticó exitosamente. El objetivo es no distraer al

[6] Product Owner: Es el dueño y responsable del producto que representa la voz del cliente.

usuario cambiándolo de contexto (llevándolo a otra pantalla) y aprovechar el impulso o iniciativa de compra del momento (ver Figura 4-1 y Figura 4-2).

Como PO, se desea brindar la capacidad al usuario de actualizar su perfil una vez que se encuentre autenticado. La barra de navegación deberá incluir la foto de perfil del usuario que se encuentra logueado y permitir acceder a la pantalla de perfil desde allí (ver Figura 4-1)

Como PO, se desea promocionar aplicaciones en la pantalla de inicio con el objetivo de firmar contratos de publicidad con diferentes desarrolladores de aplicaciones (ver Figura 4-3). Estas aplicaciones deben aparecer en una sección principal de la página de entrada al sitio o home. Asimismo las aplicaciones deberán tener algún identificador que las caracterice como promocionadas si aparecen en otras listas.

Como PO, se desea agregar una lista de las aplicaciones más importantes en la pantalla de inicio con el objetivo de facilitar el acceso a las aplicaciones que frecuentemente solicitan los usuarios (ver Figura 4-3). Desde esta lista debe ser posible navegar a una pantalla “Top Applications” en la cual se podrán acceder a todas las aplicaciones bajo esta sección. La lista debe permitir paginación infinita al hacerse visible el último ítem y el tamaño de la nueva página a pedir depende de la resolución actual con el objetivo de no degradar la performance.

Como PO, se desea buscar una aplicación dentro del store por título desde cualquier pantalla. El acceso a la búsqueda debe ser rápido y reconocible fácilmente (ver Figura 4-4 y Figura 4-5). Adicionalmente se debe proporcionar un lista de términos sugeridos a medida que el usuario tipea su búsqueda.

Como PO, se desea ver los resultados de búsqueda en forma de lista y con paginación infinita (ver Figura 4-5).

Como PO, se desea ver el detalle de una aplicación de interés cuando se hace click en el icono o título de la misma (ver Figura 4-6).

Como PO, se desea adquirir (comprar o descargar) una aplicación haciendo énfasis en proveer feedback en tiempo real de los procesos de descarga e instalación de la aplicación (ver Figura 4-7). Debe ser posible iniciar múltiples procesos de descarga en simultáneo desde las listas de aplicaciones y proveer feedback del progreso en los

diferentes estados (“processing, downloading, installing”). La pantalla de evaluación de la aplicación permitirá acciones adicionales tales como pausar, reanudar o cancelar la descarga.

Como PO, se desea ver una lista de las reviews de la aplicación cuando se accede al detalle para facilitar el proceso de elección y compra (ver Figura 4-6). Esta lista de reviews debe ser cargada solo cuando el usuario la requiera y actualizarse automáticamente cuando el mismo usuario envía una nueva review.

Como PO, se desea ver una lista de aplicaciones relacionadas cuando se accede al detalle para facilitar el proceso de búsqueda y sugerir otras aplicaciones de la misma categoría (ver Figura 4-6). Al igual que la lista de reviews esta sección debe cargarse bajo demanda ya que es una sección secundaria y no se desea demorar el tiempo de carga de la página de detalles con peticiones que están un segundo nivel de importancia.

Como PO, se quiere permitir al usuario escribir reviews de una aplicación. Esta acción sólo debe estar disponible desde la página de detalle una vez que la aplicación ha sido instalada. Es necesario que el usuario este logueado y después de enviar la review la misma debe reflejarse automáticamente en la lista de reviews como la más reciente. De la misma forma los gráficos de reviews asociados deben reflejar este nuevo valor incorporado.

Como PO, se desea agregar un gráfico de barras en la página de detalle o evaluación de aplicación que represente visualmente el estado y promedio de las reviews actuales. El gráfico debe actualizarse automáticamente cuando el usuario actual agrega una nueva review (ver Figura 4-6).

4.2 Wireframes

Los wireframes representan el diseño aproximado que la aplicación tendrá en sus diferentes resoluciones. Permiten definir la disposición de los componentes gráficos de la aplicación sin necesidad de contar con un diseño visual final.

Como parte de los requerimientos, la aplicación WASt define tres rangos de resoluciones que implican cambios de layout en las diferentes pantallas. Estas especificaciones

complementan las historias de usuarios anteriores. Generalmente son propuestas por un diseñador de interacción (UX Engineer).

4.2.1 320px <= resolución <= 767px

La mayor parte del trabajo de diseño y definición de las entidades del sistema se definen en este rango debido a que se está siguiendo el enfoque “Mobile First”. En este momento es cuando se deciden los componentes esenciales de la aplicación. El resto de los rangos definirán pequeños cambios o variaciones para presentar la información de manera más adecuada o agregar nuevos componentes con un nivel de prioridad menor.

Este rango está pensado para abarcar smartphones tanto en orientación portrait como landscape. Asimismo pueden quedar comprendidas tablets en orientación portrait.

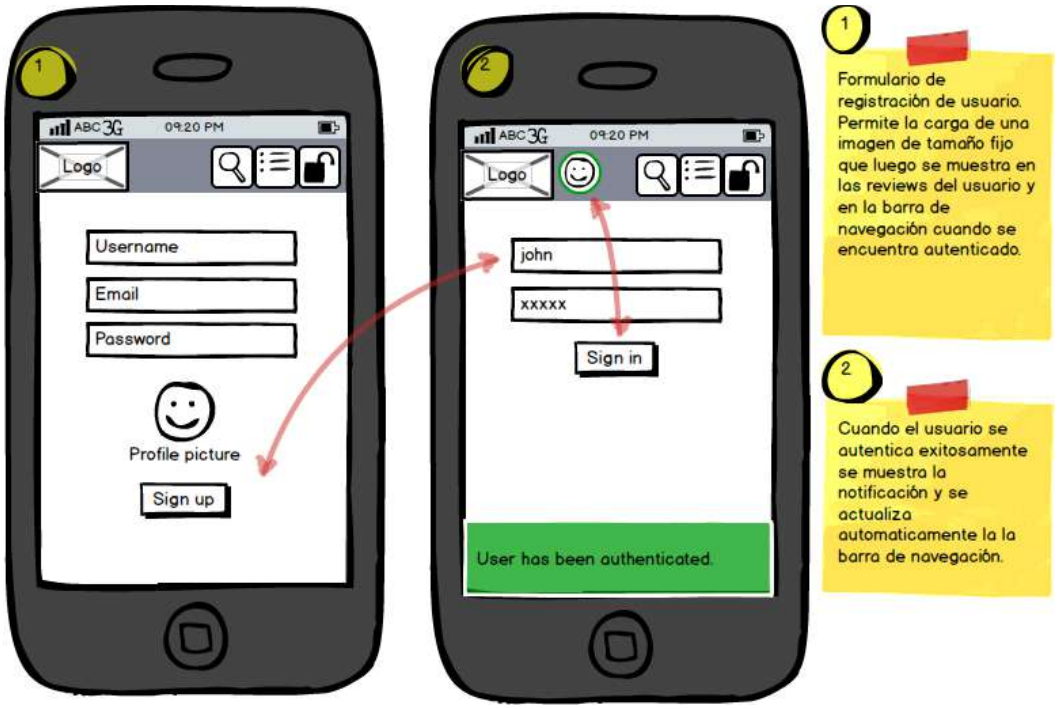


Figura 4-1: Registración y autenticación de usuario

Figura 4-2: Login en formato “modal”

Figura 4-3: Wireframe de la pantalla inicial para 320px <= resolución <= 767px

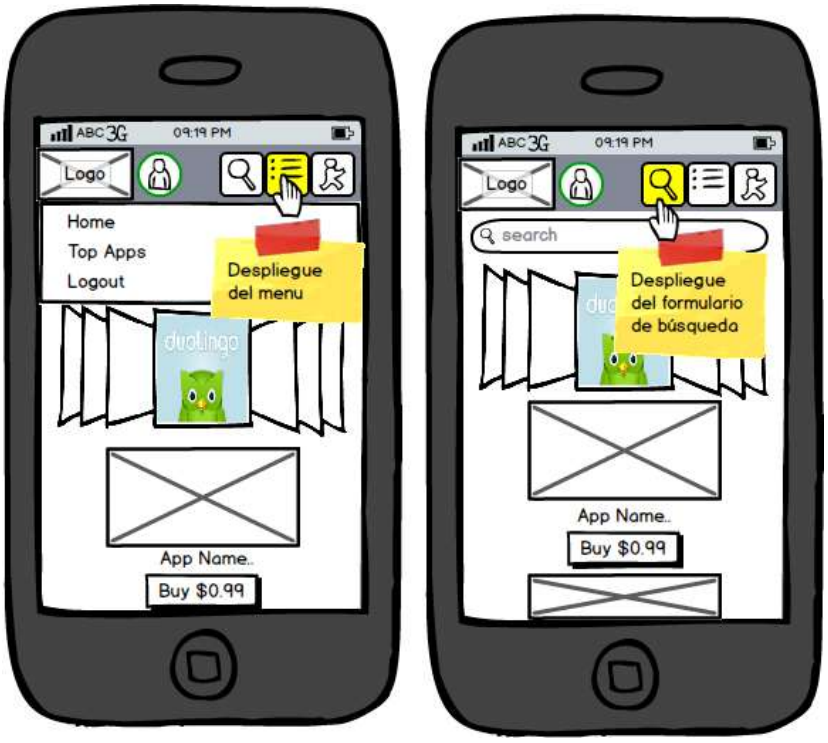


Figura 4-4: Ubicación de la barra de navegación y cuadro de búsqueda.

Figura 4-5: Pantalla de resultados de búsqueda para 320px <= resolución <= 767px

Figura 4-6: Wireframes de la pantalla de detalle para 320px <= resolución <= 767px

Figura 4-7: Notificaciones de progreso y acciones del usuario.

Este rango internamente incluye otro pequeño salto (420px) para ajustar la cantidad de columnas mostradas en las listas de aplicaciones a dos.

Figura 4-8: Ajuste de layout cuando se alcanza un ancho de pantalla de 420px

4.2.2 767px < resolución <= 980px

Este rango está destinado a abarcar tablets en orientación portrait y landscape así como también resoluciones de laptops y desktops. Además pueden ingresar smartphones de alta gama que poseen resoluciones muy elevadas (Samsung Galaxy SIII, iPhone 5).

En este rango de resolución solo se ajustan la página de inicio adecuando información adicional al carousel y se cambia las disposición de varios elementos dentro de la página de detalle de aplicación. El acordeón pasa a ser un panel con tabs, los screenshots así como gráficos de reviews se reacomodan aprovechando mejor el espacio horizontal disponible en este rango. Además todas las listas de aplicaciones pasan a tener tres columnas.

Figura 4-9: Pantalla inicial para 768px < resolución <= 980px

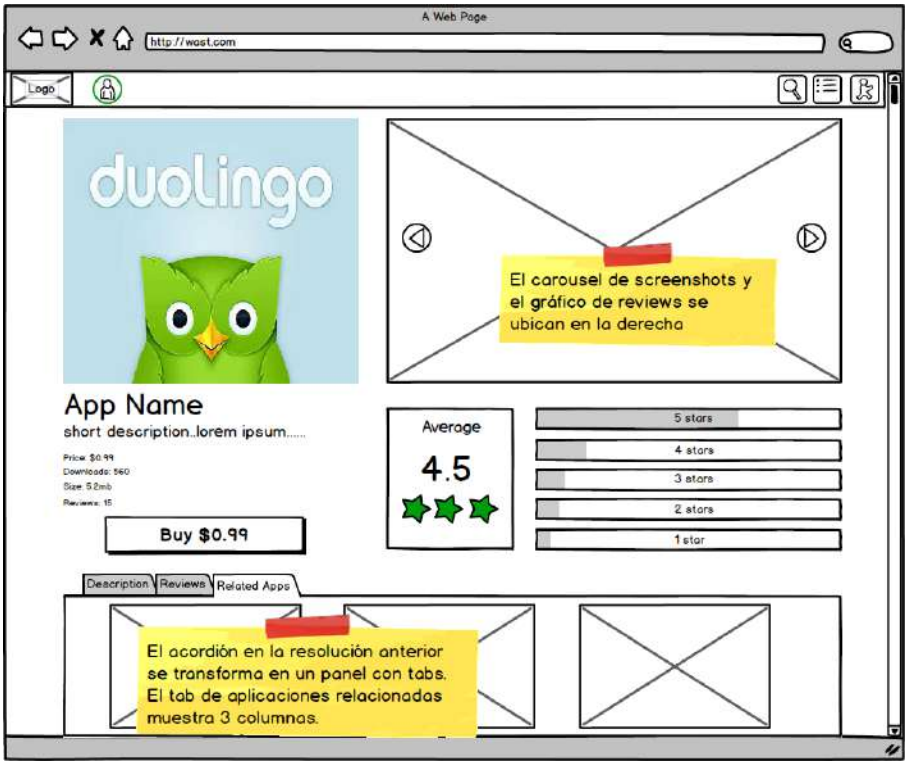


Figura 4-10: Página de detalles de aplicación en 768px < resolución <= 980px

4.2.3 980px < resolución <= 1200px

Los cambios en esta categoría se implementan en la barra de navegación y la cantidad de columnas en las listas de aplicaciones que aumenta a cuatro.

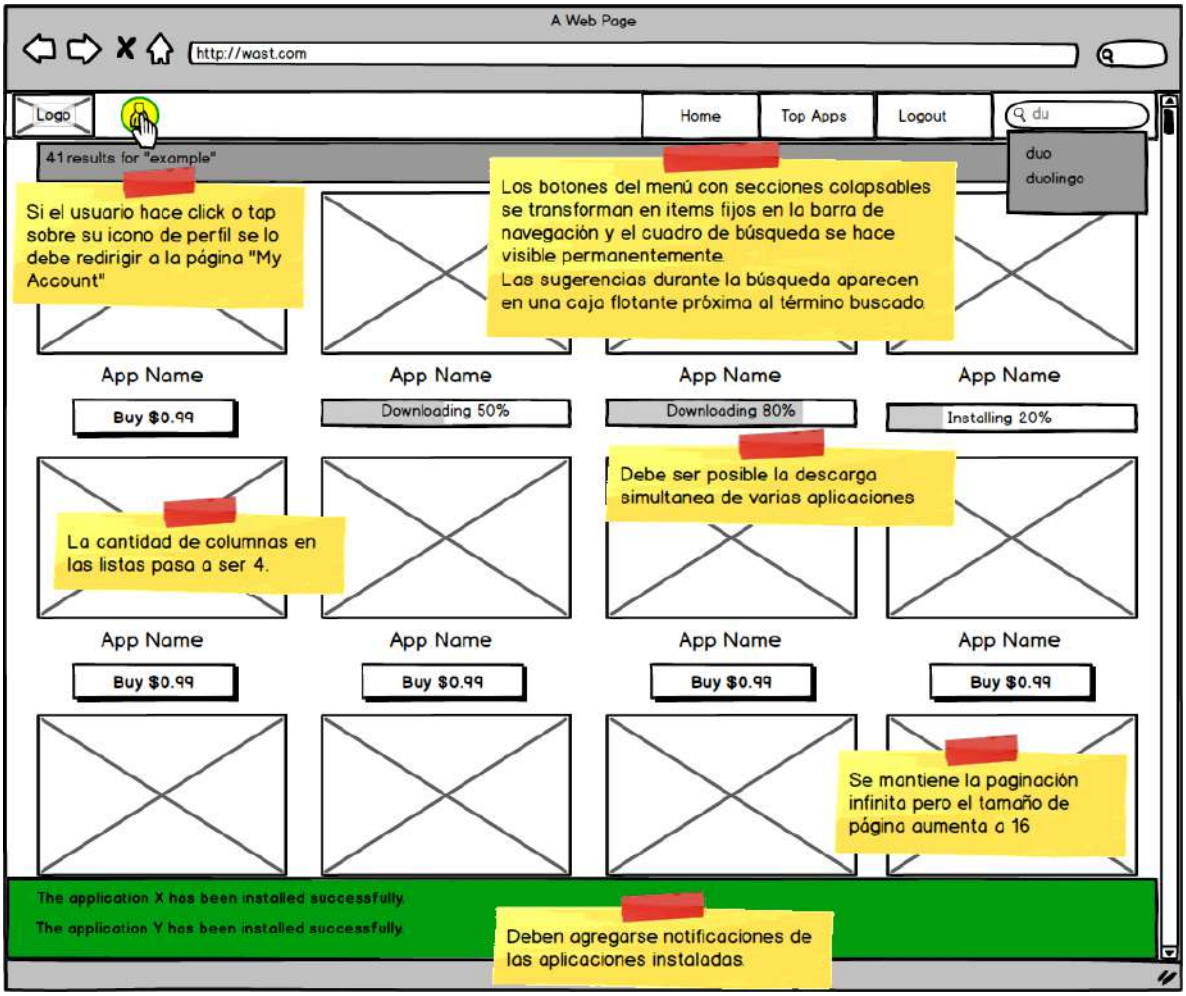


Figura 4-11: Página de búsqueda en 981px < resolución <= 1200px

4.2.4 1200px < resolución

El último rango considerado está representado por resoluciones mayores a 1200px, en esta categoría quedarían incluidas fundamentalmente desktops y laptops.

En este escenario se desea centrar el contenido horizontalmente. En el futuro se podrían agregar más funcionalidades como faceted search en la página de búsqueda o secciones de publicidad en la páginas de inicio.

Figura 4-12: Páginas con 1200px < resolución

5 Implementación

En este capítulo se especificarán los frameworks, librerías y tecnologías usadas. Asimismo se expondrán los módulos principales y las dificultades encontradas durante el diseño e implementación del prototipo.

El código de la implementación está disponible online en un repositorio de dominio y acceso público [40] así como también la aplicación web desplegada y en funcionamiento [41].

A continuación se desglosarán las tareas más importantes que tuvieron lugar durante la implementación. Básicamente pueden clasificarse en 3 grupos:

- Definición de tecnologías para el Backend e implementación del mismo.
- Análisis y elección de frameworks JavaScript y CSS3 para la creación del Frontend siguiendo el enfoque SPA y la estrategia Mobile First.
- Definición y creación de un ambiente de desarrollo y despliegue.

5.1 Backend

Uno de los puntos que se menciona en el alcance del trabajo es la implementación del prototipo en un lenguaje común entre el Frontend y Backend. Esto representa una clara ventaja y actualmente se puede lograr con el lenguaje “JavaScript”. La tendencia de utilizar un mismo lenguaje se incrementó notablemente a partir de 2009 con el surgimiento de un proyecto open source denominado Node.js [42]. Node.js es una plataforma construida sobre la base del Engine JavaScript V8 (popular por ser el mismo utilizado en el browser Chrome) que permite la construcción rápida de aplicaciones usando el lenguaje de programación JavaScript. A su vez proporciona un manejador de paquetes (npm) que permite instalar módulos para tareas específicas (manejo de archivos, servidor web, módulos de comunicación para WebSockets, acceso a bases de datos, etc).

Haciendo uso de esta plataforma ahora es muy sencillo construir una servidor web en el mismo lenguaje que el cliente.

El Backend desarrollado para el prototipo se compone de los siguiente módulos:

- Servidor web: Encargado de servir los recursos estáticos de la Webapp (imágenes, HTML, CSS, JavaScript) y exponer las APIs REST necesarias para satisfacer las necesidades del Frontend.
- Servidor de Base de Datos: Responsable de persistir la información relacionada a los usuarios, aplicaciones y reviews.
- Search Engine: Este módulo es el encargado de indexar las aplicaciones y proveer capacidades avanzadas de búsqueda como sugerencias, faceted search, paginación, etc.

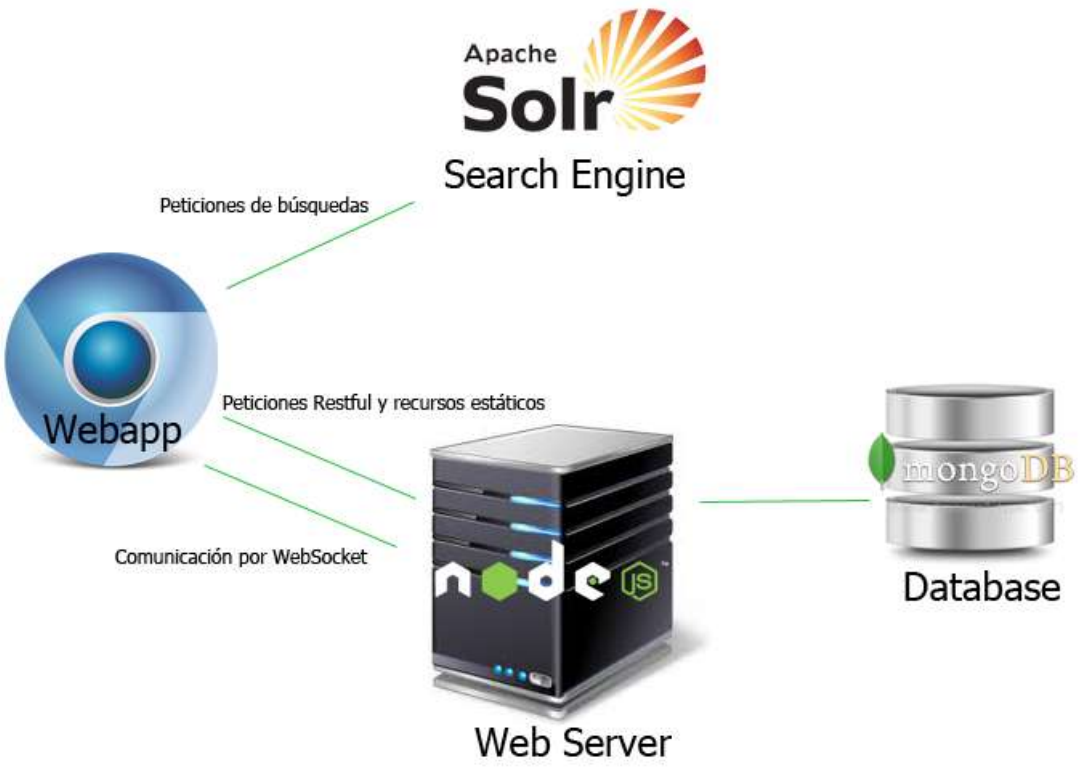


Figura 5-1: Componentes del Backend y productos elegidos

5.1.1 Servidor Web

Por los requerimientos planteados y las ventajas previamente mencionadas se decidió utilizar Node.js como plataforma para el servidor web. Sobre Node.js se acopla, entre

otros, un módulo llamado express [43] que provee toda la funcionalidad necesaria asociada a un framework de desarrollo web (manejo de sesión, cookies, routing, etc).

Otro de los módulos utilizados es socket.io [44]. La necesidad de usarlo surge para cumplir con las historias de usuario relacionadas a proveer feedback en tiempo real del proceso de descarga e instalación de una aplicación por parte de un usuario. Este módulo se encarga de abstraer la comunicación por WebSockets con los clientes y permite registrarse o emitir eventos fácilmente. Además provee soporte crossbrowser y mecanismos de fallback cuando WebSocket no es soportado por el cliente (como “long polling”).

Uno de los desafíos en la implementación de la Webapp fue permitir el uso de WebSockets para comunicación bidireccional sólo a los usuarios autenticados. Para esto fue necesario agregar una pequeña porción de código que reutilice información de la sesión de los usuarios provista por express. Cuando un cliente ingresa a la Webapp se intenta abrir un socket, si el usuario se encuentra autenticado previamente, la petición de apertura del socket debería portar la cookie del usuario y únicamente en ese caso la conexión es exitosa.

La mayor parte del trabajo de implementación en el servidor se dividió en:

- Crear el proceso de simulación de descarga e instalación de las aplicaciones una vez que el usuario se encuentra autenticado.
- Definir los handlers de cada una de las peticiones RESTful del Frontend para la sincronización y persistencia de usuarios, aplicaciones y reviews.

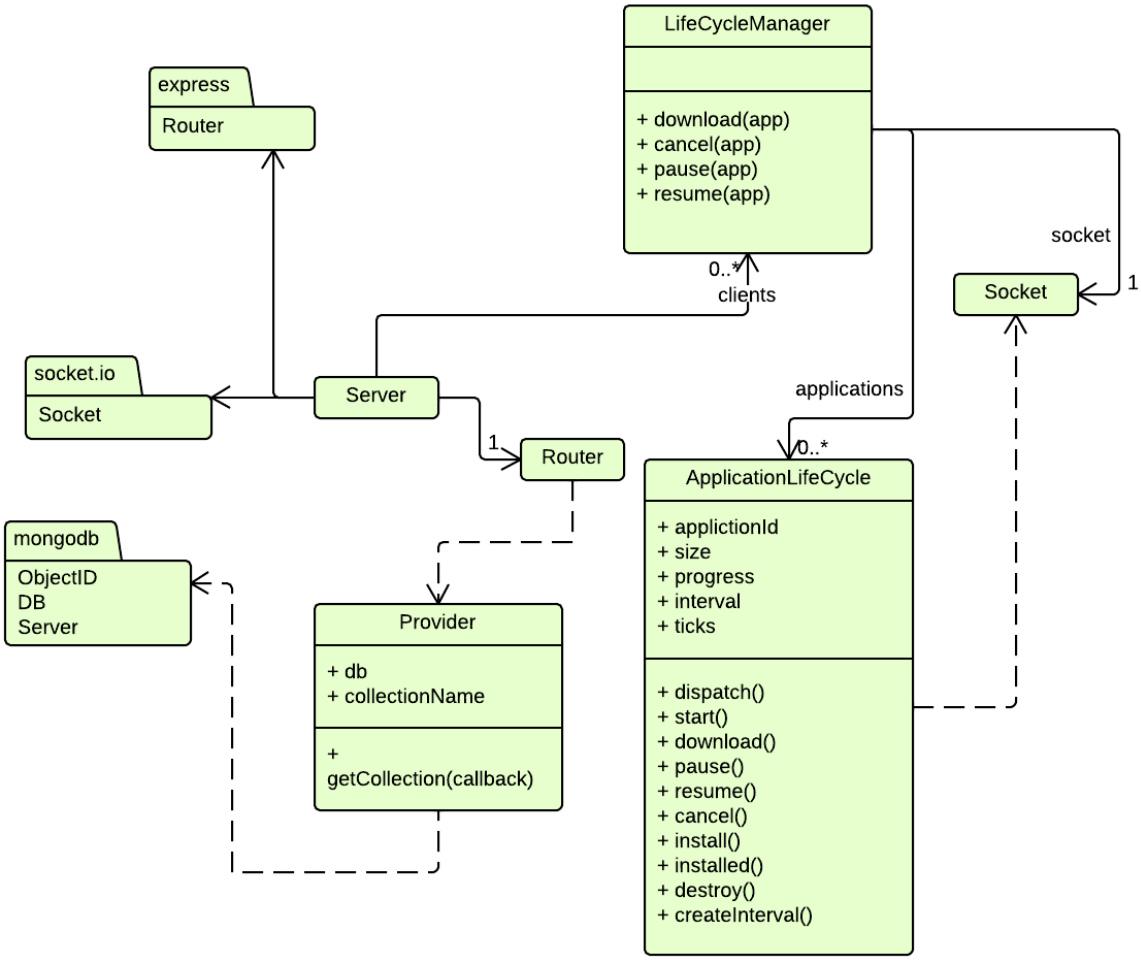


Figura 5-2: Diagrama de clases de los componentes principales del Backend (WebServer)

5.1.2 Servidor de Base de Datos

Existían muchas alternativas al momento de elegir una base de datos. Se optó por MongoDB [45] , es una base de datos orientada a documentos (NoSql) que ofrece excelente integración con Node.js. Existe un módulo “mongodb” que actúa como driver implementando todas las APIs expuestas por el servidor de base de datos.

Por otro lado una de las grandes ventajas de usar esta base de datos NoSQL es que los objetos JavaScript que representan los modelos de datos de User, Review y Application son mapeados directamente a documentos JSON y almacenados en la base de datos sin ningún tipo de mapeo o procesamiento intermedio. Asimismo mongodb soporta guardar documentos cuyos atributos contengan valores compuestos como arrays u otros objetos anidados lo cual se ajusta perfecto a los modelos de datos que se grafican a continuación.

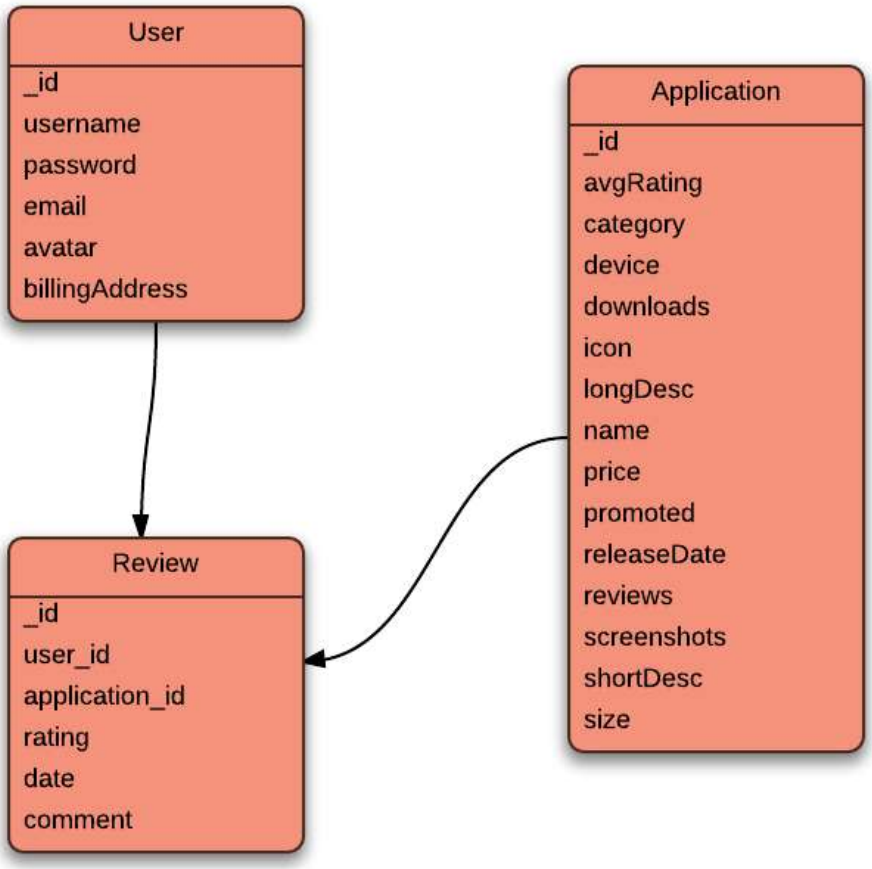


Figura 5-3: Modelo de datos

5.1.3 Search Engine

La decisión de utilizar un Search Engine se tomó para facilitar el proceso de búsqueda y sugerencias al usuario tal como se plantea en las historias de usuario. Habría sido posible implementar una API en el servidor que ejecutara las consultas sobre MongoDB, sin embargo hubiera implicado un enorme esfuerzo de programación para obtener resultados de menor calidad a los que ofrece el Search Engine seleccionado.

Apache Solr [46] fue el elegido, es un proyecto open source de la fundación Apache que se implementa sobre Lucene. Ofrece innumerables funcionalidades relacionadas a búsquedas, filtros, faceted search, sugerencias, “more like this”, etc. Además expone APIs en formato JSON lo cual suma puntos para la integración con el Frontend. Por otro lado la performance de Solr tiene sobradas pruebas de éxito en el mercado de software actual.

El trabajo realizado para poner en funcionamiento Solr junto a la Webapp se limitó a:

- Instalar Solr, modificar los archivos de configuración scheme.xml y solrconfig.xml especificando el esquema de las entidades que se desean guardar e indexar (Application). Agregar y configurar un módulo de sugerencias de búsqueda para enviarlas al Frontend cuando el usuario comienza a escribir una búsqueda.
- Crear un script que genere aplicaciones, usuarios y reviews aleatoriamente y las inserte tanto en MongoDB como en Solr para iniciar la Webapp con un set de datos preestablecidos.
- Integrar las páginas del Frontend que necesitan consumir datos directamente desde Solr (search, suggestions, top apps). Aquí fue necesario utilizar JSONP [47] para poder hacer peticiones “cross domain”.

Una de las tareas que permanece pendiente está relacionada a la sincronización de los datos de “Applications” almacenados tanto en MongoDB como en Solr. Por ejemplo cuando un usuario descarga una aplicación el número de downloads actualmente se actualiza en MongoDB pero por el momento no se envía ese update al Search Engine.

5.2 Frontend

Antes de comenzar a detallar la implementación del Frontend es importante puntualizar algunos conceptos esenciales en el desarrollo de Webapps hoy en día que servirán para justificar posteriormente las elecciones que se han tomado en torno a librerías y frameworks.

Uno de los puntos claves en el desarrollo de esta tesis es demostrar la posible construcción de determinados tipos de Webapp adaptables con niveles de interacción y experiencia de usuario similares a los de una aplicación nativa. Bien, para lograr esto hay que profundizar el concepto previamente mencionado de SPA o “Single Page Application” (ver 2.3.2).

Técnicamente la mayoría de las Webapps hoy en día son SPAs. El hecho de que toda la funcionalidad de la aplicación sucede dentro de una única página web la vuelve interesante y atractiva. La principal razón es que permite ofrecer experiencias de usuario enriquecidas y considerablemente más placenteras que una pantalla en blanco mientras se recarga una nueva página. Sin embargo soportar estas interacciones de múltiples componentes en una misma página significa que esos componentes poseen muchos más estados intermedios (Por ejemplo: menú X abierto, ítem Y seleccionado, etc.) los cuales

son difíciles de implementar si se intentan mapear a peticiones URL para su renderización en el servidor.

Las SPAs se distinguen por su capacidad de redibujar cualquier parte de la interfaz de usuario sin la necesidad de recuperar el HTML necesario desde el servidor. Esto se logra definiendo una capa que se encarga del manejo y validación de los datos por un lado y otra capa que lee esos modelos de datos y construye una presentación acorde.

Básicamente lo que se necesita es una arquitectura que permita resolver de manera organizada todos estos nuevos problemas que presentan las Webapps modernas.

Los programadores se obsesionan a menudo más con la facilidad que con la simplicidad. Se piensa que escribir el código es la parte más difícil aunque en realidad lo más complicado es el mantenimiento del mismo.

Para escribir código mantenible es necesario conservar las cosas simples. Esta tarea no es sencilla y se transforma en una lucha constante por dos motivos esenciales:

- Es fácil agregar complejidad (dependencias) para resolver un problema
- Es fácil resolver un problema de una forma que no reduce la complejidad

5.2.1 Arquitectura de las Webapps

Las Webapps modernas son generalmente estructuradas como especifica el siguiente diagrama:

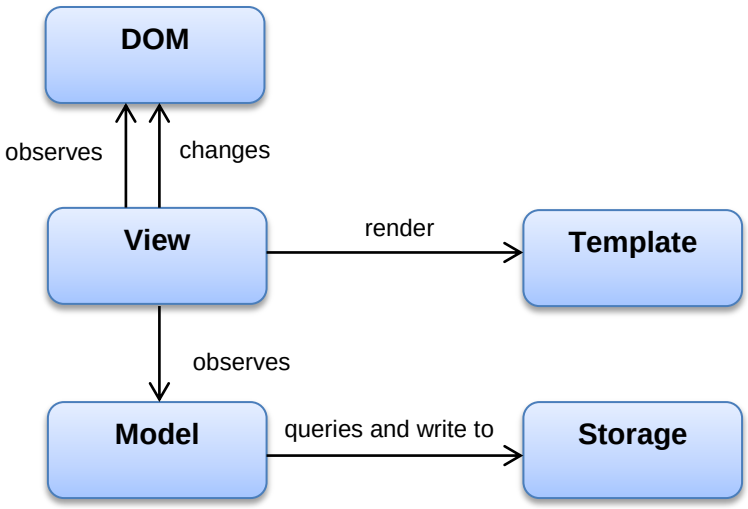


Figura 5-4: Arquitectura de las Webapps.

Las ideas que persigue esta arquitectura es:

- **DOM de solo escritura:** Significa que no se deben leer datos o estados desde elementos del DOM. Almacenar estados de la aplicación en el DOM se vuelve muy difícil de manejar rápidamente. Es mejor contar con un lugar donde residen los datos (Models) y luego renderizar las piezas de HTML necesarias desde esos modelos.
- **Modelos como única fuente de verdad:** En lugar de almacenar datos en el DOM u objetos aleatorios, debe existir un conjunto de modelos que representan el estado de la aplicación.
- **Las vistas observan cambios en el modelo:** Se quiere que las vistas reflejen el contenido de los modelos. Cuando existen múltiples vistas que dependen de un solo modelo no se desea mantener estas referencias dentro del modelo. Lo correcto es que el modelo emita eventos de los cambios que sufrió y las vistas que estén interesadas en esos cambios se registren y definan las acciones a seguir ante esos cambios (por ejemplo redibujarse).
- **Las vistas observan eventos en el DOM:** Las vistas son responsables de escuchar eventos para la porción de DOM que han generado y responder a esos eventos ejecutando las acciones correspondientes, por ejemplo: modificar un modelo, persistirlo, emitir otro eventos, etc.

5.2.2 Mix de tecnologías involucradas en el Frontend

Existen numerosos puntos a considerar en la elección de las tecnologías para cubrir los aspectos de la arquitectura señalada. Además para tornar la decisión aún más difícil el panorama puede ser intimidante al principio dado la cantidad de opciones disponibles. Sin embargo los últimos años han traído un consenso sobre las herramientas y técnicas que hacen la experiencia de desarrollo de aplicaciones de lo más agradable y productiva posible.

Hay una gran mezcla de tecnologías en el lado del cliente (Frontend), así como de librerías y prácticas que habilitan una experiencia productiva en el desarrollo de aplicaciones. Estas pueden resumirse en la siguiente figura:

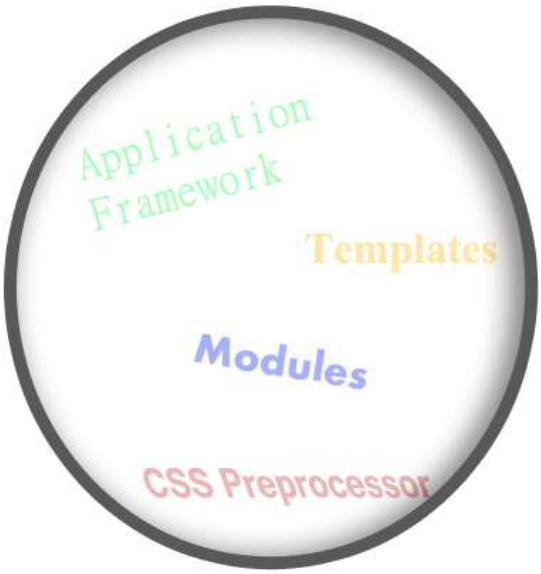


Figura 5-5: Mix de tecnologías de Frontend

Application Framework: Existen numerosos frameworks para elegir entre los que podemos mencionar Backbone.js, AngularJS, EmberJS, CanJS, SpineJS, YUI3, etc. Según la complejidad, curva de aprendizaje y conjunto de funcionalidades ofrecidas estos frameworks podrían ordenarse como sigue:

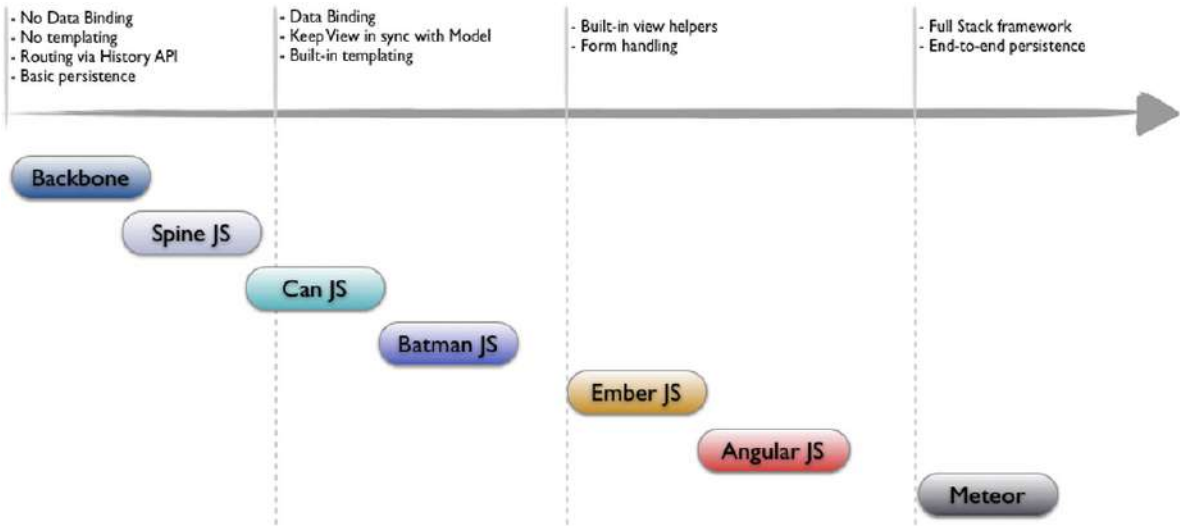


Figura 5-6:Application Frameworks ordenados por complejidad y funcionalidades ofrecidas

La mayoría de estos siguen el patrón de diseño Modelo Vista Controlador (MVC) similar al presentado en la arquitectura anterior aunque con ligeras variaciones y agregados particulares (ver Figura 5-7).

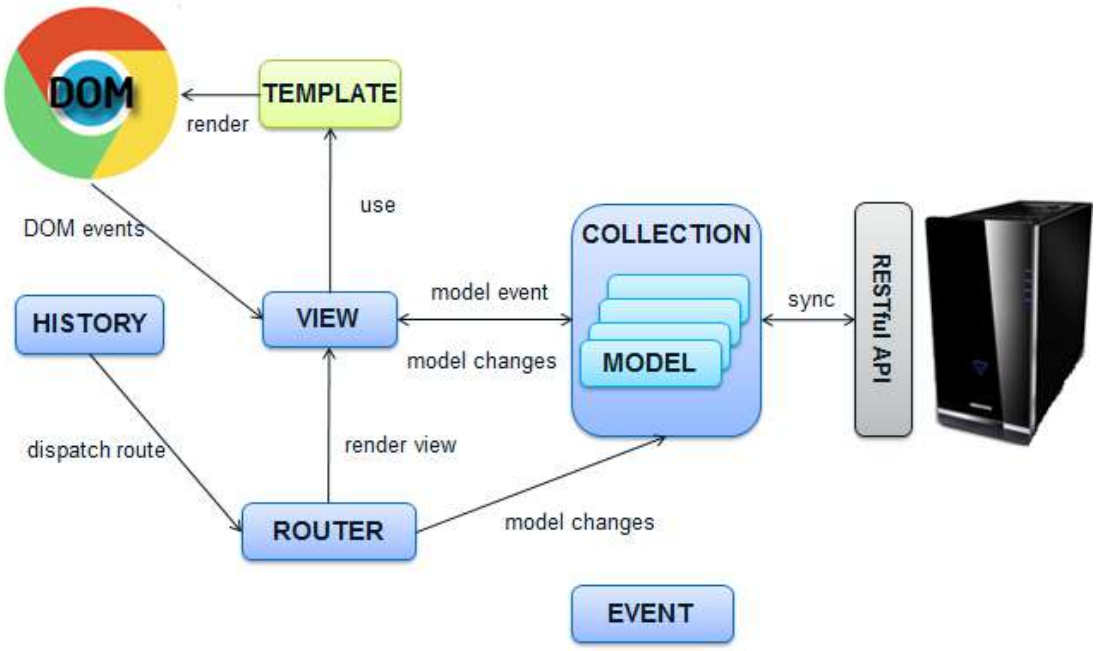


Figura 5-7: Arquitectura común provista por los frameworks JavaScript

- **Model**: Es un "wrapper" sobre un objeto JSON con soporte para cambio de propiedades mediante setters/getters y notificación de eventos ante cambios.
- **Collection**: Representan una colección de modelos y emiten notificaciones cuando los modelos son agregados, removidos o modificados dentro de la colección.
- **Events**: Es un módulo que sigue el patrón (Pub/Sub) en el cual las diferentes partes de la aplicación pueden registrarse y publicar eventos.
- **View**: Es un objeto o unidad lógica que sirve como soporte a un fragmento de DOM. Tiene la capacidad de registrarse a eventos producidos en el DOM y además puede acceder al modelo subyacente de datos. Al mismo tiempo se suscribe a otros eventos asociados al modelo o alguno en particular con semántica para la aplicación. Es el responsable de generar el HTML final fusionando las estructuras de datos de los modelos con los templates definidos para la vista.

- **Routing**: Ofrece navegación dentro de la aplicación vía URLs. Se base en API History expuesta por de los browsers.
- **Syncing**: Se encarga de la persistencia de los cambios en el modelo mediante llamadas AJAX.

Para la implementación del prototipo se eligió el framework Backbone.js [48]. Especialmente porque cubre los requerimientos planteados, es simple, fácilmente extensible, con una gran comunidad y se cuenta con experiencia previa en el mismo. Además es uno de los más estables y seguidos en lo que respecta al desarrollo de Frontend (ver Figura 5-8)

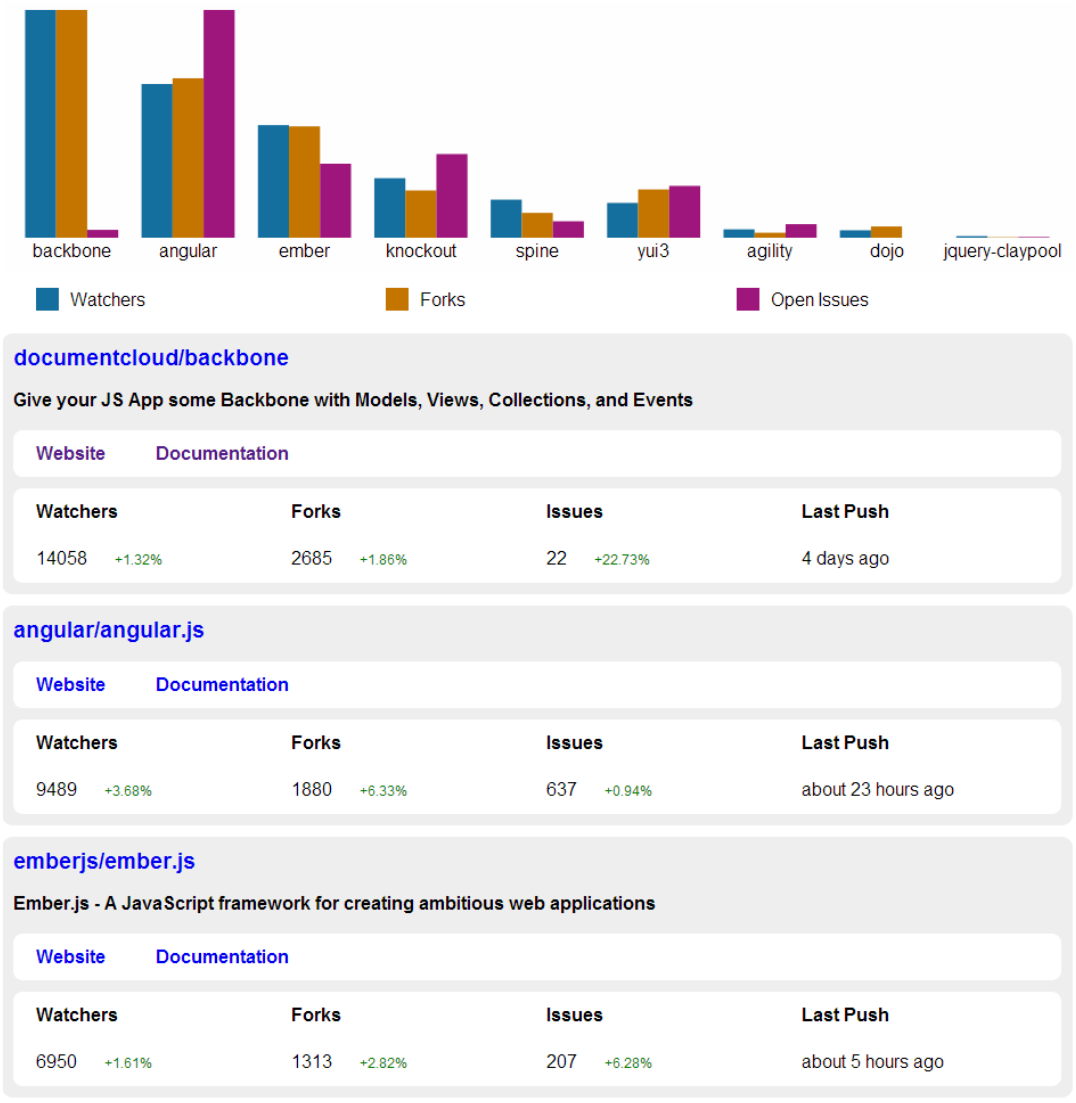


Figura 5-8: Captura tomada de (js-toolbox.org) que muestra la popularidad de los repositorios de frameworks JavaScript (4/4/2013)

Desarrollo Modular: Tradicionalmente el código JavaScript se agregaba en las páginas mediante el tag <script>. Se listaban todas las librerías y otras dependencias primero y luego una lista de todos los scripts que referencian a esas dependencias. Este estilo funciona bien cuando la aplicación no es compleja y solo es necesario incluir unos pocos archivos. Sin embargo se vuelve una pesadilla cuando la complejidad aumenta.

Por otro lado los desarrolladores desean mantener la funcionalidad encapsulada en archivos bien identificados pero al mismo tiempo al momento de despliegue de la aplicación web esos módulos deben ser ensamblados y comprimidos para optimizar el funcionamiento y disminuir la cantidad de peticiones del cliente.

Una solución es tratar a cada archivo como un Módulo identificado por un nombre, que expresa un conjunto de dependencias y expone determinada funcionalidad a través de una API.

Algunos de los frameworks mencionados previamente proveen esta funcionalidad “out of the box” como YUI3. Sin embargo debido a que el framework elegido (Backbone.js) no la posee se escogió la librería RequireJS [49].

RequireJS agrega a la aplicación la capacidad de definir módulos, dependencias y escribir la porción de código que incumbe al módulo que se está definiendo. Además contribuye a eliminar uno de los problemas más nocivos de las aplicaciones como las variables globales.

```
//nombre
define('myModule',
  //dependencias
  ['math', 'graph'],
  function ( math, graph ) {
  //Aquí se encapsula el código de myModule.
  //Todas las variables definidas en este ámbito son locales.
    return {
      plot: function(x, y){
        return graph.drawPie(math.randomGrid(x,y));
      }
    }
  };
);
```

Por otro lado RequireJS resuelve automáticamente dependencias anidadas. Cada una de estas dependencias al ser resuelta termina generando la petición del script al servidor y lo

agrega en el tag <head>. Sin embargo la librería ofrece herramientas para resolver todas la dependencias y generar un único script que puede ser minimizado y comprimido al momento de generar el paquete final optimizado.

Client-side Templates: Al momento de renderizar una porción de HTML en el cliente, el componente View es el encargado combinar el modelo al template que corresponde. Los templates son solo texto con unas pocas marcas para reemplazar el contenido del modelo y la idea es mantenerlos completamente aislados de la lógica de la aplicación. Backbone.js ofrece un mecanismo de templates provisto por una de sus dependencias (underscore.js) sin embargo se optó por incluir Handlebars.js [50] por su simplicidad y enfoque orientado a evitar lógica dentro de los templates.

Los templates para cada una de las vistas son editados como archivos ".html" con las marcas correspondientes a los atributos del modelo que se desear reemplazar (ver Figura 5-9). Sin embargo es importante recalcar que estos archivos son "pre-compilados" para generar un único archivo final "templates.js" que posteriormente se carga como una dependencias más en la inicialización de la aplicación.

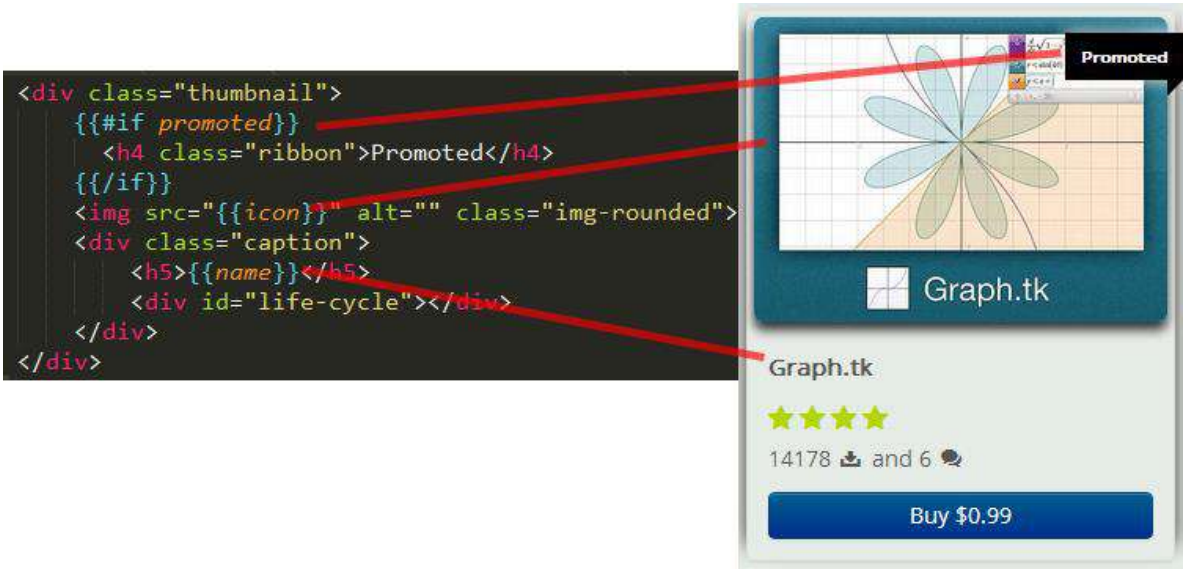


Figura 5-9: Template de Handlebars para la vista de aplicación en listas.

CSS Preprocessor y Frameworks: Otra de las herramientas muy útiles al momento de comenzar a desarrollar una Webapp es el uso de lo que se conoce “Preprocesadores CSS”. Esencialmente son lenguajes que complementan CSS3 con ciertas utilidades para mejorar la productividad durante el desarrollo.

Los dos lenguajes más conocidos son LESS y SASS. Ambos proveen un conjunto de funcionalidades parecidas como el uso de variables, funciones, mixins, herencia de selectores, etc. Estas funcionalidades contribuyen a generar código CSS más fácil de mantener. Dos ejemplo claros (ver Figura 5-10) en los que se ve su gran utilidad pueden ser:

- Uso de variables: Si un color es utilizado en varios lugares dentro de una hoja de estilos el mismo puede ser reemplazado por una variable. Además las tonalidades de colores de otros componentes pueden ser generadas en base a esta variable con diferentes funciones de colores. Así explicitando un par de variables de colores el resto es calculado automáticamente y cambiar de un tema a otro se reduce a solo cambiar el valor de estas variables.
- Uso de mixins: Los mixins son porciones de CSS reutilizables y su uso está más difundido en la generación de CSS con propiedades aun no estandarizadas y que necesitan prefijos de los diferentes browsers (-web-kit, -ms, -moz).

```
// Definicion de Variable
$bodyBackground:        #e5ebe6 ;

// Definicion de Mixin
@mixin border-radius($radius) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
          border-radius: $radius;
}

//Uso de variables y mixins.
.application {
  @include border-radius(20px);
  background: $bodyBackground;
}
```

Figura 5-10: Ejemplo de definición y uso de mixins en SASS

En la implementación del prototipo se optó por utilizar SASS. La elección está fundamentada en el hecho de proporcionar mayor funcionalidad además de que es patrocinada por varios gurúes y se integra mejor a los entornos de desarrollo y depuración (Chrome recientemente incluyó soporte para depurar código CSS con referencias al código SASS que lo generó).

Para empezar el proceso de maquetado del prototipo se optó por elegir una de las librerías responsive analizadas en la sección 3.2.4. Twitter Bootstrap fue la candidata número uno. Sin embargo esta librería incorpora muchas clases CSS en los elementos HTML que polucionan la maqueta y confunden al momento de trabajar. Por este motivo se decidió utilizarla en conjunto con semantic.gs. De esta forma se utilizan las librerías de manera combinada obteniendo lo mejor de cada una.

Ambas librerías Twitter Bootstrap y semantic.gs están disponibles en SASS por lo cual encajan perfectamente en la decisión previa.

Font-Awesome [51] es otro proyecto open source muy interesante que se utiliza en el prototipo. Es una fuente icónica que se integra con Twitter Bootstrap y define numerosos iconos muy útiles que se representan por código. De esta forma se evita el uso de imágenes y se optimiza notablemente la carga de la página.

Un punto a resaltar relacionado a la compilación tanto del código SASS como de los templates de Handlebars es que fue necesario crear tareas para automatizar estas actividades repetitivas durante el desarrollo. Para estas actividades se utilizó otro proyecto open source (si uno más!) llamado Grunt.js [52]

5.2.3 Detalles de implementación sobre la adaptabilidad aplicada al prototipo

El enfoque adoptado para lograr la adaptabilidad en el prototipo tiene 2 partes esenciales.

- Uso de media queries siguiendo la teoría de RWD y Mobile First
- Detección de características del dispositivo (por ahora solamente resolución) en las vistas de Backbone.

Uso de media queries

Se definió un catálogo de componentes CSS que extienden las clases ya provistas por Twitter Bootstrap y sobre esa base se customizan las propiedades deseadas. Para esto se utilizó una nomenclatura en la cual para cada nuevo componente que se intenta

personalizar (_forms.scss por ejemplo) se crea un nuevo archivo SASS y se extiende la clase agregando un identificador “one, two, three, four” para cada una de las diferentes variantes de estilos que sean necesarias. Así la librería se mantiene intacta y no se reemplaza ningún estilo de manera accidental.

Por otro lado cada uno de estos componentes CSS especifican las media queries necesarias siguiendo la idea de “Mobile First” con lo cual los estilos por defecto se aplican a las resoluciones menores y luego se definen las pequeñas variantes para los saltos de resolución progresivos. El siguiente fragmento de código pertenece al componente _application.scss. Es un componente nuevo que por supuesto no existe en la librería Twitter Bootstrap por lo cual se creó y se definieron dos variantes (“one” y “two”) que serán los estilos aplicados en la vista de “detalle” y “lista” respectivamente. Prestando especial atención a “.application.one” se puede observar que en los estilos por defecto se aplica un mixin que incluye 12 columnas (aquí es donde se utiliza semantic.gs la cual está configurada con un máximo de 12 columnas) y representa el 100% del ancho del contenedor.

```
.application.one {
  position: relative;
  @include column(12);
  @include application-box();
}
.application.two {...
}
@media (min-width: 420px) {
  .application.one {
    @include column(6);
    margin-bottom: 20px;
  }
  .application.two {...
  }
}
@media (min-width: 767px) {
  .application.one {
    @include column(4);
    margin-bottom: 20px;
  }
  .application.two {...
  }
}
@media (min-width: 980px) {
  .application.one {
    @include column(3);
    margin-bottom: 20px;
  }
}
```

Figura 5-11: Fragmento de código SASS de _application.scss

A medida que la resolución crece cuando alcanza un mínimo de 420px se modifica la cantidad de columnas ocupadas que pasa a ser 6/12 (50%). En los próximos valores mínimos (767px y 980px) también se produce el mismo efecto ocupado 4/12 (33.333%) y 3/12 (25%) respectivamente.

Como resultado final en los diferentes cambios de resolución las aplicaciones se acomodarán automáticamente en 1, 2, 3 y 4 columnas (ver Figura 5-12). Cabe destacar que en cada “salto” de resolución pueden refinarse aún más estilos para ajustar perfectamente la interfaz (Por ejemplo los márgenes como se ve en la figura anterior).

Figura 5-12: Media queries aplicadas.

Detección de características del dispositivo

La intención de esta idea fue optimizar el funcionamiento de la aplicación en determinadas circunstancias haciendo uso la capacidad de detección de características del dispositivo que facilita la librería Modernizr [53].

Con esto se logra por ejemplo:

- Ante cambios muy grandes en la resolución, cambiar el template aplicado a una vista para mostrar más o menos información.
- Deshabilitar la registración a eventos “touch” en las vistas si el dispositivo no soporta esta funcionalidad.
- Optimizar el tamaño de la páginas requeridas al Backend en una búsqueda de acuerdo al tipo de dispositivo (Smartphone/Tablet/PC)

Por el momento se llegó a implementar la detección de resolución del dispositivo usando Modernizr pero el mismo concepto puede ser expandido fácilmente a otras características.

Durante la implementación del prototipo se produjeron situaciones en las cuales un componente visual o vista variaba tanto de una resolución a otra que era imposible acomodarlo con el mismo template usando solo media queries. Tal es el caso del panel que muestra la descripción, reviews y aplicaciones relacionadas en la página de detalles de aplicación (ver Figura 5-14 y Figura 5-15). En resoluciones pequeñas ese panel se muestra y comporta como un acordeón pero cuando pasa a resoluciones mayores debe actuar como un panel con tabs. Además la cantidad de aplicaciones relacionadas pedidas para las diferentes resoluciones debe variar. Asumiendo que existe más espacio disponible se pueden requerir más aplicaciones.

Para dilucidar como se llevó a cabo la implementación de esta funcionalidad a continuación se presenta un diagrama simplificado de las entidades principales que actúan en la página de detalle de aplicación (el mismo no puede ser considerado un diagrama de clases de forma estricta ya que en JavaScript no existe el concepto de clases pero sí ayuda a entender el funcionamiento gracias a la orientación a objeto aplicada).

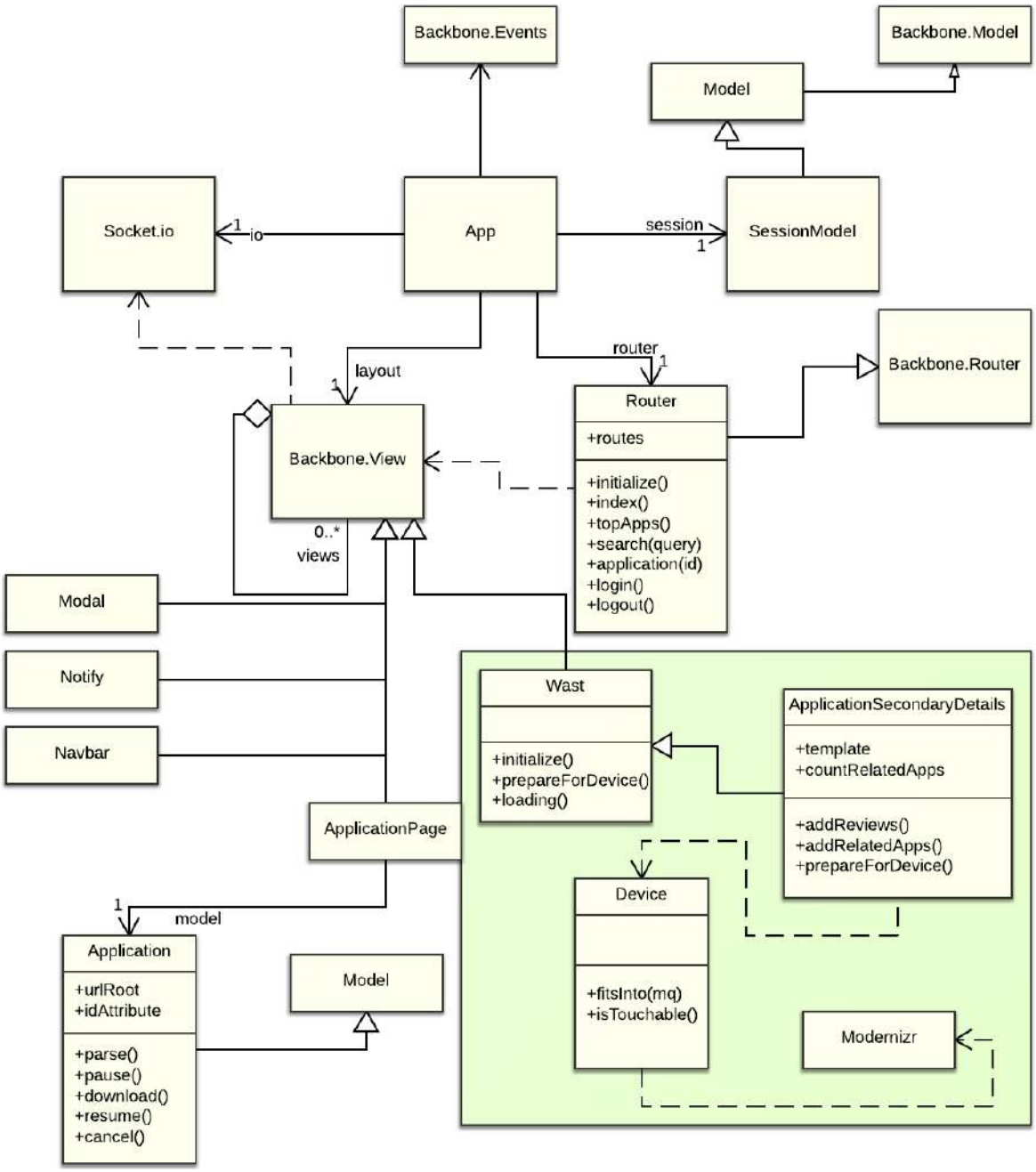


Figura 5-13: Diagrama de entidades de la página de detalles de aplicación

La vista “Wast” se registra a eventos de cambio de resolución en la ventana e invoca el método “prepareDevice” para que efectué los cambios correspondientes (por defecto no realiza operación alguna).

La vista “ApplicationSeccondaryDetails” extiende de “Wast” y sobrecarga el método “prepareDevice” para cambiar el template y actualizar la cantidad de aplicaciones relacionadas a pedir correspondiente a cada salto de resolución.

En la sobrecarga del método prepareDevice, ApplicationSeccondaryDetails hace uso del módulo “Device” el cual expone determinadas APIs que proveen información del dispositivo valiéndose de la librería Modernizr.

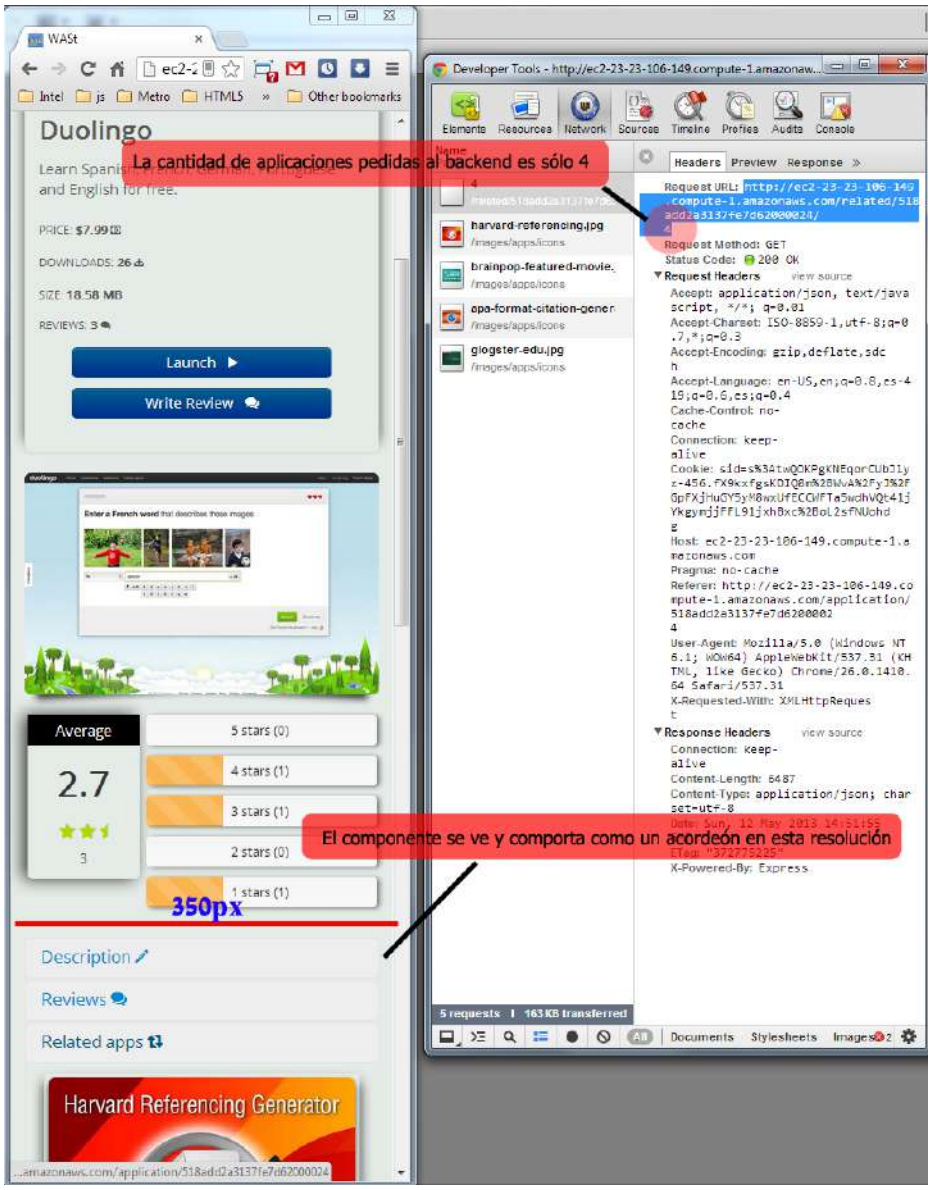


Figura 5-14: Página de detalle de aplicación a 350px

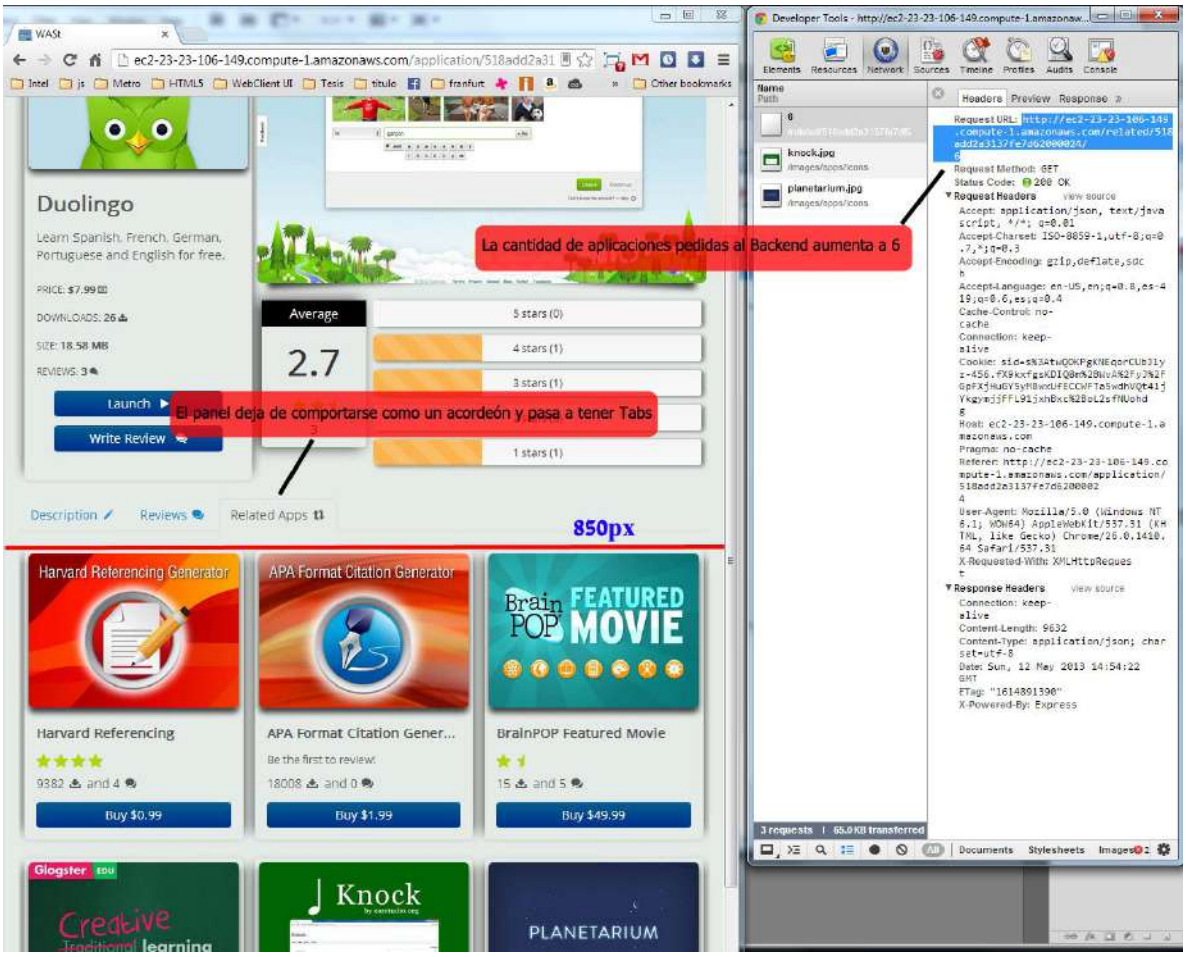


Figura 5-15: Página de detalle de aplicación a 850px

5.3 Ambiente de desarrollo y producción

5.3.1 Ambiente de desarrollo

Una vez elegidas todas las tecnologías necesarias para empezar la implementación del prototipo fue necesario definir un ambiente de desarrollo que facilite el trabajo diario.

Primero se creó un repositorio público en Github [40] para mantener el código versionado y poder ejecutar tareas de despliegue continuo en el ambiente de producción en el futuro.

Respecto a la organización de los archivos el proyecto se dividió en 2 partes: cliente y servidor. El objetivo es desacoplar los procesos de despliegue. Los cambios aplicados en el servidor no deben influir en el paquete final o artefacto generado en el módulo del cliente y lo mismo en sentido inverso.

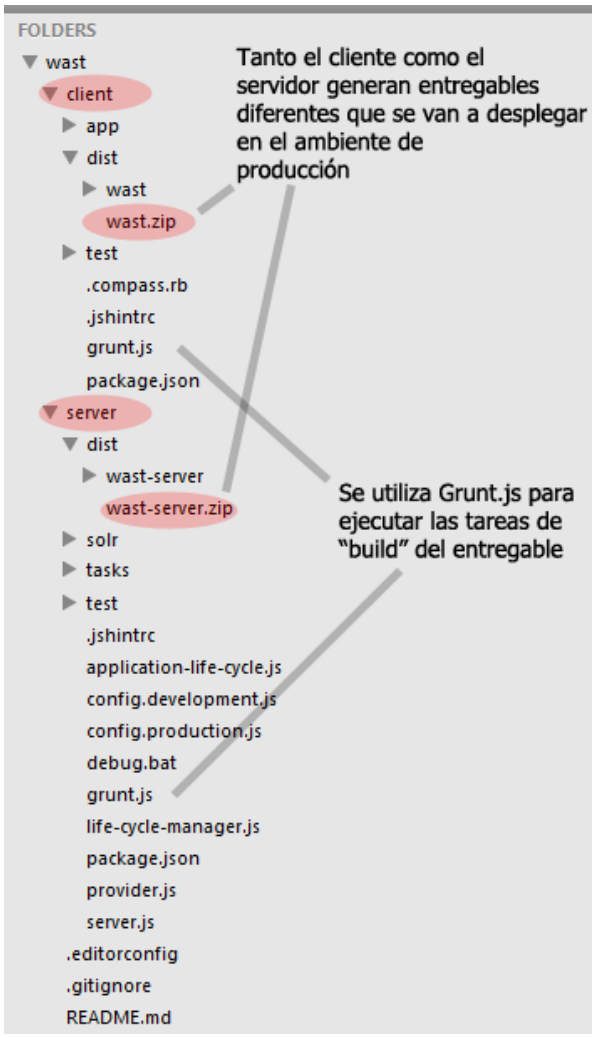


Figura 5-16: Organización de archivos de WASt

Una de las actividades más importantes en este ámbito fue evitar las tareas repetitivas como compilación de templates y código SASS, ejecución de herramientas de calidad de código sobre JavaScript (JSHint [54]), minimización del código, optimización de imágenes, compresión y generación del paquete entregable final, etc. Para estas actividades se utilizó un módulo de Node.js llamado Grunt.js.

Tanto cliente como servidor tiene un archivo grunt.js que especifica un conjunto de tareas a ejecutar durante el proceso de desarrollo, build o deployment. Por ejemplo, en el cliente, durante el desarrollo se ejecuta la tarea "grunt watch" la cual se encarga de monitorizar las modificaciones en los archivos y automáticamente genera la compilación tanto del código SASS como de los templates Handlebars y ejecuta JSHint.

Cuando el código del cliente está listo para ser enviado a producción se definieron otras tareas (“grunt default” y “grunt deploy-s”) que se encargan de ejecutar las anteriores pero también otras adicionales como minimizar el código JavaScript, optimizar las imágenes y generar un archivo comprimido que se envía automáticamente por ssh al entorno de producción configurado.

5.3.2 Ambiente de producción

El prototipo se encuentra online en [41]. El ambiente es una máquina virtual de Amazon. Específicamente el servicio ofrecido como Amazon EC2 (Elastic Compute Cloud).

La gran ventaja de estos servicios es que el costo es proporcional al uso y ante una fuerte demanda o ingreso de usuarios al sistema, el servidor puede escalar sus capacidades automáticamente.

Este servicio EC2 permite elegir entre diferentes “Amazon Machine Images” en este caso se optó por una imagen de Ubuntu Server.

Una vez instalado el sistema operativo, Amazon nos proporciona un mecanismo para crear claves “.pem” mediante las cuales podemos acceder remotamente al servidor con SSH e instalar o configurar los programas deseados. Así es que se instalaron todas las dependencias necesarias para poner en funcionamiento el prototipo, entre ellas:

- Node.js
- MongoDB
- Solr

Una vez que se obtuvo acceso por SSH al ambiente de producción y todos los servicios se encontraban en ejecución, usando las mismas claves SSH las tareas de despliegue tanto del cliente como del servidor automáticamente se conectan y descomprimen los entregables.

6 Conclusiones

Se cree que el prototipo cumplió con las expectativas de adaptabilidad planteadas. El enfoque “Mobile First” es sencillo de implementar una vez asimilada la idea, no plantea grandes desafíos a nivel técnico y ayuda a entender qué es realmente primordial para la aplicación. Sin embargo tiene que ser concebido desde las fases tempranas de definición del producto para facilitar el proceso de desarrollo.

Seguir la metodología definida por RWD aportó la mayor parte de la adaptabilidad a los dispositivos que logró el prototipo. No obstante, queda restringida sólo al tamaño de la pantalla y hay que pensar en alternativas si se desea lograr un mayor grado adaptabilidad al dispositivo en cuestión.

HTML5 ofrece cada día más características a las aplicaciones web. En el prototipo se vio reflejado la utilidad en el uso de WebSocket para notificaciones en tiempo real. Además esta soportado por prácticamente todos los browser aunque se encuentre en estado borrador. Sin embargo hay ciertas restricciones que plantean los browsers de las cuales no se puede escapar. En el prototipo los eventos de progreso enviados por WebSocket debieron ser simulados en el servidor debido a la carencia de APIs que proporcionen esa información de lado del cliente. Es en esos momentos donde se debe replantear la idea de Webapp o Aplicación Híbrida.

Se logró desarrollar el prototipo compartiendo el mismo lenguaje de programación entre el cliente y el servidor (JavaScript) se cree que este es un punto muy favorable. Pero al mismo tiempo hay tener en cuenta el crisol de tecnologías y frameworks que rodean a las Webapps actuales (Node.js, SASS, LESS, Handlebars, Backbone.js, Twitter Bootstrap, Grunt.js, etc) para no reinventar la rueda, entender cuál es el propósito de cada una y estar preparado para usarlas. De todas maneras el balance es positivo comparado a la misma aplicación desarrollada para Android, iOS, BlackBerry OS y Windows Phone.

7 Bibliografía

[1] Wikipedia. Write once, run anywhere. [Online]. http://en.wikipedia.org/wiki/Write_once,_run_anywhere

[2] Ethan Marcotte, *Responsive Web Design*.: A Book Apart, 2011.

[3] Luke Wroblewski, *Mobile First*.: A Book Apart, 2011.

[4] Wikipedia. (2012, Oct.) Wikipedia. [Online]. http://en.wikipedia.org/wiki/Web_application

[5] Bob Baxley. boxesandarrows. [Online]. http://www.boxesandarrows.com/view/what_is_a_web_application_

[6] Wikipedia. Web 2.0. [Online]. http://en.wikipedia.org/wiki/Web_2.0

[7] Miniwatts Marketing Group. Internet World Stats. [Online]. http://www.internetworldstats.com/stats.htm

[8] International Telecommunication Union. Key statistical highlights: ITU data release June 2012. [Online]. http://www.itu.int/ITU-D/ict/statistics/material/pdf/2011%20Statistical%20highlights_June_2012.pdf

[9] Royal Pingdom. (8, May) Mobile share of web traffic in Asia has tripled since 2010. [Online]. http://royal.pingdom.com/2012/05/08/mobile-web-traffic-asia-tripled/

[10] Luke Wroblewski. (2012, Feb.) Why Mobile Matters. [Online]. http://www.lukew.com/ff/entry.asp?1506

[11] Bloomberg. (2012, Jan.) EBay's PayPal Mobile Payment Volume Rose to $4 Billion Last Year. [Online]. http://www.businessweek.com/news/2012-01-11/ebay-s-paypal-mobile-payment-volume-rose-to-4-billion-last-year.html

[12] techcrunch. (2012, Jan.) eBay Forecasts $8B In Mobile Commerce Volume In 2012; PayPal Will Reach $7B. [Online]. http://techcrunch.com/2012/01/12/ebay-forecasts-

8b-in-mobile-commerce-volume-in-2012-paypal-will-reach-7b/

[13] PhoneGap. [Online]. http://phonegap.com/

[14] Christina Warren. (2010, Dec.) Netflix Talks HTML5 in its Device Apps. [Online]. http://mashable.com/2010/12/03/netflix-html5/

[15] Alexis Deveria. Can I use. [Online]. http://caniuse.com/

[16] Brad Frost. (2012, Aug.) The Many Faces of 'Mobile First'. [Online]. http://bradfrostweb.com/blog/mobile/the-many-faces-of-mobile-first/

[17] Ethan Marcotte. (2010, May) Responsive Web Design. [Online]. http://www.alistapart.com/articles/responsive-web-design

[18] Wikipedia. Responsive architecture. [Online]. http://en.wikipedia.org/wiki/Responsive_architecture

[19] Simos Yannas, "Towards Environmentally-Responsive Architecture," in *The 20th Conference on Passive and Low Energy Architecture*, Santiago – CHILE, 2003, p. 6.

[20] John Allsopp. (2000, Apr.) A Dao of Web Design. [Online]. http://www.alistapart.com/articles/dao/

[21] Josef Muller-Brockmann, *Grid Systems in Graphic Design*.: Ram Publications, 1996.

[22] W3C. Media Queries. [Online]. http://www.w3.org/TR/css3-mediaqueries/

[23] Mark Otto. Bootstrap. [Online]. http://twitter.github.com/bootstrap/

[24] The jQuery Foundation. jQuery. [Online]. http://jquery.com/

[25] Alexis Sellier. LESS. [Online]. http://lesscss.org/

[26] ZURB. Foundation. [Online]. http://foundation.zurb.com/

[27] Nathan Weizenbaum, Chris Eppstein Hampton Catlin. SASS. [Online]. http://sass-lang.com/

[28] Andy Taylor. 1140 CSS Grid. [Online]. http://cssgrid.net/

[29] Tyler Tate. The Semantic Grid System. [Online]. http://semantic.gs/

[30] Josh Hopkins. Fluid Baseline Grid. [Online]. http://fluidbaselinegrid.com/

[31] Aaron Gustafson. (2011, Nov.) On Adaptive vs. Responsive Web Design. [Online]. http://blog.easy-designs.net/archives/2011/11/16/on-adaptive-vs-responsive-web-design/

[32] Aaron Gustafson, *Adaptive Web Design*.: Easy Readers, 2011.

[33] Brad Frost. (2012, Aug.) Beyond Media Queries: Anatomy of an Adaptive Web Design. [Online]. http://bradfrostweb.com/blog/mobile/beyond-media-queries-anatomy-of-an-adaptive-web-design/

[34] Compuware. (2011) What users want from mobile. [Online]. http://www.gomez.com/resources/whitepapers/survey-report-what-users-want-from-mobile/

[35] AGILE ALLIANCE. AGILE ALLIANCE. [Online]. http://www.agilealliance.org/

[36] Wikipedia. Scrum (development). [Online]. http://en.wikipedia.org/wiki/Scrum_(development)

[37] Wikipedia. User story. [Online]. http://en.wikipedia.org/wiki/User_story

[38] Wikipedia. Website wireframe. [Online]. http://en.wikipedia.org/wiki/Website_wireframe

[39] Balsamiq Studios, LLC. Balsamiq. [Online]. http://www.balsamiq.com/products/mockups

[40] Ariel Schiavoni. arielschiavoni/wast · GitHub. [Online]. https://github.com/arielschiavoni/wast

[41] Ariel Schiavoni. Wast. [Online]. http://ec2-23-23-106-149.compute-1.amazonaws.com/

[42] Joyent, Inc. Node.js. [Online]. http://nodejs.org/

[43] express. [Online]. http://expressjs.com/

[44] Guillermo Rauch. Socket.IO: the cross-browser WebSocket for realtime apps. [Online]. http://socket.io/

[45] 10gen, Inc. MongoDB. [Online]. http://www.mongodb.org/

[46] The Apache Software Foundation. Apache Lucene - Apache Solr. [Online]. http://lucene.apache.org/solr/

[47] Wikipedia. JSONP. [Online]. http://en.wikipedia.org/wiki/JSONP

[48] Jeremy Ashkenas. Backbone.js. [Online]. http://backbonejs.org/

[49] RequireJS. [Online]. requirejs.org

[50] handlebarsjs.com. Handlebars.js. [Online]. http://handlebarsjs.com/

[51] Font Awesome. [Online]. http://fortawesome.github.io/Font-Awesome/

[52] Grunt: The JavaScript Task Runner. [Online]. http://gruntjs.com/

[53] Modernizr. [Online]. http://modernizr.com/

[54] JSHint Community. JSHint, a JavaScript Code Quality Tool. [Online]. http://www.jshint.com/