Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

DESARROLLO DE APLICACIONES WEB ABIERTAS PARA LA PLATAFORMA FIREFOX OS

TRABAJO FIN DE MÁSTER

Omar Alejandro Sotelo Torres

2013

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

DESARROLLO DE APLICACIONES WEB ABIERTAS PARA LA PLATAFORMA FIREFOX OS

Autor
Omar Alejandro Sotelo Torres

Director
Santiago Pavón Gómez

Departamento de Ingeniería de Sistemas Telemáticos

2013

Resumen

Smartphones cada vez más potentes, con mejores funcionalidades y a un menor costo, hacen que el número de aplicaciones para estos dispositivos crezca de forma exponencial, tanto para uso personal como corporativo. Actualmente, el mercado de la telefonía móvil se encuentra dominado por los sistemas operativos Android de Google e iOS de Apple. Estos sistemas operativos han destacado entre los otros debido a su rendimiento, prestaciones y demás características que incentivan a los usuarios a su uso.

Al igual que para la mayoría de los sistemas operativos móviles que existen hoy en día, para estos sistemas el desarrollo de las aplicaciones se hace generalmente de forma nativa, es decir aplicaciones programadas específicamente para cada sistema operativo, haciendo uso de APIs, librerías, kit de desarrollo y lenguajes de programación definidos por las compañías propietarias.

EL surgimiento de HTML5 y la búsqueda de la estandarización de las aplicaciones para dispositivos móviles han impulsado el surgimiento de propuestas de desarrollo hibridas o puramente web, promoviendo por parte de Mozilla la idea de llevar las tecnologías web abiertas a los terminales móviles a través de un sistema operativo construido exclusivamente haciendo uso de estas, dando originen a Firefox OS.

En este trabajo se presenta un estudio de las principales características del sistema operativo Firefox OS, así como de los recursos, herramientas y APIs ofrecidas para el desarrollo de aplicaciones web abiertas para esta plataforma. Presentado a forma de tutorial con el fin de que sirva como una guía para un usuario con conocimientos básicos de las tecnologías web y móviles, que desee desarrollar aplicaciones para este sistema operativo.

Abstract

Smartphones becoming more powerful, with better functionality at a lower cost, make the number of applications for these devices grow exponentially for both personal and corporate use. Currently, the mobile phone market is dominated by Google's Android and Apple's iOS. These operating systems have been prominent among the others due of its performance, services, and other features that encourage users to use.

As well as for others mobile operating systems that exist today, for these two systems application development is usually done natively, i.e. applications written specifically for each operating system, using APIs, libraries, development kits and programming languages defined by the owner companies.

The emergence of HTML5 and the search for the standardization of applications for mobile devices through hybrid proposals or purely web, promoted by part of Mozilla the idea to bring the open web technologies to mobile terminals through an operating system built exclusively by making use of these, giving originate in this way to Firefox OS.

This paper presents a study of the main features of Firefox OS as well as the resources, tools and APIs offered for the development of open web applications for this platform. Structured as a tutorial in order to serve as a guide for a user with basic knowledge of the web and mobile technologies that want to develop applications to this operating system.

Índice general

Índice de figuras

Índice de Tablas

Siglas

API: Application Programming Interface

App: Application

B2G: Boot to Gecko

BFFOS: Building Firefox OS

CPU: Central Processing Unit

CSP: Content Security Policy

CSS: Cascading Style Sheets

FXOS: Firefox OS

HAL: Hardware Abstraction Layer

HTML: HyperText Markup Language

ICC: Integrated Circuit Card

JS: JavaScript

JSON: JavaScript Object Notation

MMS: Multimedia Messaging System

SMS: Short Message System

SVG: Scalable Vector Graphics

UI: User Interface

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

W3C: World Wide Web Consortium

WHATWG: Web Hypertext Application Technology Working Group

WIFI: Wireless Fidelity

XML: eXtensible Markup Language

1 Introducción

El mercado de los dispositivos móviles adquiere cada día mayor relevancia a nivel mundial. La evolución tecnológica de los equipos terminales, el incremento de sus funcionalidades y la reducción de los costos, han favorecido su proliferación. Un componente importante y que marca tendencia a la hora de adquirir un terminal por parte de un usuario es su sistema operativo, siendo *Android* de *Google* e *iOS* de *Apple* los que dominan actualmente el mercado.

Al igual que para la mayoría de los sistemas operativos móviles que existen hoy en día, para estos el desarrollo de las aplicaciones se hace generalmente de forma nativa, es decir aplicaciones programadas específicamente para cada sistema operativo, haciendo uso de APIs, librerías, kits de desarrollo y lenguajes de programación definidos por las compañías propietarias. Desde la perspectiva de los desarrolladores de aplicaciones móviles, esta amplia variedad de terminales y la heterogeneidad de sus características, de la mano con el desarrollo de forma nativa implican un amplio y profundo conocimiento de cada uno de los lenguajes y los sistemas operativos en los que se desee desplegar una aplicación, así como la necesidad de desarrollar como mínimo una versión de la aplicación para cada plataforma.

En contra parte a este modelo, gracias al surgimiento y la rápida evolución de tecnologías web como HTML5, así como la necesidad de estandarizar las aplicaciones móviles para permitir su funcionamiento en múltiples plataformas, nacen modelos de desarrollo de aplicaciones móviles web (basados en HTML5) o híbridos (nativo + web), basados en *Frameworks* y librerías que permiten la creación y el funcionamiento de una misma aplicación en varias plataformas y terminales.

Partiendo de la idea de llevar las tecnologías web abiertas a los terminales móviles, *Mozilla* respaldada por fabricantes de dispositivos y empresas del sector de las telecomunicaciones, siendo la más relevante *Telefónica*, ha desarrollado un sistema operativo construido exclusivamente haciendo uso de estas tecnologías, denominado *Firefox OS*.

El presente trabajo es el fruto de una investigación de tipo exploratorio acerca del sistema operativo *Firefox OS*, llevada a cabo con el fin de comprender su funcionamiento y aprender a desarrollar aplicaciones para esta plataforma.

1.1 Objetivos

El objetivo general de este trabajo es orientar a un usuario en el desarrollo de aplicaciones para el sistema operativo *Firefox OS.*

Como objetivos específicos se proponen:

- Determinar el estado de madurez del sistema operativo *Firefox OS.*
- Mostrar una serie de recursos y herramientas que simplifican el proceso de desarrollo de aplicaciones.
- Explorar el funcionamiento de las Web APIs ofrecidas por el sistema operativo a través del desarrollo de aplicaciones básicas.
- Crear una guía para el desarrollo de aplicaciones en *Firefox OS.*

1.2 Motivación

La creciente demanda de aplicaciones móviles por parte de los usuarios ha llevado a plantearse diferentes modelos de desarrollo, destacándose principalmente tres modelos o tipos de aplicaciones móviles: nativas, web e híbridas, las cuales poseen sus fortalezas y debilidades, por lo que según los requerimientos del cliente y propios de la aplicación es conveniente hacer uso de alguno en particular.

Estudios recientes de investigadores, puestas en producción de populares aplicaciones como *Linkdin* (hibrida para *iOS* y nativa para *Android*), *Instagram* (hibrida), *Facebook* (versiones nativa y web) entre otras, han demostrado que la incorporación de las tecnologías web al ecosistema móvil es inminente, como se afirma en un informe de la consultora *Gartner* que indica que para el año 2016 el 50% de las aplicaciones móviles corresponderán a aplicaciones hibridas (1).

De esta manera la apuesta de *Mozilla* de crear un sistema operativo exclusivamente basado en tecnologías web abiertas se convierte en una alternativa bastante atractiva para los desarrolladores de aplicaciones, ya que se da paso a un nuevo modelo de desarrollo donde se hace uso del ecosistema web abierto para el desarrollo de aplicaciones "nativas" para la plataforma *Firefox OS,* razón por la cual es de interés la realización de este trabajo.

1.3 Estructura del Documento

El presente trabajo se encuentra estructurado a modo de tutorial con el fin de servir de guía a un usuario con conocimientos básicos en tecnologías web y móviles, que desee desarrollar aplicaciones para el sistema operativo *Firefox OS*. El documento ha sido dividido de la siguiente forma:

- El capítulo I presenta la introducción, los objetivos, la motivación y la estructura general del desarrollo del trabajo.

- En el capítulo II se presenta el marco conceptual, donde se abordan conceptos relacionados a las tecnologías web que utiliza el sistema operativo, necesarios para el entendimiento y desarrollo de las aplicaciones.

- En el capítulo III se presentan los aspectos más importantes del sistema operativo *Firefox OS*.

- En el capítulo IV se explican los componentes básicos de una aplicación, la configuración del entorno de desarrollo y se crea un primer proyecto, el *"Hola Mundo"*.

- Los capítulos V y VI, se centran en el desarrollo de aplicaciones. Explican en detalle las herramientas y librerías más utilizadas para la creación de aplicaciones en *Firefox OS*, así como el funcionamiento de las Web APIs.

- En el capítulo VII se abordan los procesos de distribución y publicación de las aplicaciones en el *Marketplace* la tienda de *Firefox*.

- Finalmente, se presentan una serie de conclusiones obtenidas como fruto del desarrollo de este trabajo.

2 Marco Conceptual

En esta sección del trabajo se recopilan una serie de conceptos importantes para comprender el funcionamiento de las aplicaciones en *Firefox OS*. En primer lugar se presentan las tecnologías web base del sistema operativo; a continuación se introduce el concepto de *Open Web Apps* termino que hace referencia a las aplicaciones desarrolladas con las tecnologías web abiertas, las cuales son soportadas por la plataforma *Firefox OS.*

2.1 Tecnologías Web

El posicionamiento de la web como plataforma universal para el intercambio de información, ha motivado el desarrollo de nuevas tecnologías y la evolución de las ya existentes con el fin de brindar nuevas funcionalidades y mejores servicios a los usuarios.

La incorporación de los dispositivos móviles como actores principales de este ecosistema, ha generado nuevos requerimientos por parte de los usuarios por lo que las aplicaciones deben de rediseñarse para adaptarse a lo que algunos autores denominan "la cuarta pantalla" (2), en esta medida iniciativas como *One Web*[1] y *Responsive Web Design*[2], junto con tecnologías web como HTML5, CSS3 y JavaScript han permitido la construcción de sitios web y aplicaciones altamente funcionales y eficientes, que responden a la necesidad de los usuarios de acceder a la información y los servicios desde diferentes dispositivos, independientemente del tipo de pantalla, sistema operativo y demás características del equipo terminal.

En este trabajo son de especial interés las tecnologías Web HTML5, CSS3, JavaScript y JSON, ya que son el núcleo para el desarrollo de aplicaciones para el sistema operativo *Firefox OS.*

2.1.1 HTML (Hyper Text Markup Language)

HTML es un lenguaje de marcado estandarizado por el *World Wide Web Consortium (W3C)* y el *Web Hypertext Application Technology Working Group (WHATWG),* utilizado para escribir y estructurar el contenido de un documento web, de forma que sea legible por el navegador. HTML separa el contenido de la presentación haciendo uso de una

[1] Hace referencia a la idea de construir una Web para todos (*Web for All*) y accesible desde cualquier tipo de dispositivo (*Web on Everything*).

[2] Técnica de diseño y desarrollo web que permite adaptar la aplicación al dispositivo.

serie de elementos predefinidos, los cuales se conforman por *tags* o etiquetas, encargadas de describir el contenido del documento.

Una etiqueta está conformada por una palabra clave y los símbolos menor que ("<") y mayor que (">"), generalmente usados en pares para indicar el inicio y el fin de un elemento, y entre las cuales se encuentra ubicado el contenido. La etiqueta inicial puede contener información adicional denominada atributos, los cuales se especifican a través de una pareja nombre/valor y son utilizados para definir las propiedades. La Fig. 1 muestra un modelo básico de la estructura de una página HTML.

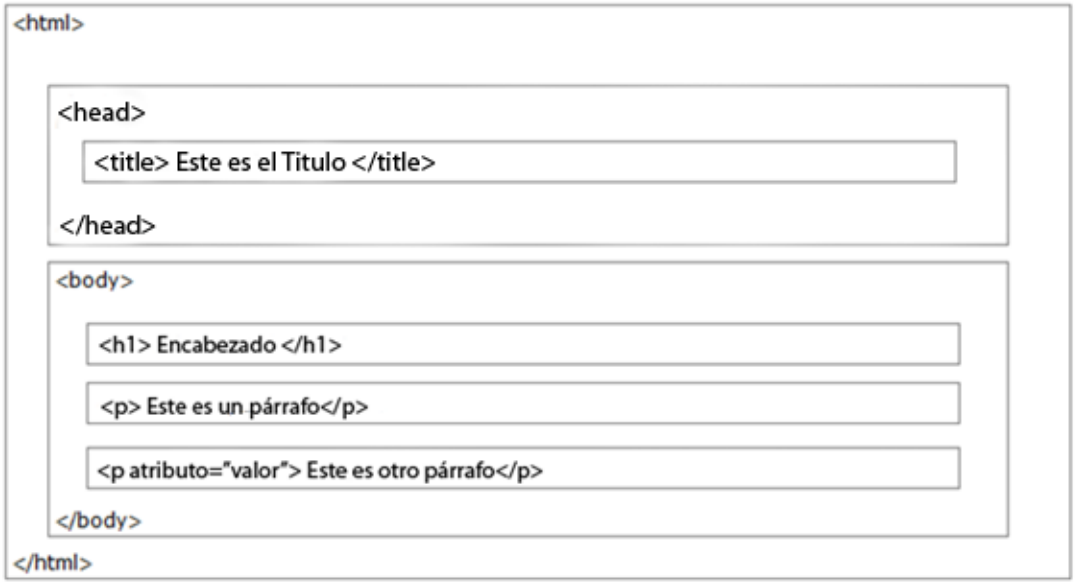

Figura 1. Estructura básica de una página HTML

La especificación más reciente de este lenguaje se conoce como HTML5, la cual a diferencia de sus antecesores no describe el estilo y formato del contenido, solo se limita a definir el propio contenido y su significado, la parte restante (estilo y formato) es definida y controlada haciendo uso de hojas de estilo o *Cascading Style Sheets (CSS)*. El hecho de separar el contenido de la presentación incrementa la eficiencia del código, facilita el mantenimiento y la accesibilidad de los sitios web, así como la adaptabilidad y compatibilidad en diferentes dispositivos.

Entre las principales características de la especificación se encuentran el soporte para el almacenamiento local, soporte SVG (*Scalable Vector Graphics)*, nuevos tipos de datos, y la incorporación de los elementos *Canvas, Audio* y *Video* que permiten dibujar y añadir elementos multimedia directamente en la página, reduciendo la necesidad de hacer uso de *plugins* externos.

2.1.2 CSS (Cascading Style Sheets)

Las hojas de estilo en cascada son un mecanismo utilizado para definir como se muestran los elementos HTML. Los estilos se definen en archivos con extensión CSS, el hecho de tener el estilo independiente del contenido, permite su reúso en diferentes archivos HTML, así como la modificación del aspecto visual de una o varias páginas web solo con modificar este archivo, ofreciendo a los desarrolladores control total sobre el estilo y el formato de los documentos.

Los estilos fueron añadidos en la versión 4 de HTML con el fin de simplificar y reducir el tamaño de los documentos, ya que en versiones anteriores como HTML 3.2 el uso de atributos en las etiquetas para definir propiedades como color, tamaño y tipo de letra debían ser añadidos en cada una de las paginas, convirtiéndose en un proceso largo y costoso para los diseñadores. La versión actual de las hojas de estilo se conoce como CSS3, la cual viene definida por una serie de módulos, que añaden funcionalidades a las existentes en su antecesora CSS2 manteniendo de esta manera la compatibilidad entre las versiones.

Las hojas de estilo se componen por un conjunto de reglas que definen el estilo de uno o más elementos de un documento. Una regla se compone por dos partes: un selector y una declaración; El selector funciona como enlace entre el documento y el estilo, definiendo los elementos en los cuales se aplica la declaración. Esta última, establece el efecto a causar a través de la definición de una propiedad y el valor que se le asigne. La Fig. 2 muestra un ejemplo de la sintaxis utilizada.

Figura 2. Sintaxis de CSS adaptado de (3)

A pesar de que los estilos pueden ser añadidos directamente sobre el documento HTML haciendo uso del elemento "`<style>…</style>`", no se recomienda pues como se mencionó anteriormente la idea es separa el contenido de la presentación. Para vincular una hoja de estilo externa a un documento se hace uso del elemento "`<link>`", el cual debe ir situado en la sección `<head>` como se observa en la Fig. 3.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Titulo</title>
        <link rel="stylesheet" type="text/css" href="http://HojaEstilo.css" />
    </head>
    <body>
    </body>
</html>
```

Figura 3. Vinculo de una CSS a un documento HTML

2.1.3 JavaScript

JavaScript (JS) es un lenguaje de script orientado a objetos, enfocado principalmente a la web por lo que puede ser embebido dentro de las páginas HTML. Es un lenguaje sencillo, ligero y con tipos flexibles de datos (*loosely typed*), lo que significa que las variables no requieren ser definidas como un tipo específico de dato. Sintácticamente es un lenguaje similar a C, C++ y JAVA con operadores, estructuras de control, sentencias y soporte de tipos de datos de la misma naturaleza.

JavaScript puede ser utilizado tanto en aplicaciones cliente como en el servidor; en el lado cliente ha sido utilizado principalmente en los browser, donde su propósito general es proveer dinamismo y extender las funcionalidades de lenguajes como HTML a través de scripts, permitiendo de esta manera una mejor interacción del usuario con las páginas. En otras palabras, mientras que HTML se utiliza para almacenar y dar formato a una página web, CSS para dar el estilo de presentación, JavaScript es usado para crear aplicaciones web enriquecidas.

En un documento HTML, es posible hacer uso de JavaScript insertando el código directamente en el documento, o desde un archivo externo con extensión JS el cual contiene el código a utilizar. En el primer caso el código debe ser insertado entre las etiquetas "`<script>…<script>`", siendo posible utilizarlo en la sección `<head>` y/o `<body>.` Para vincular un script externo se debe hacer uso del atributo *src* de la misma etiqueta indicando como valor la ubicación y el nombre del archivo. Es aconsejable ubicar estas funciones en la sección `<head>` o al final de la página con el fin de separarlas del contenido. La Fig. 4 muestra un ejemplo de la utilización de JavaScript en un documento HTML.

```
<!DOCTYPE html>
<html>
    <head>
    <script src="Script.js"></script>
    </head>
    <body>
        <script>
            document.write("<h1>Esto es Javascript</h1>");
        </script>
    </body>
</html>
```

Figura 4. Utilización de JavaScript en HTML

2.1.4 JSON (JavaScript Object Notation)

JSON es un formato para el intercambio de datos, similar a XML, independiente del lenguaje y la plataforma. Su sintaxis es un subconjunto de la sintaxis de notación de objetos de JavaScript.

Los objetos en JSON, se definen con un par de llaves de apertura y cierre ("{ }"), dentro de las cuales se especifican un conjunto de pares nombre/valor. Cada nombre es seguido por dos puntos (":") y a continuación el valor correspondiente. Las parejas nombre/valor se separan a través de comas (","). La Fig. 5 presenta un ejemplo de esta sintaxis.

Figura 5. Sintaxis de JSON

2.2 Open Web Apps

Una *Web App* es una versión de la web optimizada para su funcionamiento y visualización en dispositivos móviles, desarrollada con tecnologías web abiertas (HTML5, CSS3, JavaScript…), que se puede ejecutar en cualquier equipo terminal que cuente con un navegador, sin importar el sistema operativo del que este disponga.

Una *Web App* pura presenta una serie de limitaciones, como el hecho de ser una aplicación accesible exclusivamente a través de la web, por lo que el usuario deberá contar con una conexión a internet siempre que desee hacer uso de esta, además al no ser una aplicación nativa algunas de las funcionalidades propias del dispositivo no estarán disponibles. Otro factor en contra, es el hecho de que este tipo de aplicaciones

no cuentan con una tienda de aplicaciones para su distribución, siendo más difícil llegar a los usuarios finales.

Bajo la sentencia "la web es la plataforma", *Mozilla* promueve el desarrollo de un tipo de aplicaciones basadas en la web, denominadas *Open Web Apps*, las cuales son aplicaciones similares a las *Web Apps*, pero a diferencia de estas, pueden ser instaladas en los dispositivos, permitiéndoles funcionar sin necesidad de una conexión a internet y hacer uso del almacenamiento local, además del acceso a las características propias del terminal y su distribución a través de las tiendas de aplicaciones o a través de servidores propios.

Una característica importante de las *Open Web Apps* es que requieren de un *Web Runtime* para su ejecución, el cual es similar a un navegador web pero totalmente invisible es decir sin el marco, los botones y barras de herramientas típicas.

Las Open Web Apps se dividen en dos categorías *Packaged Apps* (aplicaciones empaquetadas) y *Hosted Apps* (aplicaciones alojadas). La diferencia principal entre estas dos es que el primero consiste en un archivo de extensión ZIP que contiene todos los recursos de la aplicación, mientras que el segundo hace referencia a las aplicaciones que se ejecutan desde un servidor en un determinado dominio.

3 Introducción a Firefox OS

En este capítulo se abordan los aspectos más importantes del sistema operativo *Firefox OS*. Partiendo de la definición de lo que es la plataforma, seguido de su historia y evolución, a continuación se presenta su modelo de negocio, su nicho de mercado, se explica su arquitectura, principales componentes y tipos de aplicaciones. Finalmente se realiza una introducción a la tienda de aplicaciones de *Mozilla* el *Marketplace*.

3.1 ¿Qué es Firefox OS?

Firefox OS es un sistema operativo orientado a dispositivos móviles basado en *Linux* y en el motor de renderizado Gecko. El sistema operativo fue desarrollado por *Mozilla Foundation* con el apoyo de importantes empresas del sector de las Telecomunicaciones principalmente *Telefónica*, con el objetivo de ofrecer un entorno móvil dedicado a aplicaciones desarrolladas por completo haciendo uso de tecnologías web abiertas como HTML, CSS y JavaScript.

3.2 Surgimiento de Firefox OS

La historia del sistema operativo *Firefox OS* empieza en el 2011, año en el cual Mozilla basado en la idea de que las tecnologías web abiertas podían reemplazar los sistemas propietarios para el desarrollo de aplicaciones, inicia con el proyecto *Boot to Gecko (B2G)* (4) bajo la dirección del Dr. Andreas Gal y soportada por la comunidad de desarrolladores.

En julio de 2012 el proyecto *B2G* pasó a llamarse *Firefox OS* y fue presentado al mundo en el *Mobile World Congress 2013* llevado a cabo en la ciudad de Barcelona. A partir de ese momento aparecieron en el mercado los primeros dispositivos para desarrolladores, comercializados por la empresa *Geeksphone* conocidos como *Keon* y *Peak*. El 1 de Julio de 2013 se realizó oficialmente el lanzamiento del *ZTE Open* en España, el primer dispositivo móvil destinado a usuarios finales con *Firefox OS* como sistema operativo, comercializado por *Telefónica*; se espera que en el transcurso del presente año, dispositivos de otros fabricantes con *Firefox OS* salgan a la venta, principalmente en mercados emergentes como Brasil, Colombia, Venezuela, India y China.

3.3 Firefox OS en el Mercado

El mercado de los dispositivos móviles adquiere cada día mayor importancia a nivel mundial, actualmente se encuentra dominado por los sistemas operativos *Android* de *Google* e *iOS* de *Apple*. Según un informe de la consultora *IDC* (5) en el primer trimestre del 2013 estos dos sistemas operativos consiguieron un 92.3% del mercado total de *Smartphones*.

A pesar de este dominio en el mercado, donde aparentemente no hay cabida para un competidor más, *Firefox OS* apuesta por los mercados emergentes donde los *Smartphones* no han tenido gran penetración debido a su alto coste; a través de dispositivos de "gama media-baja", con buenas prestaciones a precios bajos. Este modelo de negocio se fundamenta en cubrir la brecha existente en estos mercados y poco a poco ir evolucionando hacia dispositivos de gamas más altas para en un futuro poder competir de igual a igual con los grandes sistemas.

A pesar de ser un sistema operativo joven e inmaduro, *Firefox OS* se perfila como un fuerte competidor, debido a su versatilidad, bajos requerimientos hardware y filosofía *Opensource,* acompañado del gran esfuerzo que actualmente están realizando los desarrolladores para su continuo mejoramiento y adaptación a los nuevos estándares tecnológicos, lo que se evidencia en la política de actualizaciones propuesta por *Mozilla* (actualizaciones del sistema operativo cada tres meses y actualizaciones de seguridad cada 6 semanas). A la fecha del presente trabajo son pocos los terminales móviles presentados con este sistema operativo, la mayoría de ellos exclusivos para desarrolladores, pero se espera que en los próximos meses se comience a inundar el mercado. Entre los terminales que se han anunciado, además de los teléfonos móviles y tabletas de diversos fabricantes como *Alcatel, ZTE, Huawei, LG, Sony* y *Geeksphone* se destacan dispositivos que amplían las posibilidades de expansión del sistema operativo, evidenciando la existencia de nuevos mercados, como lo son el *Smartwatch Bambook* de la empresa China *Shanda*, así como una serie de productos que serán desarrollados por la empresa *Foxconn*, entre los que se encuentran *Smartphones*, Tabletas, *Smart TVs*, Pizarras Electrónicas y Pantallas para exteriores. Como se aprecia el abanico de oportunidades para este naciente sistema operativo es bastante amplio.

3.4 Arquitectura de Firefox OS

Firefox OS cuenta con una arquitectura modular compuesta por tres capas bien definidas y relacionadas entre sí, encargadas de la gestión de los servicios y los recursos del dispositivo (parte hardware), soportar y ejecutar las tecnologías web y proveer la interfaz de usuario. La Fig. 6 muestra la arquitectura del sistema operativo, es importante mencionar que al encontrarse la plataforma aun en desarrollo esta

arquitectura no es definitiva pero es una buena referencia para entender su funcionamiento.

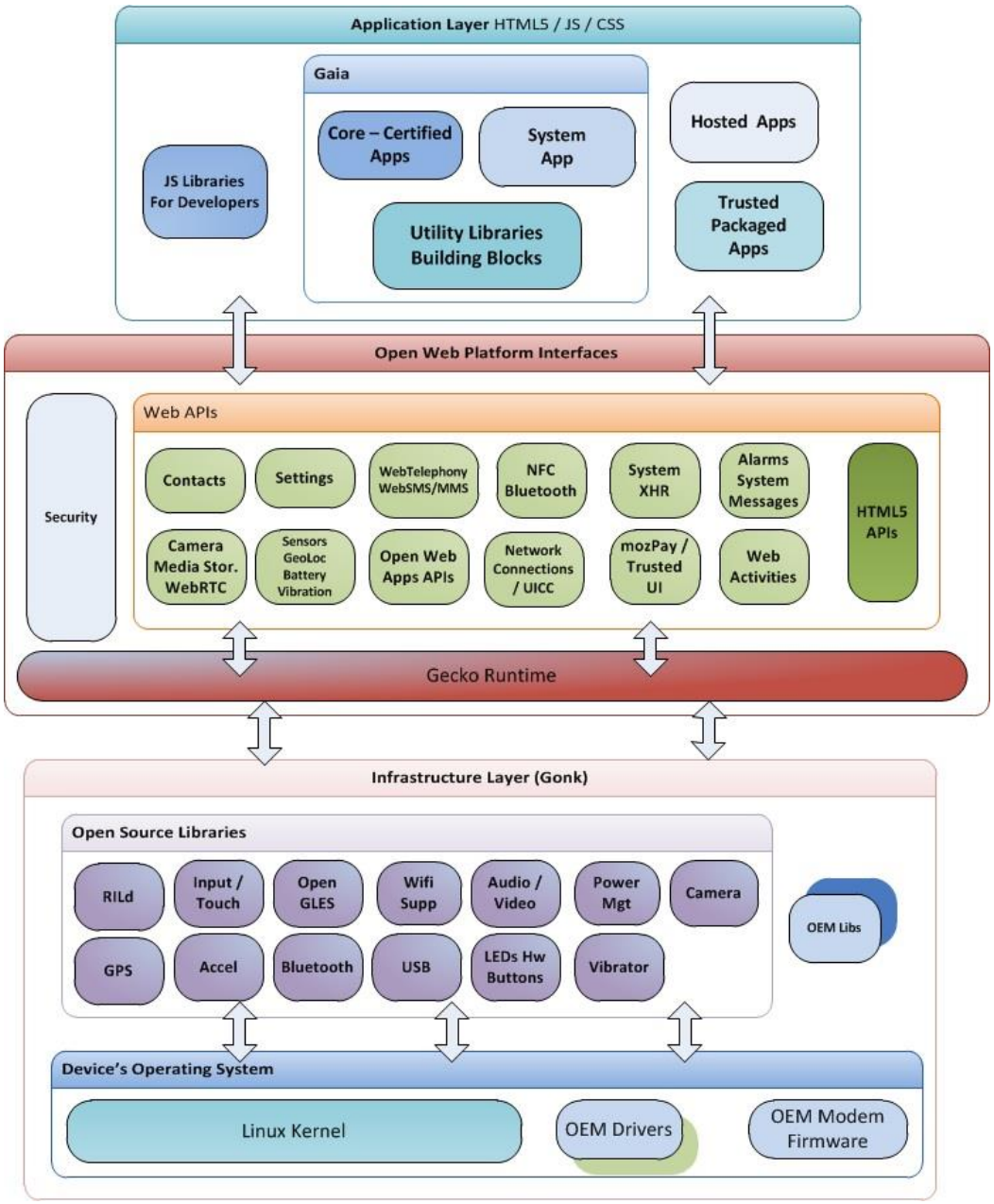

Figura 6. Arquitectura de Firefox OS (6)

- **Capa de Infraestructura (Gonk):** Esta es la capa de más bajo nivel del sistema operativo. Se encuentra compuesta básicamente por un *kernel Linux*, y una capa de abstracción de hardware (HAL) que consta de un conjunto de drivers y librerías para la manipulación de los componentes hardware del terminal. El *kernel*, así como la mayoría de las librerías utilizadas son proyectos *Open Source*, de igual manera algunas de las partes de HAL son compartidas con *Android* (6)

para poder hacer uso de algunas características del teléfono como el GPS y la Cámara.

- **Interfaces Abiertas de la plataforma Web (Gecko):** Esta capa intermedia, consta de un conjunto de APIs implementadas sobre el entorno de ejecución Gecko. El cual provee todo el soporte necesario a estándares abiertos como HTML5, CSS3 y JavaScript, garantizando de esta manera que cualquier tecnología utilizada en los navegadores web, pueda ser ejecutada en este sistema.

- **Capa de Aplicación (Gaia):** La capa de aplicación es la encargada de proveer la interfaz de usuario del sistema operativo, su principal componente es Gaia, el cual se encarga de implementar las aplicaciones propias del teléfono como la pantalla de bloqueo, la pantalla home, el menú y de poner en marcha las aplicaciones presentes en el sistema. Esta capa está construida en su totalidad haciendo uso de tecnologías web.

3.5 Tipos de Aplicaciones en Firefox OS

Las aplicaciones soportadas por *Firefox OS* se conocen como *Open Web Apps* y se clasifican en aplicaciones alojadas y empaquetadas.

3.5.1 Aplicaciones Alojadas *(Hosted Apps)*

Una aplicación alojada o *Hosted App* es una aplicación web que se encuentra alojada y desplegada en un servidor web de acceso público, disponible para su utilización o instalación por parte de los usuarios. Una característica importante de este tipo de aplicaciones es que por cuestiones de seguridad no tienen acceso a todas las características del dispositivo ofrecidas a través de las Web APIs.

3.5.2 Aplicaciones Empaquetadas *(Packaged Apps)*

Una aplicación empaquetada es una aplicación web que tiene todos sus recursos contenidos en un archivo ZIP en lugar de tenerlos en un servidor web. Este tipo de aplicaciones a su vez se clasifica en tres categorías de acuerdo al nivel de seguridad y al nivel de acceso a las Web APIs: Apps con privilegios, certificadas o empaquetadas simples.

- **Aplicación Certificada:** una aplicación certifica está habilitada para hacer uso de todas las características del dispositivo a través de las web APIs. Generalmente este tipo de aplicaciones son aprobadas por el operador o el fabricante y por cuestiones de seguridad no están destinadas a aplicaciones de terceras partes ya que tienen acceso por defecto a todas las funciones críticas del sistema.

- **Aplicación con Privilegios:** Este tipo de aplicaciones hacen uso de las Web APIs que no son consideradas como funciones críticas del sistema, sin embargo requieren de un proceso de certificación y firma digital por parte de tienda de aplicaciones, con el fin de garantizar su integridad, funcionamiento y brindar mayor seguridad a los usuarios.

- **Aplicación empaquetada simple:** Este tipo de aplicaciones no hacen uso de Web APIs sensibles, por lo que no requieren de ningún tipo de certificación especial, por parte de la tienda de aplicaciones.

3.6 Firefox MarketPlace

El *Firefox Marketplace* es una tienda en línea destinada a la distribución de las aplicaciones web construidas usando HTML5, compatibles con diferentes plataformas entre ellas *Firefox OS*. Similar a otras tiendas como el *AppStore* y *GooglePlay*, el *Marketplace* organiza las aplicaciones por categorías, de acuerdo al tipo de aplicación y a la popularidad de estas entre los usuarios, entre las cuales se pueden encontrar aplicaciones gratuitas como de pago.

Un aspecto a resaltar de la tienda de aplicaciones es que el registro para los desarrolladores es totalmente gratuito y que la distribución de aplicaciones por parte de estos debe pasar por un proceso de certificación, de acuerdo a las características propias de cada aplicación.

4 Iniciando a programar

Las aplicaciones para *Firefox OS*, se desarrollan haciendo uso de las tecnologías web presentadas en el segundo capítulo. Esta sección del documento pretende orientar al desarrollador en la configuración del entorno de programación, así como mostrar la estructura básica de una aplicación para este sistema operativo y la configuración de los archivos necesarios para su correcto funcionamiento.

4.1 Configurando el entorno de programación

Como en todo proceso de programación, a la hora de desarrollar una aplicación es necesario contar con un entorno de desarrollo, para realizar los procesos de codificación y pruebas. En el caso de *Firefox OS*, este entorno se compone principalmente de 2 piezas.

En primer lugar un editor de texto para la creación del código, se recomienda el uso de *Sublime Text*[3] o *Notepad++*[4] ya que estos editores están orientados al proceso de codificación, o si se prefiere se puede hacer uso de cualquier IDE con soporte para desarrollo web.

El siguiente componente requerido es un simulador del sistema operativo, necesario para probar las aplicaciones. Para este fin, *Mozilla* incorpora en las diferentes versiones de su navegador *Firefox* (Estándar, Nightly y Aurora) a modo de complemento un simulador del entorno *Firefox OS*. Para instalarlo es necesario acceder a la sección *complementos* del menú del navegador y buscar e instalar el complemento *Firefox OS Simulator* como se observa en la Fig. 7. Una vez instalado, se creara un acceso a este en la sección *herramientas para el desarrollador web* del navegador.

Figura 7. Instalación del Simulador de Firefox OS

[3] Disponible para su descarga en http://www.sublimetext.com/
[4] Disponible para su descarga en http://notepad-plus-plus.org/

4.2 Componentes básicos de una aplicación

Para el desarrollo de una aplicación es necesario fijar una estructura de carpetas y archivos que permita la interpretación del sistema operativo, así como la estandarización de la organización interna de la aplicación para su posterior distribución en el *Marketplace*. Al igual que en el desarrollo de una aplicación web, el punto de partida para la creación de una aplicación en *Firefox OS* es la creación del directorio raíz, el cual va a contener todos los elementos que componen la aplicación.

Una vez establecido este directorio, se debe proceder a la creación de los archivos y carpetas, que varía según la aplicación que se esté desarrollando, pero que debe ajustarse a una plantilla estándar, la cual debe contener como mínimo los siguientes ficheros:

- **index.html:** este archivo se debe encontrar en el directorio raíz, ya que corresponde a la página principal de la aplicación, la cual contiene entre otras cosas, enlaces internos a las hojas de estilo, ficheros JavaScript y archivos multimedia.

- **manifest.webapp:** al igual que el index.html debe estar ubicado en el directorio raíz de la aplicación. El manifest.webapp es el componente más importante de la aplicación, está escrito en formato JSON y contiene toda la información relevante de la aplicación. La información contenida en el archivo se estructura en tres grupos: configuración de la aplicación (nombre, descripción, versión), configuración de los datos del desarrollador (nombre, pagina web) y los permisos requeridos para acceder a los recursos del terminal a través de las Web APIs.

La Tabla 1 muestra los posibles campos permitidos en el archivo Manifest, si se desea profundizar en el tema o consultar nuevos campos admitidos de acuerdo a las actualizaciones del sistema operativo se recomienda consultar (7), de los cuales solo son obligatorios para su funcionamiento `name` y `description`, adicionalmente `launch_path` en el caso de las aplicaciones empaquetadas y la información del desarrollador para su publicación en el *Marketplace*. Otros campos son requeridos de acuerdo al tipo de la aplicación.

Una vez creado el archivo manifest.webapp es posible verificarlo a través de un validador online suministrado por Mozilla disponible en https://marketplace.firefox.com/developers/validator , por otro lado el Simulador de Firefox OS realiza la validación del archivo Manifest indicando al desarrollador los posibles fallos y la línea en la que se encuentran, al momento de desplegar una aplicación.

Tabla 1. Campos del Manifest.webapp

Configuración de la App	
`activities`	Especifica un conjunto de Web Activities que la App soporta.
`appcache_path`	Especifica el *path* o ubicación del archivo Manifest del appcache.
`chrome`	Agrega una interfaz de navegación en la parte inferior de la pantalla. Únicamente disponible a partir de Firefox OS 1.1.
`csp`	Especifica la política de seguridad de la aplicación.
`default_locale`	Especifica el idioma por defecto que se utiliza en el manifest.webapp, entre los valores aceptados se encuentra "en", "es", "fr".
`description`	Especifica una descripción entendible por los humanos de la App.
`fullscreen`	Especifica si se debe lanzar la aplicación a pantalla completa o no, los posibles valores para este campo son `true` o `false`.
`icons`	Especifica el path y la resolución de los iconos de la App. Para subscribir una App en el *Marketplace* se requiere al menos un icono de 128 px.
`installs_ allowed_from`	Especifica los lugares desde los cuales es posible instalar la aplicación.
`launch_path`	Especifica el path desde el cual se lanza la aplicación.
`locales`	Este campo permite especificar diferentes idiomas para la información contenida en el Manifest de acuerdo a la localización.
`messages`	Especifica los mensajes del sistema que son capturas por la App, como notificaciones de mensajes entrantes y llamadas.
`name`	Especifica un nombre entendible por los humanos para la App, la longitud máxima debe ser de 128 caracteres.
`orientation`	Un vector que define las orientaciones en las que se bloqueará la aplicación, incluso si cambia la orientación del dispositivo
`origin`	Especifica el origen de la aplicación, utilizado exclusivamente para aplicaciones privileged o certified. Únicamente disponible a partir de Firefox OS 1.1
`redirects`	Utilizado exclusivamente para aplicaciones privileged o certified, permite redirigir la aplicación para realizar la autenticación externamente. Únicamente disponible a partir de Firefox OS 1.1
`type`	Define el tipo de la aplicación, los valores posibles son web, privileged y certified, si no se especifica el valor por defecto es web.
`version`	Define la versión actual del archivo Manifest.
Configuración de los datos del desarrollador	
`developer`	Especifica la información relacionada al desarrollador de la App, este campo es obligatorio para subscribir una App en el *Marketplace*. Este campo incluye los subcampos `name` y `url`.
Permisos requeridos	
`permissions`	Define el conjunto de los permisos que la App necesita para acceder o controlar las APIs del dispositivo. Este campo incluye los sub campos `description` (especifica la intención detrás de la solicitud) y `access` (especifica el tipo de acceso requerido), este último puede tomar los valores `readonly`, `readwrite`, `readcreate` y `createonly`.

Gracias a HTML5 existe la posibilidad de añadir al proyecto un segundo archivo manifest, denominado **manifest.appcahe**, que permite habilitar el uso de características de la aplicación sin conexión a la red; a través de la definición de los archivos que deben cargarse a nivel local y el uso de la caché del terminal para almacenar los datos.

- **Carpetas para los recursos:** como una buena práctica de programación es recomendable la creación de carpetas destinadas al almacenamiento de los archivos utilizados en la aplicación. Aunque no se especifica una nomenclatura en particular, se aconseja hacer uso de nombres que sean consistentes con su contenido, así por ejemplo las carpetas *css*, *js* e *img* contendrán en su interior las hojas de estilo, los ficheros JavaScript y las imágenes e iconos de la aplicación respectivamente.

4.3 Creación de un Proyecto

Una vez mostrados los componentes básicos de una aplicación, se procederá a la creación de un primer proyecto, con el fin de comprobar el entorno de desarrollo, si se desea la plantilla básica para la construcción del proyecto puede ser descargada de (8), la cual tiene la estructura que se observa en la Fig. 8.

Figura 8. Estructura básica de un proyecto en Firefox OS

En este primer ejemplo se va a desarrollar la aplicación Hola Mundo, esta aplicación básica e introductoria solo requiere de tres elementos: un archivo index.html, un manifest.webapp y un icono.

Archivo Index.html: Este archivo corresponde a la página inicial de la aplicación, aquí se define el contenido a mostrar, en este caso el mensaje "Hola Mundo", el fichero se estructura de la siguiente manera:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hola Mundo</title>
    </head>
    <body>
        <h1>Hola Mundo</h1>
    </body>
</html>
```

Archivo manifest.webapp: En este archivo se define la información básica de la aplicación, se incluyen entre otras cosas la ubicación del icono a utilizar, indicando su resolución.

```
{   "name": "Hola Mundo",
    "description": "Primera aplicación para Firefox OS",
    "version": "0.1",
    "launch_path": "/index.html",
    "icons":{
          "60":"/img/icono_60.png"
            },
   "developer": {
             "name": "Nombre_Desarrollador",
             "url": "http://www.mipagina.com"

                }
}
```

Icono de la Aplicación: El icono es una parte importante aunque no obligatoria, ya que es la carta de presentación de una aplicación, *Firefox OS* ha estipulado que el icono de una aplicación debe ser una imagen con una dimensión de 60x60 pixeles en formato PNG y preferiblemente de forma circular. Con el fin de cumplir estos requerimientos de diseño Mozilla ofrece a los desarrolladores una serie de plantillas descargables de (9) donde adicionalmente se explica a detalle el proceso de diseño. Para esta aplicación se ha diseñado el icono que se observa en la Fig. 9, haciendo uso de la plantilla, y ha sido almacenado en la carpeta *img* con el nombre *icono_60.png*.

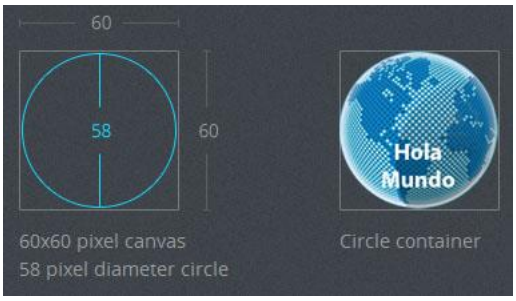

Figura 9. Icono de la Aplicación Hola Mundo. Adaptado de (9)

Con todos los ficheros listos es momento de probar la aplicación, para lo que se abre el navegador, se accede al emulador y se carga el fichero manifest.webapp que se acaba de crear, el resultado obtenido se presenta en la Fig.10.

Figura 10. Aplicación Hola Mundo desplegada en el Simulador de Firefox OS. En la parte superior se observa el administrador de aplicaciones del Simulador, en la parte inferior el emulador y la aplicación desplegada.

5 Herramientas y recursos para la creación de aplicaciones

En la sección anterior del documento, se presentaron los componentes mínimos que conforman el entorno de desarrollo de aplicaciones para Firefox OS, a pesar de que con estos componentes básicos es posible crear aplicaciones de manera adecuada, en la mayoría de ocasiones es necesario la utilización de recursos adicionales como plantillas, librerías y frameworks que faciliten el proceso codificación y pruebas, la reducción de los tiempos de desarrollo y la optimización de las aplicaciones. En este capítulo se recopilan una serie de elementos que enriquecen este ecosistema.

5.1 Herramientas para la creación de Interfaces de Usuario

La interfaz de usuario es uno de los componentes que sin duda marcara el éxito o el fracaso de una aplicación, factores como la estética, la usabilidad y las características de interacción determinan su aceptación o no por parte de los usuarios. Para la creación de interfaces de aplicaciones para *Firefox OS* existen dos alternativas; la primera consiste en basar el desarrollo en el diseño web adaptativo (*responsive web design*), a través del cual se logra que la aplicación se adapte a los diferentes tipos de pantalla de los dispositivos, si se desea realizar este tipo de desarrollo se puede hacer a través de herramientas como *Bootstrap*[5] y *Foundation*[6] o frameworks para el desarrollo multiplataforma, los cuales se presentaran más adelante.

La segunda alternativa se basa en el uso de plantillas desarrolladas por la comunidad de *Mozilla* basadas en Gaia, la principal ventaja de este método es que se dota a las aplicaciones de un aspecto "nativo", ya que se construyen con los mismos elementos (temas, colores, iconos, botones) que el sistema operativo. A continuación se presentan una serie de proyectos con recursos útiles para la creación de las interfaces de las aplicaciones.

5.1.1 Building Firefox OS

El proyecto Building Firefox OS (BFFOS) se originó con la idea de crear un conjunto de componentes HTML/CSS reutilizables con el fin de acelerar el desarrollo de las aplicaciones. El proyecto se puede descargar desde el repositorio oficial de Gaia en Github, o desde su página oficial[7], en la cual se explica a detalle cada uno de los elementos disponibles, su implementación y el resultado obtenido. Dentro del proyecto se definen tres grupos de componentes: *Building Blocks, CSS Transitions* e *Icons.*

[5] http://getbootstrap.com/2.3.2/
[6] http://foundation.zurb.com/
[7] http://buildingfirefoxos.com/index.html

- **Building Blocks:** corresponden a una serie de componentes reutilizables que definen los principales elementos de interacción, como los controles básicos, menús y barras de herramientas. La tabla 2 muestra los elementos disponibles.

Tabla 2. Componentes definidos en la sección Building Blocks de BFFOS

Elemento	Descripción
Action Menu	Presenta una lista de acciones relacionadas al contenido de la aplicación, para su selección por parte del usuario.
Buttons	Los Botones desencadenan una acción al ser presionados por el usuario, Firefox OS ofrece una amplia variedad de estilos de acuerdo a el contexto de la aplicación (Botones por defecto, deshabilitados, listas de botones)
Confirm	Este elemento es utilizado para confirmar una acción o informar al usuario de un evento.
Drawer	Proporcionan acceso a opciones de navegación de nivel superior que pueden ser demasiado numerosas para una interfaz de pestañas o barra de herramientas.
Edit Mode	Permite convertir el contenido en editable por el usuario.
Filters	Permite filtrar la información, así como los elementos de navegación (navegación secundaria).
Headers	Permite definir una etiqueta o título para la vista activa, así como elementos de navegación o entradas.
Input Areas	Definen una serie de elementos para la entrada de texto.
Lists	Es utilizada para mostrar un conjunto de elementos consecutivamente, tales como contactos, mensajes, etc…
Progress and Activity	Permite suministrar al usuario información de manera gráfica acerca del progreso de una acción.
Scrolling	Permite el desplazamiento de la pantalla cuando el número de elementos no se puede mostrar por completo en una sola vista.
Seek Bars	Es un elemento de control que permite desplazarse a través del contenido, o de los posibles valores de una variable.
Status	Enlaces de información al usuario de manera transitoria, por lo general, para confirmar una acción del usuario o para alertar al usuario de un evento del sistema.
Switches	Elementos que funcionan como interruptores que permiten activar o desactivar un elemento determinado o su selección.
Tabs	Permiten la navegación entre diferentes vistas en una sola pantalla.
Toolbars	Las barras de herramientas contienen iconos que representan acciones, indicadores o elementos de navegación correspondientes a la vista activa.
Value Selectors	Permiten a los usuarios seleccionar uno o más valores de un solo campo de texto.

- **CSS Transitions:** proveen una serie de efectos y animaciones a la interfaz de usuario que mejoran la experiencia del usuario al momento de interactuar con la aplicación.

- **Icons:** en esta sección se reúne un conjunto de iconos clasificados de acuerdo a su posible función: comunicaciones, multimedia, acciones primarias y configuración, en la Fig. 11 se presentan algunos ejemplos.

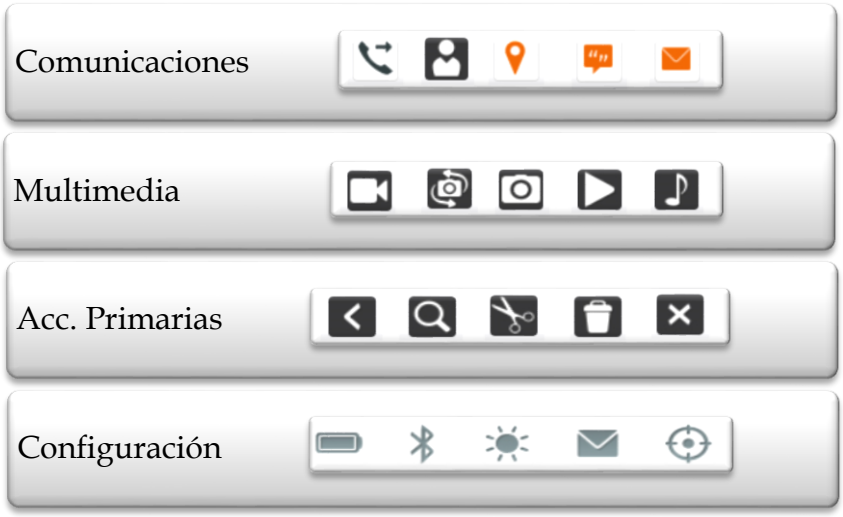

Figura 11. Ejemplos de iconos disponibles en Building Firefox OS

5.1.2 FxOSstub

FxOSstub es un proyecto desarrollado por Pierre Richard orientado a desarrolladores con conocimientos muy básicos en tecnologías web, con el fin de ofrecer un punto de partida hacia el desarrollo de aplicaciones para Firefox OS. El proyecto está disponible para su descarga en (10) y se compone por 2 plantillas en las cuales se recopilan una serie de elementos utilizados comúnmente en las aplicaciones móviles, acompañados de una breve explicación de su utilización y funcionamiento.

La plantilla 1 es la más básica del proyecto (Fig. 12(a)). Cuenta con una estructura de carpetas y archivos muy sencilla compuesta por 4 elementos: los archivos index.html, manifest.webapp y el directorio *style* que contiene la hoja de estilo y los iconos utilizados y el directorio *js* donde se encuentra el archivo *install.js* que contiene el código que permite que la aplicación se pueda instalar desde el botón presente en la aplicación destinado para tal fin. Al ser una plantilla, la aplicación no tiene ninguna funcionalidad, simplemente provee una interfaz con elementos básicos que se pueden adaptar para la construcción de nuevas aplicaciones.

La plantilla 2 es la versión actual del proyecto (Fig. 12(b)) y es una evolución de la plantilla 1 que cuenta con un mayor grado de elaboración e incorpora nuevos

elementos y recursos para la construcción de interfaces, tales como *sidebar*, *menuaction* y *menulist*.

Figura 12. Plantillas FxOSstub. La figura (a) corresponde a la plantilla 1, la figura (b) a la plantilla 2.

5.1.3 Mortar

Mortar es una colección de plantillas y herramientas desarrolladas por la comunidad de *Mozilla* para enriquecer el ecosistema de desarrollo de aplicaciones y brindar un punto de partida a los nuevos desarrolladores basándose en la simplicidad y en buenas prácticas de programación. El proyecto se encuentra disponible para su consulta y descarga en (11), y su objetivo principal es agilizar el desarrollo de aplicaciones a través de la utilización de las plantillas y la simplificación y automatización de tareas.

Cada una de las plantillas de Mortar cuenta con una estructura de archivos y carpetas definida que se puede apreciar en la Fig. 13, archivos iniciales (HTML, JavaScript, Manifest.webapp) bien definidos, e integra algunas librerías y herramientas como *zepto*, *install webapps*, *require.js* y *volo*. Es importante mencionar que en la estructura de archivos definida por Mortar, los archivos estáticos como los HTML, CSS y JavaScript se encuentran dentro del directorio www.

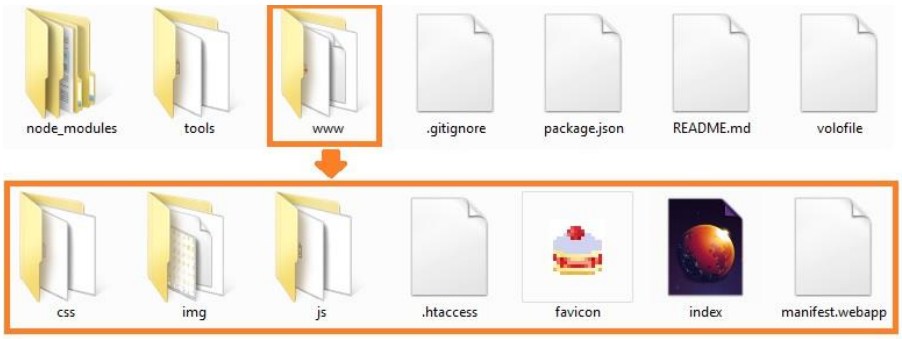

Figura 13. Estructura de carpetas y archivos de una plantilla de Mortar

Actualmente están disponibles para su uso 4 plantillas (Fig. 14), cada una destinada a diferentes tipos de aplicaciones y necesidades, las cuales son:

- **Mortar app stub:** es la plantilla más básica consta únicamente de los elementos necesarios para iniciar a desarrollar una aplicación.
- **Mortar list detail:** esta plantilla ofrece una lista de elementos modificables, similar a una lista de tareas.
- **Mortar tab view:** ofrece una distribución basada en *tabs* (pestañas) y botones que permiten la navegación para visualizar el contenido de las diferentes vistas.
- **Mortar game stub:** una plantilla mínima de juegos que permite manejar eventos de entrada y los muestra en pantalla similar a un control de un videojuego.

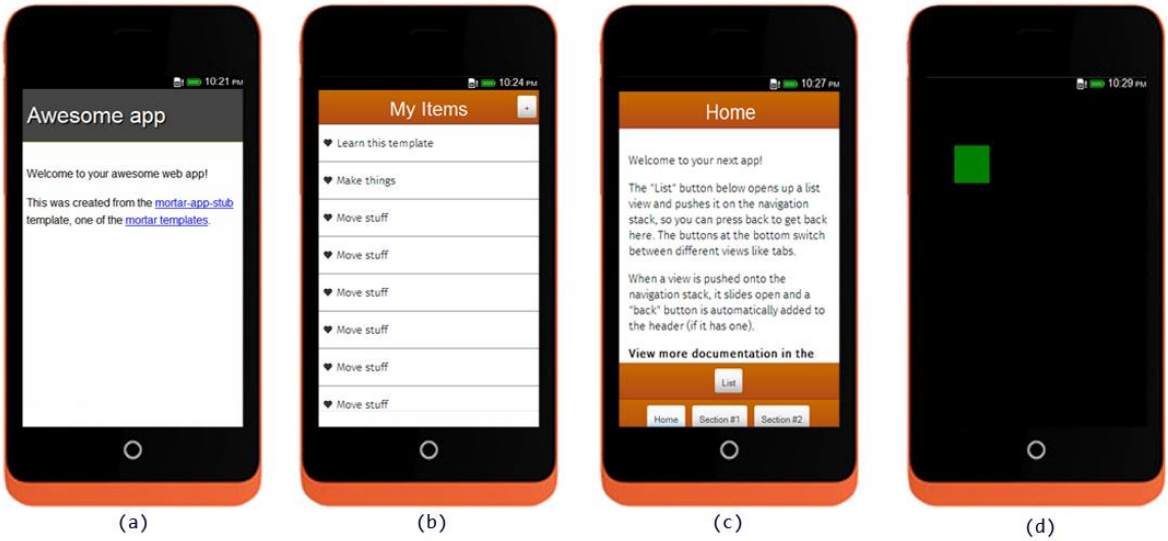

Figura 14. Set de plantillas de Mortar. De izquierda a derecha app stub (a), list detail (b), tab view (c) y game stub (d)

5.2 Práctica 1: Agenda de Eventos

5.2.1 Enunciado práctica 1: Agenda de eventos

En esta sección se propone una primera práctica, que consiste en la realización de una Agenda de citas o eventos. La aplicación debe mostrar en la pantalla principal una lista con los eventos programados y disponer de un botón que permita la creación de nuevos eventos. Al seleccionar uno de los elementos de la lista de eventos se debe desplegar una segunda pantalla en la cual se muestren los detalles del evento y a su vez permita su edición.

Esta práctica se desarrollara haciendo uso de Mortar y los componentes de interfaz presentados anteriormente, con el fin de comprender a mayor detalle su funcionamiento.

5.2.2 Solución practica 1

Para iniciar la práctica es necesario realizar la integración de las herramientas con las que trabaja Mortar al ecosistema de desarrollo; el primer paso será entonces instalar *volo* un servidor basado en *node.js* que facilita el proceso de desarrollo local, la automatización de tareas y la optimización de las aplicaciones para su distribución.

Si no se cuenta con *node.js*, se recomienda seguir el tutorial disponible en (12) donde se explica el proceso de instalación para los diferentes sistemas operativos. Una vez instalado *node.js*, se procede a la instalación de *volo*, a través del comando:

```
npm install -g volo
```

Como se mencionó en la sección anterior Mortar ofrece una serie de plantillas base para el desarrollo de aplicaciones; para la construcción de esta aplicación se utilizara la plantilla *list detail*, ya que es la que mejor se adapta a los requerimientos de la práctica. A través del comando `create` en la línea de comandos se procede a la descarga de la plantilla:

```
volo create FX_Agenda mozilla/mortar-list-detail
```

FX_Agenda corresponde al nombre de la carpeta raíz del proyecto dentro de la cual se va a clonar la plantilla y mozilla/mortar-list-detail corresponde a la plantilla que se desea descargar. Una vez descargada la plantilla el directorio raíz tendrá la estructura de archivos y carpetas mostrada en la Fig. 13.

A continuación se procede a la comprobación del estado actual del proyecto, es decir de la plantilla descargada y de su funcionamiento con el fin de determinar los elementos a modificar (crear, editar o eliminar) para el desarrollo de la aplicación. Es posible realizar la prueba a través del emulador Firefox OS Simulator si desea, o hacer uso del servidor *volo* para desplegar la aplicación en el navegador, esto se hace a través del comando `volo serve` en la consola y accediendo a la dirección http://localhost:8008 en el navegador web, adicionalmente se puede hacer uso de la herramienta vista de diseño adaptable para verificar la visualización de la aplicación en diferentes tamaños de pantalla, como se observa en la Fig. 15.

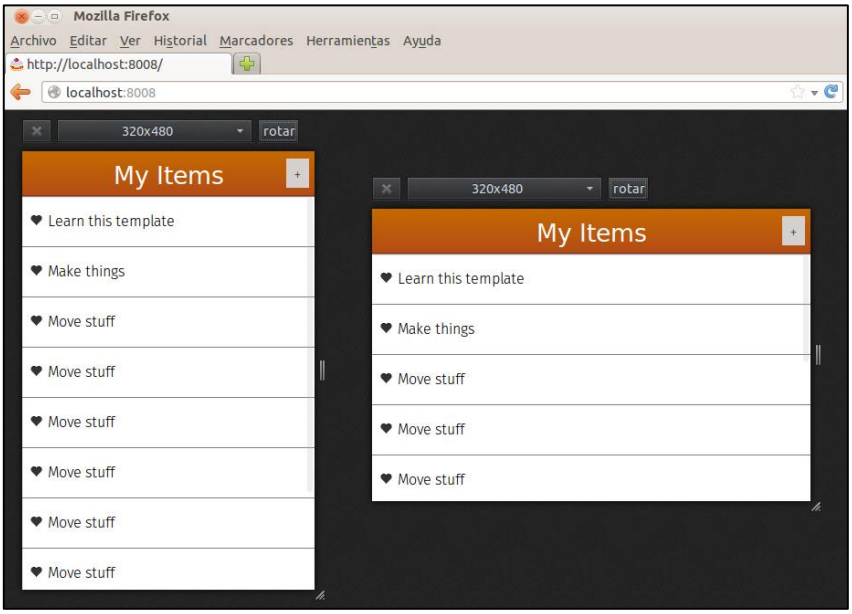

Figura 15. Vista Adaptable de la aplicación desplegada en el servidor Volo

La plantilla requiere de algunas pequeñas modificaciones para adaptarse a los requerimientos de la aplicación propuesta en esta práctica, en primer lugar es necesario eliminar todo el contenido irrelevante como es el caso de los elementos que trae la lista por defecto, en segundo lugar es necesario añadir los elementos *Fecha, Hora de Inicio* y *Hora de Finalización* a la interfaz a través de la cual se crean nuevos eventos y finalmente modificar la interfaz de la vista detallada de los eventos para mostrar todos los detalles del evento.

A continuación se explica el código para la creación de la aplicación.

Index.html

En la sección <head> se encuentran definidos los metadatos y se enlazan las hojas de estilo de la aplicación

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style" content="black">
    <link rel="stylesheet" href="css/app.css">
  </head>
```

En la sección `<body>` se define toda la interfaz de la aplicación la cual ha sido dividida en tres secciones.

En primer lugar se define la vista principal la cual se compone por un elemento `<header>` en el cual se define el título de la sección y un botón que permite la navegación hacia la opción de añadir un evento, adicionalmente será la vista en la cual se cargue la lista de eventos definidos por el usuario.

```
<body>
 <x-listview class="list">
 <header><h1>Agenda</h1>
  <button class="add" data-view=".edit" data-animation="slideDown">+</button>
 </header>
 </x-listview>
```

A continuación se define una segunda vista destinada a visualizar de manera más detallada un elemento seleccionado de la lista de eventos de la pantalla principal. Se define un elemento de tipo `<header>` en el cual se muestra el título de la actividad seleccionada y un botón que permite la edición del evento.

```
 <x-view class="detail">
   <header>
     <h1>Detalles</h1>
     <button data-view="x-view.edit">Editar</button>
   </header>
   <h1 class="title"></h1>
   <p  class="desc"></p>
   <p  class="date"></p>
   <p  class="timebegin"></p>
   <p  class="timeend"></p>
  </x-view>
```

Esta tercera sección corresponde al formulario a través del cual el usuario crea los eventos o los modifica.

```
  <x-view class="edit">
    <header><h1>Editar</h1></header>
    <div class="field">Titulo: <input type="text" name="title" /></div>
    <div class="field">Descripci&oacute;n:<input type="text" name="desc"/>
    </div>
    <div class="field">Fecha: <input type="date" name="date" /></div>
```

```
  <div class="field">Hora Inicio: <input type="Time" name="timebegin"/>
  </div>
  <div class="field">Hora Fin: <input type="Time" name="timeend"/></div>
  <div align="center">
       </br><button type="submit" class="add">A&ntilde;adir</button></div>
</x-view>
```

Finalmente se enlazan los scripts de la aplicación. Mortar hace uso de *require.js*, una librería que permite la carga de los archivos JavaScript en módulos con el fin de optimizar la aplicación, por esta razón solo se define la ubicación de esta librería, los demás archivos JavaScript a utilizar se definen en el archivo app.js.

```
<script type="text/javascript" data-main="js/init.js"src="js/lib/require.js">
</script>
</body>
</html>
```

App.js

En este archivo se definen los archivos JavaScript que hacen parte de la aplicación a través de la función require(), adicionalmente se definen las funciones encargadas del despliegue de la lista de eventos, los detalles de un evento y para añadir/editar un evento.

```
// Vista principal – Carga lista eventos
    var list = $('.list').get(0);

// Vista de un Evento a detalle
    var detail = $('.detail').get(0);
    detail.render = function(item) {
    $('.title', this).html(item.get('title'));
    $('.desc', this).html(item.get('desc'));
    $('.date', this).html("Fecha: "+item.get('date'));
    $('.timebegin', this).html("Hora de Inicio:"+
    item.get ('timebegin'));
    $('.timeend', this).html("Hora de Finalizaci&oacute;n: "+
    item.get('timeend'));
};

// Se precarga el formulario con los datos del evento a editar
    var edit = $('.edit').get(0);
    edit.render = function(item) {
    item = item || { id: '', get: function() { return ''; } };
    $('input[name=id]', this).val(item.id);
    $('input[name=title]', this).val(item.get('title'));
    $('input[name=desc]', this).val(item.get('desc'));
    $('input[name=date]', this).val(item.get('date'));
    $('input[name=timebegin]', this).val(item.get('timebegin'));
    $('input[name=timeend]', this).val(item.get('timeend'));
};

    edit.getTitle = function() {
    var model = this.view.model;
    if(model) {
        return model.get('title');
```

```
        }
        else {
            return 'Nuevo Evento';
        }
    };


    // Añadir un evento
    $('button.add', edit).click(function() {
        var el = $(edit);
        var title = el.find('input[name=title]');
        var desc = el.find('input[name=desc]');
        var date = el.find('input[name=date]');
        var timebegin = el.find('input[name=timebegin]');
        var timeend = el.find('input[name=timeend]');
        var model = edit.model;

        if(model) { model.set({ title: title.val(), desc: desc.val(),
        date:    date.val(),    timebegin:    timebegin.val(),    timeend:
        timeend.val()});
        }else {
          list.add({   title:   title.val(),   desc:   desc.val(),   date:
          date.val(), timebegin: timebegin.val(), timeend: timeend.val()
          });
        } edit.close();
    });
});
```

Manifest.webapp

El archivo Manifest se modifica del que trae la plantilla para adaptarlo a la aplicación.

```
{
  "version": "0.1",
  "name": "Agenda",
  "description": "Aplicación para crear listas de tareas a realizar",
  "launch_path": "/index.html",
  "icons": {
    "60": "/img/icons/logo60.png",
    "128": "/img/icons/logo128.png"
  },
  "developer": {
    "name": "Nombre_Desarrollador"
   },
  "installs_allowed_from": ["*"],
  "default_locale": "es"
}
```

Una vez finalizada la aplicación se hace uso del comando `volo build` con el fin de obtener una versión compilada y optimizada de la aplicación, la cual se creara dentro de un nuevo directorio denominado www-build.

La aplicación resultante es la que se observa en la Fig. 16, cuyo código está disponible para su descarga en (13). Es importante mencionar que aunque no se hizo

uso a fondo de todas las características de *volo* debido a la simplicidad de la aplicación desarrollada, la herramienta permite la adición de otras librerías, la creación automática del archivo appcache, la integración con Github, entre otras características que pueden ser consultadas en su página oficial[8].

Figura 16. Practica 1: Agenda de Eventos

5.3 Frameworks y Librerías

El uso de frameworks y librerías se ha convertido en algo cotidiano en el desarrollo de aplicaciones, especialmente en la web, donde cada día se requiere de aplicaciones más complejas y atractivas para los usuarios. El uso de este tipo de herramientas permite a los desarrolladores la reutilización de componentes, la automatización de tareas y la introducción de funcionalidades concretas de una manera sencilla y adecuada, permitiendo reducir los tiempos de desarrollo y optimizar el rendimiento de las aplicaciones.

Actualmente existen una gran cantidad de frameworks y librerías orientadas al desarrollo de aplicaciones web y móviles, en esta sección se presentan algunas de estas herramientas a tener en cuenta para el desarrollo de aplicaciones para *Firefox OS*.

5.3.1 Frameworks para el desarrollo multiplataforma

Cada vez es más común el uso de los frameworks para el desarrollo multiplataforma ya que son herramientas que permiten el funcionamiento de una misma aplicación en diferentes sistemas operativos. Entre las principales características de este tipo de herramientas se encuentran el soporte multipantalla, el diseño de interfaces de manera gráfica, la generación de pantallas de manera dinámica, el manejo de eventos a alto nivel y la integración de servicios externos.

[8] http://volojs.org/

A pesar de que no es necesario un Framework para el desarrollo de aplicaciones para *Firefox OS*, todos aquellos que permitan el desarrollo en HTML5, CSS3 y JavaScript son compatibles, entre los Frameworks que vale la pena destacar se encuentran **eMobc, Joapp, JQuery Mobile, LungoJS** y **Sencha Touch.**

5.3.2 Librerías orientadas a móviles

Una librería o biblioteca es un módulo de software orientado a suministrar una funcionalidad concreta, por ejemplo realizar una animación, la manipulación de datos o funciones de programación específicas. Como es bien sabido en la programación para dispositivos móviles un factor importante a la hora de desarrollar una aplicación factores como su rendimiento, tiempos de respuesta y tamaño son fundamentales, siendo de suma importancia hacer uso de librerías optimizadas para este fin (menor tamaño, mayor velocidad y óptimo funcionamiento). En la Tabla 3 se presentan una serie de micro librerías que pueden ser de utilidad al momento de desarrollar una aplicación.

Tabla 3. Micro librerías JavaScript para el desarrollo de aplicaciones

Librería	Descripción	Tamaño
Backbone.js	Esta librería brinda una estructura MVC a la aplicación, permite el manejo de eventos personalizados, y el trabajo con servicios Rest.	6.2 Kb
fader.js	Librería para animación que brinda el efecto de fader a los elementos HTML.	0.6 Kb
Janis	Una librería destinada a realizar efectos de transición CSS.	1.4 Kb
Lscache	Librería destinada a la emulación de la función de almacenamiento local en cache de HTML5	1.0 Kb
modernizr	Librería para la detección de las características HTML5 y CSS3 en el navegador del cliente.	4.3 Kb
QuoJS	Es una biblioteca JavaScript que simplifica el manejo de eventos y las interacciones Ajax.	10 Kb
Require.js	Es una biblioteca para el manejo y la carga de los archivos JavaScript por módulos, que permite incrementar la eficiencia del código	14 Kb
swipe	Librería para el control y la transición de imágenes, optimizada para dispositivos táctiles.	1.9 Kb
toast	Librería para la carga de recursos CSS y JavaScript a en tiempo de ejecución.	0.5Kb
Xui	Una biblioteca DOM para la creación de aplicaciones HTML5 móviles.	0.9 Kb
Zepto	Es una biblioteca JavaScript orientado a objetos que simplifica el manejo de eventos, las interacciones Ajax, similar a JQuery.	21 Kb

Si se desea profundizar en alguna de las librerías presentadas o consultar otras opciones se recomienda consultar MicroJS[9] un proyecto de Thomas Fuchs que recopila y cataloga una gran cantidad de librerías optimizadas para su uso en móviles.

5.4 Práctica 2: Calculadora de IMC

5.4.1 Enunciado práctica 2: Calculadora de IMC

En esta sección se propone una segunda práctica, que consiste en la realización de una aplicación capaz de calcular el Índice de Masa Corporal (IMC) de un usuario. Esta aplicación debe recoger los parámetros peso y altura, y retornar el valor calculado a través de la fórmula: peso / (altura x altura), adicionalmente debe mostrar información al usuario acerca de su estado nutricional según el valor de IMC obtenido.

Para la realización de esta práctica se hará uso del Framework de desarrollo Lungo, con el fin de mostrar una de las alternativas para el desarrollo de aplicaciones para Firefox OS presentadas en la sección anterior.

5.4.2 Solución practica 2

Lungo es un framework de código abierto basado en las tecnologías web HTML5, CSS3 y JavaScript que simplifica el diseño y la construcción de aplicaciones multiplataforma, su versión actual y con la cual se desarrolla esta práctica es la 2.2 disponible para su descarga desde su página oficial[10] o desde el repositorio de GitHub[11].

Para iniciar con el desarrollo de la práctica se define el directorio raíz *FX_IMC* que contendrá los componentes de la aplicación, en este caso se hará uso de la estructura básica presentada en la sección 4.3 de este documento. Una vez establecida la estructura de archivos y carpetas, y descargado el framework (archivo ZIP) se procede a su integración, para esto es necesario descomprimirlo y añadir los archivos JavaScript y CSS a el proyecto en sus respectivas carpetas *js* y *css*. Un aspecto importante a tener en cuenta es que Lungo requiere de la librería Quojs[12] para su correcto funcionamiento, por lo que se debe descargar y añadir al proyecto. Una vez realizado este proceso la estructura de la aplicación es la que se presenta en la Fig. 15.

[9] http://microjs.com
[10] http://lungo.tapquo.com
[11] https://github.com/tapquo/Lungo.js
[12] http://quojs.tapquo.com

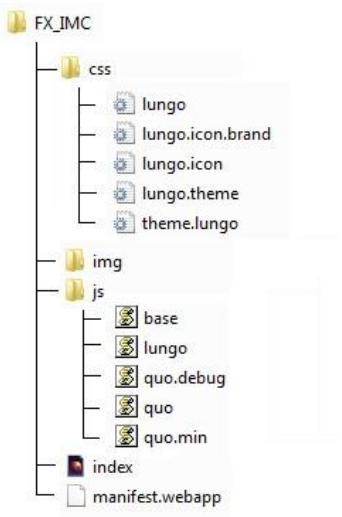

Figura 17. Estructura de carpetas y archivos de la aplicación después de la integración del Framework Lungo y la librería Quojs

Con el entorno de desarrollo listo se procede a la codificación de la aplicación, en primer lugar se creara el archivo principal *index.html* el cual contendrá el formulario para el ingreso de los datos por parte del usuario, los elementos de navegación y todo el contenido referente a la aplicación.

Index.html

El primer paso es enlazar las hojas de estilo en la sección `<head>` del documento:

```
…
<link rel="stylesheet" href="css/lungo.css">
<link rel="stylesheet" href="css/lungo.icon.css">
<link rel="stylesheet" href="css/lungo.icon.brand.css">
<link rel="stylesheet" href="css/theme.lungo.css">
</head>
```

Lungo establece una estructura básica para los documentos html basada en elementos `<section>` y `<article>`, los cuales representan contenedores dentro de los cuales se ubican los demás elementos. En primer lugar se define un elemento de tipo `<section>` el cual funciona como contenedor principal de todos los elementos de la interfaz de usuario:

```
<body>
<section id="main" data-transition="slide">
…
</section>
```

Dentro de `<section>` se define el elemento `<header> que` corresponde a una barra superior donde se indica el título de la sección y en la cual es posible añadir botones para la navegación entre las diferentes vistas, aquí se define como título *Calculadora de IMC.*

```
<header data-title="Calculadora de IMC">
</header>
```

Seguido de esto se definen los elementos `<article>,` que son contenedores más pequeños similares a `<section>` que pueden ser utilizados en una sola vista o representar una vista cada uno, en los cuales se especifica el contenido. La aplicación cuenta con 2 elementos `<article>` que corresponden a una vista cada uno, e identificados por el atributo *id* para permitir la navegación.

La vista principal identificada con el *id="main-article"* y el atributo *class="active"* que indica que debe mostrarse primero, contiene el formulario a través del cual se recogen los parámetros y los elementos necesarios para devolver el resultado al usuario:

```
<article id="main-article" class="active">

<div align="center">
   <img src="/img/imc.png" alt="IMC" width="240" height="70">
</div>
<div class="form">
   <br><label >Introduzca su peso (Kg)</label>
   <input type="number" id="peso"></input>
   <br><label> Introduzca su estatura (cm)</label>
   <input type="number" id="alt"></input>
   <div align="center">
      <buttontype="button" onclick="calcular()">Calcular</button>
      <button  type="button" onclick="limpiar()">Limpiar</button>
      <h1 align="center" id="respuesta"><br>Su IMC es :  </h1>
   </div>
</div>
</article>
```

La vista secundaria identificada con el *id="info-article"*, contiene información acerca del IMC y su relación con el estado nutricional, en la aplicación corresponde a una segunda vista a la cual se accede a través de un elemento de navegación destinado para este fin:

```
<article id="info-article">
…
</article>
```

Una vez definido el contenido de la aplicación, se implementa un elemento <footer>, que contiene 2 botones destinados a la navegación entre las páginas:

```
<footer>
   <nav>
       <a href="#main-article" data-icon="home" class="active" data
        router="article"> </a>
       <a href="#info-article" data-icon="info" data-router="article"></a>
       </nav>
</footer>
</section>
```

Finalmente, después del cierre del elemento `<section>` se enlazan los scripts del Framework y se inicializa Lungo a través de la función JavaScript Lungo.init():

```
<script src="js/quo.js"></script>
<script src="js/lungo.js"></script>
<script src="js/calculo.js"></script>
<script>
      Lungo.init({name: 'Practica 2: Calculadora IMC'});
 </script>
```

Hasta este punto se encuentra construida en su totalidad la interfaz de la aplicación, el resultado obtenido se presenta en la Fig. 18. Las funciones limpiar () y calcular () que son llamadas a través de los botones Limpiar y Calcular se encuentran definidas en el archivo *calculo.js*

Calculo.js

```
function calcular()
    {
    var p = document.getElementById("peso").value;
    var h = document.getElementById("alt").value;

    if(p==0 || h==0)
    {
        alert("Los valores introducidos no son validos");
    } else{
     h = h/100;
     var r = p/[h*h];
     r = r.toFixed(1);
     var estado="Su estado nutricional es ";
     if (r<18.5){
        estado=estado+"bajo de peso";
     }else if (18.5<=r && r<=24.9){
        estado=estado+"normal";
     }else if (25<=r && r<=29.9){
        estado=estado+"sobrepeso";
     }else if (30<=r && r<=34.9){
        estado=estado+"obesidad tipo I";
     }else if (35<=r && r<=39.9){
        estado=estado+"obesidad tipo II";
     }else if (40<=r){
        estado=estado+"obesidad tipo III";
     }
    document.getElementById("respuesta").innerHTML="<br>Su IMC es :
    "+r+"<br>"+estado;
    }}

function limpiar()
    {
    document.getElementById("peso").value="";
    document.getElementById("alt").value="";
    document.getElementById("respuesta").innerHTML="<br>Su IMC es :”;

    }
```

Con la aplicación lista se procede a la configuración del archivo *manifest.wepapp*

manifest.webapp

```
{
  "version": "0.1",
  "name": "Calculadora IMC ",
  "launch_path": "index.html",
  "description": "Aplicación para el Calculo del Indice de Masa Corporal",
  "icons": {
        "60": "/img/logo60.png",
        "128": "/img/logo128.png"
  },
  "developer": {
    "name": "Nombre_Desarrollador"
  },
  "installs_allowed_from": ["*"],
  "default_locale": "es"
}
```

El código de la aplicación está disponible para su descarga en (13).

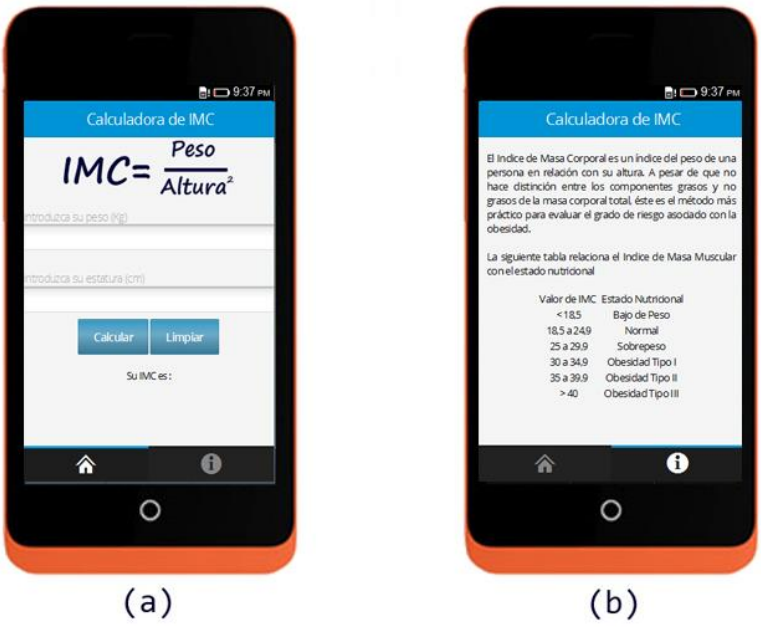

Figura 18. Practica 2: Calculadora de IMC

6 WebApis y WebActivities

En el presente capitulo se describen una serie de elementos utilizados en Firefox OS, que permiten al desarrollador, interactuar con los recursos hardware del dispositivo, crear tareas en segundo plano y acceder a ciertas funciones propias del sistema operativo. Este tipo de elementos son necesarios a la hora de desarrollar aplicaciones que requieran de un cierto grado de procesamiento y funcionalidad.

6.1 Web APIs

Las Web APIs son un conjunto de APIs *(Application Programming Interface)* desarrolladas con el fin de permitir a las Web Apps el acceso a los recursos hardware de un dispositivo. Actualmente se encuentran disponibles aproximadamente 33 APIs compatibles con la plataforma Firefox OS, algunas de las cuales hacen parte del estándar HTML5 y otras añadidas por Mozilla, comunidad que constantemente trabaja en la creación de nuevas APIs y su estandarización por parte del *System Applications Working Group*[13] del W3C.

Las Web APIs están muy relacionadas con los tipos de aplicaciones soportadas por Firefox OS, ya que como se mencionó en la sección 3.5 estas se catalogan de acuerdo a los permisos que requieran para acceder a las APIs, de esta manera se puede decir que existen 3 tipos de Web APIs:

- ***Regular APIs:*** Son aquellas que se encuentran disponibles para cualquier tipo de aplicación ya sea alojada o empaquetada.
- ***Privileged APIs:*** Son aquellas que se encuentran disponibles para las aplicaciones privilegiadas, las cuales deben definirse como tal en el manifest.webapp, hacen uso de una CSP (Content Security Policy) y deben ser aprobadas y firmadas por el MarketPlace.
- ***Certified APIs:*** este tipo de APIs solo están disponibles para el sistema operativo, ya que permiten el acceso a partes muy sensibles del hardware del dispositivo.

Para el caso de las *Privileged APIs* y las *Certified APIs* es necesario definir los permisos en el archivo manifest.webapp de acuerdo al API que se desee utilizar, por ejemplo si se desea hacer uso del API Contacts (*Privileged API*), se deberian añadir las siguientes líneas.

[13] http://www.w3.org/2012/sysapps/

```
"permissions": {
    "contacts": {
        "description": "Required for autocompletion in the share screen",
        "access": "readcreate"
    }
 }
```

6.1.1 APIs para la comunicación

- **WebBluetooth:** Esta proporciona acceso a bajo nivel al hardware de Bluetooth del dispositivo. Esta API está disponible únicamente para aplicaciones certificadas.
- **Mobile connection API:** esta API tiene dos propósitos, suministrar acceso a la información específica y detallada acerca del estado actual de la conexión móvil del dispositivo, y proveer acceso a las capacidades específicas que se encuentran integradas en el ICC (tarjeta SIM). Esta API está disponible únicamente para aplicaciones certificadas.
- **Network Stats API:** El API de estadísticas de red permite realizar un control del uso de los datos y utilizar estos datos en las aplicaciones privilegiadas. Esta API está disponible únicamente para aplicaciones certificadas.
- **TCP Socket API:** Esta API permite el establecimiento de conexiones a través de sockets, permitiendo a los desarrolladores la implementación de cualquier tipo de protocolo disponible en la parte superior, incluso protocolos creados por ellos mismos para satisfacer las diferentes necesidades que se puedan llegar a tener.
- **Telephony:** Permite a las aplicaciones la gestión de las llamadas telefónicas y utilizar la interfaz de usuario de telefonía integrada. Esta API está disponible únicamente para aplicaciones certificadas y no está estandarizada.
- **WebSMS:** permite a las aplicaciones la creación, él envió y la recepción de SMS y MMS, así como la gestión de los mensajes almacenados en el dispositivo. Esta API está disponible únicamente para aplicaciones certificadas y no está estandarizada.
- **Network Information API: p**rovee información básica sobre la conexión de red actual, tal como el ancho de banda utilizado por el usuario mediante el dispositivo o la dosificación de la conexión.
- **Wifi Information API:** provee información sobre la conexión wifi actual, tal como la fuerza de la señal, el nombre de la red, y otras redes wifi disponibles. Esta API está disponible únicamente para aplicaciones certificadas.

6.1.2 APIs para el manejo de datos

- **FileHandle API:** permite la manipulación de archivos, incluyendo la creación de archivos y la modificación de su contenido.
- **IndexDB:** es un API para el **a**lmacenamiento grandes cantidades de información estructurada del lado cliente que incluye soporte para búsquedas de alto rendimiento a través del uso de índices.
- **Contacts API:** provee acceso y permite la gestión de los contactos del usuario almacenados en el dispositivo. Esta API está disponible para aplicaciones privilegiadas y certificadas.
- **Device Storage API:** permite a las aplicaciones la gestión y manipulación de archivos almacenados en una ubicación central en el dispositivo como por ejemplo la carpeta de imágenes, música, videos y a la tarjeta de memoria externa. Esta API está disponible para aplicaciones privilegiadas y certificadas.
- **Settings API: provee** acceso a las opciones de configuración del sistema que se almacenan de forma permanente en el dispositivo, de modo que puedan ser modificadas o monitoreadas por la aplicación. Esta API está disponible para aplicaciones privilegiadas y certificadas, actualmente no está estandarizada.

6.1.3 APIs para el acceso al hardware

- **Ambient Light Sensor API:** Esta API provee acceso al sensor de luz, con el cual una aplicación puede detectar variaciones en el nivel de luz ambiental que rodea al dispositivo, de modo que, por ejemplo, se pueda relacionar el cambio de intensidad de luz ambiental con el nivel de brillo o un cambio en el contraste de color en la interfaz de usuario (UI).
- **Battery Status API:** proporciona información sobre el nivel de carga de la batería y si el dispositivo se encuentra conectado o no a un punto de carga. Permitiendo generar notificaciones de acuerdo a un determinado cambio en el nivel de la batería.
- **Geolocation API:** permite al usuario proporcionar su ubicación a las aplicaciones siempre y cuando este haya proporcionado los permisos respectivos.
- **Proximity API:** permite determinar la proximidad del dispositivo a objetos cercanos, posibilitando la reacción a un cambio de este tipo.
- **Device Orientation API:** provee notificaciones cuando la orientación del dispositivo cambia.

- **Vibration API:** permite a las aplicaciones controlar la vibración de hardware del dispositivo.
- **WebFM API: p**rovee soporte para la funcionalidad de radio FM del dispositivo, permite encender o apagar la radio así como el cambio de estaciones radiales. Actualmente no está estandarizada.
- **Camera API:** permite a las aplicaciones tomar fotografías, capturar información de la lente (zoom, balance de blancos) y grabar videos haciendo uso de la cámara integrada del dispositivo. Esta API está disponible exclusivamente para aplicaciones certificadas y actualmente no está estandarizada.
- **Power Management API:** permite a las aplicaciones encender o apagar aquellos elementos hardware que consuman energía como por ejemplo la pantalla y el CPU. Adicionalmente provee soporte para escuchar e inspeccionar eventos de bloqueo de recursos. Esta API está disponible exclusivamente para aplicaciones certificadas y actualmente no está estandarizada.

6.1.4 Otras APIs

- **Alarm API:** permite la configuración de alarmas en el dispositivo, permitiendo la programación de aplicaciones como el despertador, calendario y actualizaciones para que estas se activen en un momento específico. Las aplicaciones que se deseen que reaccionen a las alarmas deben ser registradas en los mensajes de alarma, ya que la alarma es enviada a las aplicaciones a través del API de mensaje del sistema (System Message API).
- **Simple Push API:** permite a la aplicación tener la capacidad de activarse para recibir notificaciones. Las aplicaciones pueden requerir o solicitar una URI que puede compartir con sus servidores, para que ellos puedan usar los números de versión que serán entregados a las aplicaciones. Esta API podría llegar a ser utilizada como una herramienta de sincronización, o más aun para obtener los últimos datos de los servidores de terceros.
- **Web Notificactions:** permite el envió de notificaciones que serán mostradas fuera de la aplicación al nivel del sistema.
- **Apps API:** brinda soporte para la instalación y administración de las aplicaciones web en el dispositivo. Esta API no está estandarizada.
- **WebPayment API:** Permite al contenido web iniciar el pago y reembolsos por bienes virtuales, así como la confirmación por parte del proveedor. Esta API no está estandarizada.

- **Browser API:** es una extensión del elemento HTML <iframe> que provee soporte para construir un navegador web usando tecnologías web. Esta API está disponible para aplicaciones privilegiadas y certificadas. Actualmente no está estandarizada
- **Idle API:** permite notificar a las aplicaciones cuando el usuario se encuentra inactivo y deja que la aplicación tome las medidas correspondientes cuando el usuario tenga inactivo el dispositivo. Esta API está disponible exclusivamente para aplicaciones certificadas.
- **Permissions API:** permite mostrar los permisos requeridos por las aplicaciones a los usuarios, así como su gestión. . Esta API está disponible exclusivamente para aplicaciones certificadas y actualmente no está estandarizada
- **Time/Clock API:** Provee soporte para ajustar la hora del dispositivo. Esta API está disponible exclusivamente para aplicaciones certificadas y actualmente no está estandarizada.

Para mayor información acerca de estas APIs y otras que sean incorporadas en un futuro a la plataforma Firefox OS se recomienda revisar la documentación oficial disponible en (14) y (15).

6.2 Práctica 3: Battery Status

6.2.1 Enunciado Práctica 3: Battery Status

Esta práctica tiene como objetivo la realización de una aplicación que permita monitorear el estado de la batería a través del uso de la Web API Battery Status. La aplicación consta de una solo pantalla en la cual se presenta al usuario información de la batería del dispositivo como su nivel de cargar, si actualmente se encuentra conectado a una fuente de poder y el tiempo restante para su carga completa o el tiempo para su descarga.

Para el desarrollo de esta aplicación se hace uso de la plantilla app-stub de Mortar y se incorporan además algunos de los elementos de Building Firefox OS.

6.2.2 Solución Practica 3

En primer lugar es necesario descargar la plantilla de Mortar app-stub, esto se realiza haciendo uso de volo y a través del comando `create` en la consola:

```
volo create bateria mozilla/mortar-app-stub
```

Una vez descargada la plantilla el directorio raíz tendrá la estructura de archivos y carpetas típica de las plantillas de Mortar, ahora el siguiente paso a seguir será añadir

al proyecto los recursos a utilizar del proyecto BFFOS, en este caso se hará uso de los elementos headers.css y buttons.css, por consiguiente se añaden estos archivos dentro de la carpeta css del proyecto, adicionalmente se deben agregar sus dependencias (recursos relacionados) para la construcción de la interfaz de usuario, estos archivos se ubican dentro de las carpetas `img/ui/headers` e `img/ui/buttons.` Es importante verificar que los *paths* a los recursos estén bien definidos en los archivos headers.css y buttons.css ya que si no son correctos la interfaz no se mostrara de manera adecuada.

Una vez organizada la estructura de la aplicación se procede a la modificación de los archivos necesarios para su creación.

app.css

En este archivo se definen todas aquellas hojas de estilo necesarias para el funcionamiento adecuado de la aplicación, adicionalmente se incluyen algunas funciones para darle estilo a los elementos de la interfaz principal. El archivo contiene el siguiente código.

```
@import "headers.css";
@import "buttons.css";

html, body {
      margin: 0;
      padding: 0;
      background-image: url(../img/bg01.png);
      font-family: 'Open Sans', sans-serif;
          font-size: 11pt;
          font-weight: 400;
          color: #969696;
    }

   table{
      text-align: justify;
      padding: 10px;
    }
```

index.html

En la sección <head> se encuentran definidos los metadatos y se enlazan las hojas de estilo de la aplicación. A continuación en el `<body>` se define la interfaz, compuesta por un encabezado que contiene el nombre de la aplicación, a continuación una imagen que representa la batería y varía según su estado, seguido de esta se define una tabla de 1 columna y 3 filas donde se presentara al usuario el nivel de carga, el estado actual de la batería (cargando/descargándose) y el tiempo para completar su carga/descarga. Finalmente, un elemento botón el cual invocara la API. El código contenido en este archivo es el siguiente:

```
<!DOCTYPE html>
  <html>

    <head>
      <meta charset="utf-8">
      <title>Bateria</title>
      <meta name="description" content="">
      <meta name="viewport" content="width=device-width">
      <link rel="stylesheet" href="css/app.css">
     </head>

     <body>
       <section class="skin-dark" role="region">
        <header>
          <menu type="toolbar">
          <button aria-controls="info"><span class="icon icon-about">info
          </span></button>
          </menu>
          <h1>Bateria</h1>
        </header>
       </section>
       <br><br><br>
       <div align="center"><img id="image" src="img/9.png" width="200"
        height="100"><br>
       <table border="0">
            <tr><td> <span id="battery-pct"></span></td></tr>
            <tr><td><span id="battery-state"></span></td></tr>
            <tr><td><span id="battery-time"></span></td></tr>
       </table>
       </div>
       <div align="center"><button id="get-battery">Verificar</button></div>
       <script type="text/javascript"  data-main="js/init.js"
        src="js/lib/require.js"></script>
    </body>
  </html>
```

app.js

En este archivo se definen los archivos JavaScript que hacen parte de la aplicación a través de la función require( ), adicionalmente se definen las funciones encargadas de la comunicación con la API y de enviar la respuesta a la interfaz para presentarla al usuario.

```
define(function(require) {
  require('jquery-1.10.2.min');

  $(function() {
    $("#get-battery").click(function() {
    $("#battery-pct").text("Nivel de Carga: " +
       Math.round(navigator.battery.level * 100) + "%");
    $("#battery-state").text((navigator.battery.charging)? "Actualmente
       Cargando" : "Actualmente Descargandose");
    if(navigator.battery.charging){
        $("#battery-time").text("Tiempo para carga completa: " +
           parseInt(navigator.battery.chargingTime / 60, 10) + "mins.");
```

```
  }else{
      $("#battery-time").text("Tiempo de carga restante: " +
         parseInt(navigator.battery.dischargingTime / 60, 10) + "mins.");
  }
  changeImage();
  });
});

 function changeImage()
 {
  x = Math.round(navigator.battery.level * 100);
  element=document.getElementById('image');
  if (x==0)
  {
    element.src="/img/8.png";
  }else if(0<x && x<10){
    element.src="/img/7.png";
  }else if(10<x && x<20){
    element.src="/img/6.png";
  }else if(20<=x && x<26){
    element.src="/img/5.png";
  }else if(26<=x && x<50){
    element.src="/img/4.png";
  }else if(50<=x && x<80){
    element.src="/img/3.png";
  }else if(80<=x && x<95){
    element.src="/img/2.png";
  }else if(90<=x){
    element.src="/img/1.png";
  }
}

});
```

manifest.webapp

El archivo Manifest se modifica del que trae la plantilla para adaptarlo a la aplicación.

```
{

  "version": "0.1",
  "name": "Basicas",
  "description": "Ejemplo del uso de Battery Status API",
  "launch_path": "index.html",
  "icons": {
    "60": "/img/icons/logo60.png",
    "128": "/img/icons/logo128.png"
  },
  "developer": {
    "name": "Alejandro Sotelo"
  },

  "default_locale": "es"
}
```

Una vez finalizada la aplicación se compila y optimiza con el comando `volo build`. La aplicación resultante es la que se observa en la Fig. 19. El código de esta aplicación está disponible para su descarga en (16).

Figura 19. Practica 3: Battery Status

6.3 Web Activities

Las Web activities sirven para realizar invocaciones a otras aplicaciones propias del sistema operativo con el fin de delegar una actividad en particular. Este tipo de invocaciones generalmente es realizado por el usuario y en caso de disponer de múltiples actividades para la realización de una tarea será este el encargado de seleccionar una para llevar a cabo la acción solicitada, así por ejemplo en una aplicación en la que se requiere una imagen, es posible hacer el llamado a la galería de imágenes o a la aplicación de la cámara para obtenerla, evitando la necesidad de implementar esta funcionalidad en la aplicación origen.

También hacen posible que una aplicación tenga la capacidad de manejar este tipo de actividades, es decir permitir que soporte el llamado por parte de otras actividades para la realización de alguna tarea. Si se desea que una aplicación cuente con la capacidad de manejar alguna actividad, se debe especificar dicha capacidad en particular al momento de desarrollar la aplicación en el archivo manifest.webapp, de la siguiente manera:

```
"activities": {

  "activity_name": {
      "href": "valor",
      "disposition": "valor",
      "filters": "valor"
      },
      "returnValue": true
  }}
```

`activities:` la sección activities define el conjunto de actividades a las que la aplicación podrá acceder.

`activity_name`: este valor corresponde al nombre de la actividad que se quiere definir, los posibles valores para este campo se presentan en la Tabla 4 (columna 1).

`href:` esta propiedad especifica la ubicación de la página que debe ser accedida cuando la aplicación inicia la actividad, esta debe ser abierta de acuerdo a lo especificado en la propiedad `disposition.`

disposition: esta propiedad específica cómo debe ser presentada la página al ser invocada la actividad. Los posibles valores para esta propiedad son: `window` (despliega la App que realiza la actividad) e `inline` (se presenta como un popup sobre la aplicación original que requiere la actividad). Esta propiedad es opcional y en caso de no ser definida el valor por defecto es `window`.

return value: esta propiedad es opcional y especifica si la actividad debe retornar o no un valor.

filters: esta propiedad es opcional y define una serie de filtros que sirven para determinan que aplicaciones pueden realizar una determina actividad solicitada por otra aplicación, los valores de filtro pueden ser personalizados, definidos a través un valor, un vector o un objeto de definición, mas sin embargo los nombres de los filtros deben corresponder con los definidos en MozActivityOptions, los filtros pueden tener siguientes propiedades:

- **required:** booleano que indica si la correspondiente propiedad MozActivityOptions.data debe existir o no.
- **value:** define un valor o vector de valores. El valor de correspondiente propiedad MozActivityOptions.data debe ser igual o al menos uno de los valores definidos en el filtro.
- **min:** si el valor esperado es un número, define su valor mínimo. El valor de la correspondiente propiedad MozActivityOptions.data debe ser mayor o al menos igual al valor definido.
- **max:** si el valor esperado es un número, define su valor máximo. El valor de la correspondiente propiedad MozActivityOptions.data debe ser menor o al menos igual a el valor definido
- **pattern:** un patrón string basado en la sintaxis para definir expresiones regulares de JavaScript. El valor correspondiente de la propiedad

MozActivityOptions.data debe coincidir con el patrón. Requiere al menos de Firefox OS 1.2.

- **regexp:** un string que contiene un literal del regexp basado en la sintaxis para definir expresiones regulares de JavaScript. A diferencia de `pattern` puede coincidir solo con una parte del valor correspondiente de la propiedad MozActivityOptions.data, haciendo uso de meta caracteres. Solo funciona en Firefox OS 1.0 y 1.1.

La Tabla 4, muestra las actividades provistas por Firefox OS, junto con su aplicación correspondiente y los filtros que pueden ser utilizados para su declaración.

Tabla 4. Actividades de Firefox OS (16)

Nombre	Aplicación	Datos esperados (Filtros)
browse	Gallery	type: "photos"
configure	Settings	target: "device"
costcontrol/balance costcontrol/data_usage costcontrol/telephony	Costcontrol	Ninguna

dial	Communication	type: "webtelephony/number",
		number: {
		regexp:/^[\\d\\s+#*().-]{0,50}$/
		}

new	Communication	type: "webcontacts/contact"
	Email	type: "mail"

	SMS	type: "websms/sms",
		number: {
		regexp:/^[\\w\\s+#*().-]{0,50}$/
		}

open	Communication	type: "webcontacts/contact"
	Gallery	type: ["image/jpeg", "image/png", "image/gif", "image/bmp"]
	Music	type: ["audio/mpeg", "audio/ogg", "audio/mp4"]
	Video	type: ["video/webm", "video/mp4", "video/3gpp", "video/youtube"]
pick	Camera, Gallery, Wallpaper	type: ["image/*", "image/jpeg"]
	Communication	type: ["webcontacts/contact", "webcontacts/email"]
record	Camera	type: ["photos", "videos"]

save-bookmark	Homescreen	type: "url",
		url: {
		required:true,
		regexp:/^https?:/
		}

share	Bluetooth	number: 1
	Email, Wallpaper	type: "image/*"

view	Browser	type: "url"
		url: {
		required: true,
		regexp: /^https?:.{1,16384}$/
		}

	Email	type: "url",
		url: {
		required:true,
		regexp:/^mailto:/
		}

	PDFs	type: "application/pdf"
	Video	type: ["video/webm", "video/mp4", "video/3gpp", "video/youtube"]
update	Communication	type: "webcontacts/contact"

6.4 Práctica 4: Funciones Básicas (Web Activities)

6.4.1 Enunciado Práctica 4: Funciones Básicas

Esta práctica tiene como objetivo la realización de una aplicación que permita delegar una serie de acciones a otras aplicaciones del sistema operativo a través de las Web Activities. La aplicación consta de una solo pantalla a través de la cual se presentan una serie de opciones que representan las acciones a realizar en caso de ser seleccionadas por el usuario.

Para el desarrollo de esta aplicación se hace uso de la plantilla app-stub de Mortar y se incorporan además algunos de los elementos de Building Firefox OS.

6.4.2 Solución Practica 4

Para la creación de la plantilla básica de esta aplicación se seguirá el procedimiento utilizado en la *Practica 3: Battery Status.* Para comenzar se debe descargar la plantilla de Mortar app-stub y añadir al proyecto los recursos headers.css y buttons.css (junto con sus dependencias) del proyecto BFFOS.

Una vez organizada la estructura de la aplicación se procede a la modificación de los archivos necesarios para su la creación.

main.css

Este archivo se crea con el fin de dar un mejor aspecto visual a algunos de los elementos de la interfaz principal, el archivo contiene el siguiente código:

```
html, body {
      margin: 0;
      padding: 0;
      background-color: #fff;
    }
    body {
      background: none;
    }
    section[role="region"] {
      margin-bottom: 1.5rem;
      position: relative;
    }
    p{
      text-align: justify;
      padding: 10px;
    }
    #main {
      position: relative;
      padding-left: 2rem;
      padding-right: 2rem;
    }
```

app.css

En este archivo se definen todas aquellas hojas de estilo necesarias para el funcionamiento adecuado de la aplicación. En este caso se deben añadir las siguientes líneas

```
@import "buttons.css";
@import "headers.css";
@import "main.css";
```

index.html

En la sección <head> se encuentran definidos los metadatos y se enlazan las hojas de estilo de la aplicación. A continuación en el <body> se define la interfaz, en este caso se compone por un encabezado que contiene el nombre de la aplicación y una serie de botones a través de los cuales se invocan las funciones que ejecutan cada una de las Web Activities definidas.

```
<html>
  <head>
    <meta charset="utf-8">
    <title>funBasic</title>
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width">
    <link rel="stylesheet" href="css/app.css">
  </head>
```

```
    <body>
    <section role="region">
      <header>
        <menu type="toolbar"></menu>
        <h1>funBasicas</h1>
      </header>
    </section>

  <p>Esta app permite la ejecucion de funciones propias del sistema
  operativo, a traves del uso de las Web Activities</p>

  <div id="main">
       <button id="dial">Llamar</button>
       <button id="add-contact">A&ntilde;adir contacto</button>
       <button id="send-sms">Enviar un SMS</button>
       <button id="compose-email">Enviar E-mail</button>
       <button id="take-picture">Tomar una Foto</button>
       <button id="browse-picture">Galeria de Fotos</button>
       <button id="share">Compartir</button>
  </div>
  <script type="text/javascript" data-main="js/init.js"
  src="js/lib/require.js"></script>
 </body>
</html>
```

app.js

En este archivo se definen los archivos JavaScript que hacen parte de la aplicación a través de la función require( ), para esta aplicación se definió la funcionalidad en el archivo webapp.js, por consiguiente debe añadirse de la siguiente manera:

```
define(function(require) {
    require('./webapp');
});
```

webapp.js

Aquí se definen 7 funciones cada una de las cuales implementa la WebActivity designada por el atributo id del correspondiente Botón:

```
(function () {

    var dial = document.querySelector("#dial");
    if (dial) {
        dial.onclick = function () {
            var call = new MozActivity({
                name: "dial",
                data: {
                    number: "+34"
                }
            });
        }}
```

```
var addContact = document.querySelector("#add-contact");
    if (addContact) {
        addContact.onclick = function () {
            var newContact = new MozActivity({
                name: "new",
                data: {
                    type: "webcontacts/contact"
                }
            });
        }}

var sendSMS = document.querySelector("#send-sms");
    if (sendSMS) {
        sendSMS.onclick = function () {
            var sms = new MozActivity({
                name: "new",
                data: {
                    type: "websms/sms"

                }
            });
        }}

 var composeEmail = document.querySelector("#compose-email");
    if (composeEmail) {
        composeEmail.onclick = function () {
            var createEmail = new MozActivity({
                name: "new",
                data: {
                    url: "mailto:ejemplo@email.com"
                }
            });
        }}

 var pickImage = document.querySelector("#take-picture");
    if (pickImage) {
        pickImage.onclick = function () {
            var pick = new MozActivity({
                name: "pick",
                data: {
                    type: ["image/png", "image/jpg", "image/jpeg"]
              }
            });
        }}

  var share = document.querySelector("#browse-picture");
    if (share) {
        share.onclick = function () {
            var sharing = new MozActivity({
                name: "browse",
                data: {
                     type: "photos"

                }
            });
         }}

    var share = document.querySelector("#share");
    if (share) {
```

```
        share.onclick = function () {
            var sharing = new MozActivity({
                name: "share",
                data: {
                     number: 1

                }
            });
        }}
})();
```

manifest.webapp

El archivo Manifest se modifica del que trae la plantilla para adaptarlo a la aplicación.

```
{

  "version": "0.1",
  "name": "funBasicas",
  "description": "Ejemplo del uso de las Web Activities en FXOS",
  "launch_path": "index.html",
  "icons": {
    "60": "/img/icons/logo60.png",
    "128": "/img/icons/logo128.png"
  },
  "developer": {
    "name": "Alejandro Sotelo"
  },

  "default_locale": "es"
}
```

Una vez finalizada la aplicación se compila y opimiza la aplicacion con el comando `volo build.` La aplicación resultante es la que se observa en la Fig. 20 (a), el resto de vistas corresponden a cada una de las aplicaciones a las cuales se le ha delegado una actividad. El código de esta aplicación está disponible para su descarga en (17).

Figura 20. Practica 4: funBasicas

7 Distribución y Publicación de las aplicaciones

Hasta este punto se han presentado una serie de conceptos, herramientas y recursos que han permitido el diseño y la creación de aplicaciones para Firefox OS. En este último capítulo se explicara el proceso de distribución de las aplicaciones, con el fin de llevarlas a los usuarios finales.

7.1 Distribución a través de un servidor Web

Una vez finalizada la aplicación el primer paso es alojarla o desplegarla en un servidor web a través del cual los usuarios puedan instalarla. Dependiendo de las capacidades y de las necesidades de la aplicación se puede alojar en servidores como GitHub (recomendado para aplicaciones que no requieren procesamiento del lado del servidor) o en soluciones genéricas de alojamiento web.

Una vez la aplicación se encuentra desplegada en el servidor, se procede a su publicación con el fin de que los usuarios puedan instalarla. Para el caso de aplicaciones alojadas existen dos posibilidades: publicarla a través del Marketplace o implementar un código en JavaScript en el servidor de alojamiento que permita gestionar la instalación y actualización de la aplicación en los navegadores de los usuarios, para más información acerca de este procedimiento remítase a (17).

7.2 Distribución a través del Marketplace

La distribución de aplicaciones a través del Marketplace está destinada exclusivamente a aplicaciones empaquetadas (no confundir distribución con publicación ya que en el Marketplace se permite la publicación de ambos tipos de aplicaciones), las cuales por su naturaleza requieren acceso a las características de los dispositivos a través de las Web APIs.

En este sentido el Marketplace establece unas políticas de seguridad y unos criterios de revisión de las aplicaciones con el fin de garantizar su integridad, funcionalidad y veracidad de modo que se ofrezca a los usuarios aplicaciones seguras. Estos criterios dependen del tipo de aplicación a distribuir, aplicación certificada, con privilegios o empaquetada simple.

Una vez aprobado el proceso de revisión, se firma criptográficamente el archivo ZIP con una clave privada, brindando mayor seguridad a los usuarios de que la aplicación ha sido revisada cuidadosamente en busca de problemas potenciales de seguridad, privacidad y operatividad.

Si se desea revisar el conjunto de requisitos que una aplicación debe cumplir para ser distribuida a través del Marketplace se recomienda revisar los criterios de revision del marketplace, disponibles en (18).

7.3 Publicación de aplicaciones en el Marketplace

El Marketplace de Firefox OS proporciona infraestructura para que los desarrolladores ofrezcan las aplicaciones a los usuarios finales, a continuación se describen los pasos para la publicación de una aplicación en la tienda:

Iniciar Sesión / Crear una cuenta

Para acceder al centro de desarrolladores del Marketplace de Firefox se debe acceder a la dirección https://marketplace.firefox.com/developers/. A continuación seleccionar la opción ***Entrar*** ubicada en la parte superior derecha, se solicitara el inicio de sesión, el cual es controlado a través de Mozilla Persona, si ya se tiene una cuenta basta con ingresar la dirección de correo e iniciar sesión, de no ser así se debe proceder a la creación de una cuenta en https://login.persona.org/, para lo que es necesario simplemente disponer de una dirección de correo valida.

Enviar la aplicación

Una vez creada la cuenta y autentificados se accede a la opción ***Enviar aplicación***, la cual se compone de cuatro secciones. En primer lugar se presenta el acuerdo para desarrolladores (Fig. 21) en el que se estipulan los términos de uso de la plataforma. Una vez revisado se procede a dar clic en **Aceptar y continuar**.

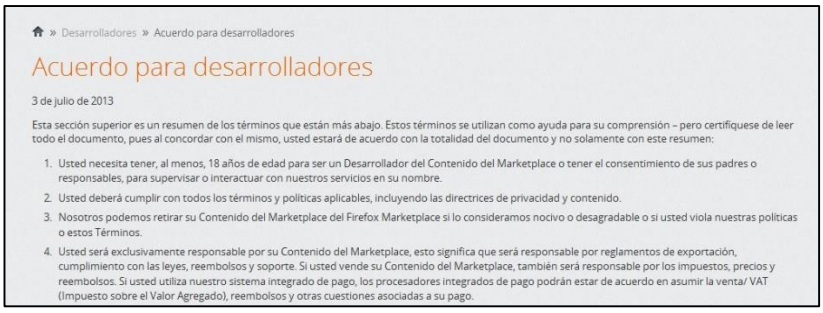

Figura 21. Acuerdo para desarrolladores

En la segunda sección (Fig. 22) se deben definir las siguientes características de la aplicación:

- **Tipo de aplicación:** Gratuita o de Pago
- **Dispositivos con los que la aplicación es compatible:** Firefox OS, Firefox (Windows Mac y Linux), Firefox Mobile (Smartphones Android) y Tableta Firefox (Tabletas Android), siendo posible seleccionar más de una opción.

- **Tipo de Aplicación:** Alojada (Ingresar la URL) o Empaquetada (Cargar el archivo ZIP[14]) una vez seleccionada la opción se procede al proceso de validación del archivo Manifest, si todo es correcto hacer clic en ***Continuar***, de lo contrario se debe revisar el informe de validación y corregir el problema.

Figura 22. Información de la aplicación

El siguiente paso es editar los detalles de la aplicación (Fig. 23), donde automáticamente se rellenan los campos que sean posibles de acuerdo a la información suministrada en el archivo Manifest. En este punto es importante rellenar los campos obligatorios resumen, categorías, política de privacidad, tipos de dispositivos y añadir al menos una captura de pantalla.

Figura 23. Detalles de la aplicación

Una vez finalizado este proceso, en la cuarta sección (Fig. 24) se notifica al usuario de que el proceso ha culminado con éxito, pero esto no indica que la aplicación haya sido publicada ya que aún debe ser sometida al proceso de revisión.

[14] Se deben comprimir los componentes (archivos y carpetas) de la aplicación en un archivo ZIP, más no la carpeta contenedora (directorio raíz).

Figura 24. Proceso de publicación finalizado

Proceso de Revisión

Antes de que la aplicación sea publicada, debe pasar por un proceso de revisión, con el fin de garantizar que cumple con los requisitos de calidad y seguridad; de acuerdo al tipo de aplicación este proceso puede tardar más o menos tiempo. Para el seguimiento de este proceso el desarrollador puede hacer uso de la opción ***Administrar mis envios*** **(**Fig. 25**),** donde además se puede realizar la gestión de la aplicación.

Una vez superado el proceso de revisión es notificado al correo registrado en la cuenta e inmediatamente (si se seleccionó la opción en el proceso de publicación) la aplicación estará disponible para su descarga en el Marketplace (Fig. 26).

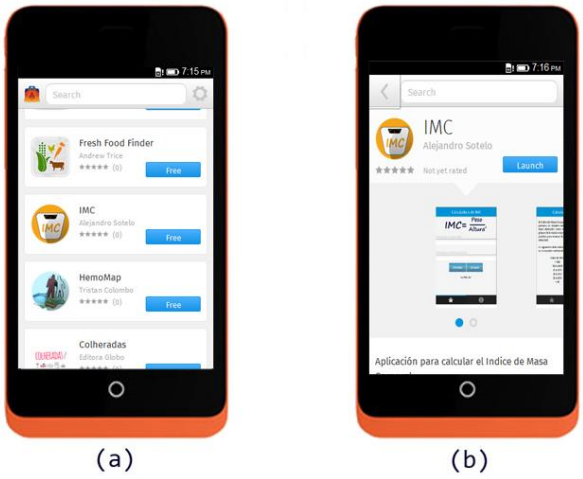

Figura 25. Aplicación disponible en el Marketplace

8 Conclusiones

Como resultado del proceso de investigación y tras la elaboración de este documento, son varias las conclusiones acerca de Firefox OS que se han podido obtener. Para comenzar, se puede decir que la evolución de las tecnologías web junto con la idea de convertir la web en la plataforma universal, han permitido el surgimiento de nuevos escenarios para el despliegue de aplicaciones, favoreciendo en esta medida el surgimiento de Firefox OS y perfilándolo como uno de los sistemas operativos a tener en cuenta en un futuro próximo.

El hecho de ser uno de los únicos sistemas operativos para dispositivos móviles basado en su totalidad en código abierto y soportado en una comunidad tan amplia de desarrolladores como es la de Mozilla, promueve su rápido desarrollo, mantenimiento y constante actualización, todo esto sumado al fuerte apoyo recibido por empresas fabricantes y de telecomunicaciones como es el caso de Telefónica, garantizan en cierta medida su permanencia en el mercado.

Respecto a su modelo de negocio, es evidente un planteamiento minimalista pero a la vez realista a través del cual se busca cubrir la brecha existente dejada por los sistemas operativos dominantes en el mercado, lo que favorece su afianzamiento y posicionamiento a un ritmo lento pero seguro en el mercado de la telefonía móvil.

En relación al desarrollo de aplicaciones, es importante destacar el hecho de que las tecnologías usadas sean las mismas que para el desarrollo web, lo que permite que para un desarrollador con conocimientos previos en estas tecnologías la curva de aprendizaje sea corta; en contraparte, si un usuario desea iniciarse en el desarrollo de aplicaciones para esta plataforma, su excelente documentación y herramientas como Mortar, BBFOS entre otras, brindan un punto de partida bastante completo facilitando el proceso.

A pesar de que aun las aplicaciones para la plataforma no han alcanzado un nivel de madurez adecuado, debido a los limitantes en el manejo de algunas características hardware de los dispositivos, se espera que gracias a la creación y el esfuerzo por la estandarización de las Web APIs en unos pocos años el desarrollo web se situé al mismo nivel alcanzado por el desarrollo nativo.

Aunque es habitual en todos los sistemas operativos para dispositivos móviles la existencia de una tienda para la publicación de aplicaciones, el Marketplace se diferencia de sus competidores en dos aspectos significativos: en primer lugar y sin duda la característica más llamativa para los desarrolladores, es que permite la publicación de aplicaciones sin coste alguno y en segundo lugar la posibilidad de publicar aplicaciones alojadas en servidores externos, esto sin descuidar los aspectos de seguridad ya que todas a las aplicaciones deben pasar por un proceso de validación y certificación al igual que en las demás tiendas con el fin de verificar su veracidad, integridad y funcionalidad.

Finalmente y para dar por terminado este trabajo se espera que la información expuesta en este documento haya sido lo suficientemente clara y concisa y que sirva como orientación a un usuario en el desarrollo de aplicaciones web abiertas para la plataforma Firefox OS.

Bibliografía

1. **Gartner, Inc.** Gartner Says by 2016, More Than 50 Percent of Mobile Apps Deployed Will be Hybrid. [En línea] 4 de Feb de 2013. http://www.gartner.com/newsroom/id/2324917.

2. **Kader, Jalil Abd-el, y otros.** Entrar a la Cuarta Pantalla: Guía para Pensar en Móvil. [En línea] [Citado el: 11 de Julio de 2013.] www.4tapantalla.com.

3. **W3Schools.** *CSS Syntax - Learn CSS.* [En línea] [Citado el: 30 de Mayo de 2013.] http://www.w3schools.com/css/css_syntax.asp.

4. **Mozilla Wiki.** *Booting to the Web.* [En línea] [Citado el: 10 de Junio de 2013.] https://wiki.mozilla.org/Booting_to_the Web.

5. IDC Corporate USA. *Worldwide Quarterly Mobile Phone Tracker 1Q.* [En línea] 16 de Mayo de 2013. [Citado el: 11 de Junio de 2013.] http://www.idc.com/getdoc.jsp?containerId=prUS24108913.

6. Mozilla Developer Network and Individual Contributors. *Firefox OS Architecture.* [En línea] [Citado el: 2013 de Junio de 14.] https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Platform/Architecture?redirectlocale=en-US&redirectslug=Mozilla%2FFirefox_OS%2FArchitecture.

7. Mozilla Developer Network and Individual Contributors. *App Manifest.* [En línea] [Citado el: 16 de Junio de 2013.] https://developer.mozilla.org/en-US/docs/Web/Apps/Manifest.

8. Sotelo, Omar Alejandro. GitHub. ***Repositorio FirefoxOS_Template.*** [En línea] http://github.com/alejo1889/FirefoxOS_Template.

9. Mozilla. Mozilla Style Guide. *Firefox OS app icons.* [En línea] [Citado el: 15 de Junio de 2013] http://www.mozilla.org/en-US/styleguide/products/firefoxos/icons/

10. Jaxo. GitHub. *Repositorio fxosstub.* [En línea] https://github.com/Jaxo/fxosstub.

11. Mozilla. GitHub. *Repositorio Mortar.* [En línea] https://github.com/mozilla/mortar.

12. Alvarez, Miguel Angel. Desarrollo Web. *Guía para la instalación del framework para Javascript del lado del servidor, NodeJS.* [En línea] [Citado el: 17 de Julio de 2013.] http://www.desarrolloweb.com/articulos/instalar-node-js.html.

13. Sotelo, Omar Alejandro. GitHub. *Repositorio Practica 1: Calculadora IMC.* [En línea] https://github.com/alejo1889/FX_IMC.

14. Mozilla Developer Network and Individual Contributors. *Web API.* [En línea] [Citado el: 28 de Julio de 2013.] https://developer.mozilla.org/es/docs/WebAPI.

15. Mozilla Developer Network and Individual Contributors. *Documentation Status.* [En línea] [Citado el: 23 de Julio de 2013.] https://developer.mozilla.org/en-US/docs/WebAPI/Doc_status.

16. Sotelo, Omar Alejandro. GitHub. *Repositorio Practica 3: Practica3(Battery Status).* [En línea] https://github.com/alejo1889/Practica3.

17. Mozilla Developer Network and Individual Contributors. *Web Activities.* [En línea] [Citado el: 31 de Julio de 2013.] https://developer.mozilla.org/en-US/docs/WebAPI/Web_Activities.

18. Sotelo, Omar Alejandro. GitHub. *Repositorio Practica 4:practica_WebAPI (funBasicas).* [En línea] https://github.com/alejo1889/practica_WebAct.

19. Network, Mozilla Developer. App Installation and Management APIs. [En línea] https://developer.mozilla.org/en-US/docs/Web/Apps/JavaScript_API?redirectlocale=en-US&redirectslug=JavaScript_API.

20. Firefox Marketplace. *Criterios de revision del marketplace.* [En línea] [Citado el: 15 de Julio de 2013.] https://marketplace.firefox.com/developers/docs/review.