Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Grado

ANÁLISIS DE LA API DE FIREFOX OS. DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN PARA TERMINALES CON FIREFOX OS

Alumno: Juan Antonio Romero Olmo

Tutor: Prof. D. Víctor Manuel Rivas Santos
Dpto: Informática

Junio, 2015

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

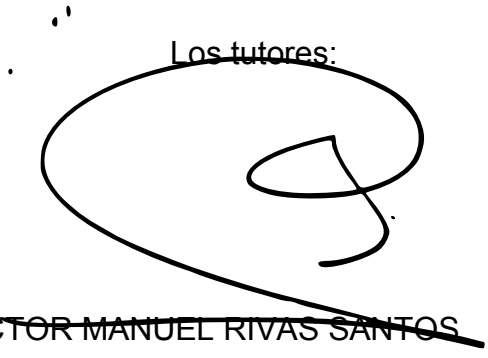
Don VICTOR MANUEL RIVAS SANTOS, tutor del Proyecto Fin de Carrera titulado: ANÁLISIS DE LA API DE FIREFOX OS. DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN PARA TERMINALES CON FIREFOX OS, que presenta, JUAN ANTONIO ROMERO OLMO, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Junio de 2015

El alumno:

JUAN ANTONIO ROMERO OLMO

Los tutores:



VICTOR MANUEL RIVAS SANTOS

AGRADECIMIENTOS

La realización y defensa de este Trabajo de Fin de Grado supone para mi la conclusión de otro proyecto mucho más grande, culminar con éxito una titulación que hace tiempo dejé inacabada. Quiero dedicar esta memoria a todos aquellos que en algún momento han contribuido a hacerme llegar hasta aquí:

A mi familia, especialmente a mi madre, por su paciencia a la hora de aceptar mis decisiones, sobre todo las equivocadas, y por sus incansables recordatorios de *"tienes que acabar aquello que empezaste"*.

A *Víctor*, mi director de proyecto, profesor, compañero y amigo, por toda su ayuda y comprensión, he aprendido mucho contigo.

A *David*, por sus críticas, consejos y sugerencias, que se han traducido en una gran contribución a esta memoria.

A todos mis compañeros y amigos, en especial a *Mari Angeles Sánchez* y *David Fernández*, por todo el apoyo que me habéis prestado y por los buenos "ratos" que hemos pasado juntos durante este tiempo.

Muchísimas gracias.

Tabla de contenidos

Tabla de ilustraciones

1. INTRODUCCIÓN

1.1. Introducción al proyecto

Cuando nacieron los teléfonos móviles, sus principales funcionalidades eran: llamar y enviar mensajes de texto SMS, pero con la llegada de los smartphones[1] el móvil se ha convertido en un dispositivo multifuncional que no sólo comunica, sino que ayuda a aprender, relacionarse y divertirse.

En el comienzo del nuevo milenio se disparó la evolución del mercado de los contenidos y aplicaciones para móviles, propiciada por la apertura al desarrollo de software a terceros, de sistemas operativos como Windows Mobile, Symbian, BlackBerry de RIM, Android o Mac iOS, a diferencia del convencional entorno de programación de los teléfonos móviles estándar.

De esta forma los fabricantes de dispositivos móviles trataron de hacer sus productos más atractivos para los clientes, introduciendo cada vez mayor cantidad de aplicaciones, dando así a los usuarios la posibilidad de personalizar sus dispositivos eligiendo las aplicaciones que desearán. A las pretensiones de los usuarios se unen las de los desarrolladores de software para móviles, que quieren libertad para desarrollar aplicaciones móviles sin restricciones para el usuario final.

En la actualidad nos encontramos en un universo tecnológico en el que podemos encontrar multitud de terminales con diferentes configuraciones hardware y un amplio conjunto de sistemas operativos con distintos niveles de aceptación por parte de los usuarios.

1.2. Propósito

El propósito de este trabajo es mostrar una visión general de la implementación y de las funcionalidades del sistema operativo de la fundación Mozilla, *Firefox OS*. Para ello se hará una revisión de sus características y su arquitectura para conseguir un conocimiento teórico completo del mismo. Para comprobar la complejidad del desarrollo de aplicaciones para este sistema

[1] Teléfono móvil con características de conectividad convencionales y capacidades de cómputo y almacenamiento semejantes a un ordenador personal.

operativo y su funcionamiento, se construirá, a modo de ejemplo, una aplicación compatible con el mismo, que se usará en el terminal ZTE Open.

1.3. Objetivos del proyecto

Se pretende conseguir la consecución de los puntos descritos en el apartado anterior. Para ello, se tratará de dar una descripción detallada del entorno para el que se desarrolla la aplicación, la forma de interactuar con él y una explicación de *¿cómo?* y *¿con qué?* podemos desarrollar una aplicación que sea soportada por este sistema operativo para dispositivos móviles.

2. DESCRIPCIÓN DE LAS TECNOLOGIAS ESTUDIADAS

2.1. Firefox OS

Firefox OS[2] (conocido como Boot to Gecko[3] o B2G) es un sistema operativo móvil, basado en HTML5 con núcleo Linux, de código abierto, para varias plataformas. Es desarrollado por Mozilla[4] Corporation con el apoyo de otras empresas y una gran comunidad de voluntarios de todo el mundo.

Inicialmente estuvo enfocado a dispositivos móviles, smartphones y tabletas, incluidos los de gama baja. Están diseñados para ser una alternativa a otros dispositivos de gama alta con sistemas operativos como Android o IOS, a un precio más asequible. En España, el 2 de julio de 2013, Telefónica comenzó la venta de los primeros terminales con Firefox OS, el ZTE Open y el Peak de Geeksphone. Firefox OS se puede aplicar también a otros dispositivos como Raspberry Pi y próximamente se podrá usar en computadores de bajo consumo y televisores (Smart TVs).

2.1.1. Terminología

Antes de comenzar la explicación en profundidad del sistema operativo, será necesario conocer una serie de términos que ayudarán a comprenderla.

- **B2G**

  Abreviatura de Boot to Gecko.

- **Boot to Gecko**

  Es el nombre del código para el proyecto Firefox OS. Probablemente también se pueda ver este término usado para referirse a Firefox OS, ya que dicho nombre fue usado mucho tiempo antes de que el proyecto tuviese un nombre oficial.

[2] http://www.mozilla.org/es-ES/firefox/os/
[3] Gecko es la capa de Firefox OS que proporciona la misma implementación de estándares web abiertos usada por Firefox y Thunderbird, así como muchas otras aplicaciones.
[4] http://mozilla.org

- **Gaia**

Es la interfaz de usuario de la plataforma Firefox OS. Todas las imágenes que aparecen en la pantalla desde que Firefox OS se activa, son iniciadas en Gaia. Implementa el bloqueo de pantalla, la pantalla de inicio, y todas las aplicaciones estándar.

Gaia está implementada completamente usando HTML, CSS y JavaScript, siendo sus únicas interfaces con el sistema operativo las WebAPIs, las cuales son implementadas por la capa denominada Gecko.

- **Gecko**

Es la capa que provee todo el soporte para el trio de estándares abiertos: HTML, CSS y JavaScript. Esto hace que dichas APIs funcionen bien en cualquier sistema operativo que soporte Gecko. incluye, entre otras cosas, un paquete de red, un paquete gráfico, un motor de diseño, una máquina virtual de JavaScript, y capas de portabilidad.

- **Gonk**

Gonk es el sistema operativo de bajo nivel de la plataforma Firefox OS, consiste en un kernel Linux y una capa de abstracción de hardware de espacio de usuario (HAL por sus siglas en inglés). El kernel y varias de las librerias del espacio del usuario son proyectos de código abierto: GNU/Linux, libusb, bluez, etc. Algunas partes de HAL son compartidas con el proyecto Android, como por ejemplo el GPS y la cámara.

Se puede decir que Gonk es una simple distribución de GNU/Linux. Su propósito es conseguir la portabilidad de Gecko; ésto es, que se puedan ejecutar comandos Gecko en Gonk.

Desde que el proyecto Firefox OS tiene control total sobre Gonk, se pueden mostrar interfaces de Gecko que no pueden mostrar otros sistemas operativos. Un claro ejemplo es el acceso directo a todo el

paquete de telefonia o al almacenamiento del dispositivo, funcionalidades a las que no puede tener acceso ningún otro sistema operativo.

- Jank

Este término, generalmente empleado en el área de las aplicaciones móviles, se refiere al efecto causado por código lento o ineficiente en una aplicación, que podría blquear la actualización de la interfaz de usuario y provocar su lentitud o que no responda.

2.2. Arquitectura del Sistema Operativo Firefox OS

Para los desarrolladores web, la parte más importante, es entender que la interfaz de usuario completa es una aplicación web, **Gaia**, que es capaz de mostrar e iniciar otras aplicaciones web. Cualquier modificación que se haga a la interfaz de usuario y cualquier aplicación que se cree para ejecutarse en Firefox OS, son páginas web, aunque con acceso mejorado al hardware del dispositivo móvil y sus servicios.

2.2.1.El proceso de Auto-arranque

Cuando se arranca un dispositivo que usa Firefox OS, la ejecución empieza con los procesos de arranque primario. Seguidamente, continúa el proceso de carga del sistema operativo, sucediéndose procesos de auto-arranque de nivel superior. Al final, la ejecución se le da al kernel de Linux.

Cabe resaltar algunos puntos sobre estos procesos:

- Los procesos de arranque son los que normalmente muestran la primera pantalla vista por el usuario durante la carga (el logo del fabricante).
- Los procesos de arranque cargan imágenes al dispositivo, pudiendo ser de distintos tipos; la mayoría de teléfonos móviles utilizan Fastboot[5], aunque existen otros, por ejemplo el Samsung Galaxy S II usa el protocolo Odin.

[5] Fastboot es el protocolo utilizado para actualizar el sistema de archivos de la memoria flash en los dispositivos Android, desde un host a través de USB

- Al final del proceso de auto-arranque, se carga y se ejecuta la imagen del módem. Esta depende de cada dispositivo y puede ser propietaria en algunos de ellos.

2.2.2.El Kernel de Linux

El kernel usado por Gonk es muy similar al kernel de Linux original del que deriva. Existen algunas pequeñas variaciones hechas por AOSP[6] las cuales aún no se han subido. Los fabricantes también suelen modificar el kernel y actualizan dichos cambios según su programación.

El proceso de arranque de Linux[7] está bien documentado en Internet. El núcleo Linux activará dispositivos y ejecutará procesos esenciales para arrancar procesos esenciales como *b2g* [procesos básicos de Firefox OS, contenedores de Gecko] y *rild* [un proceso relacionado con la telefonía que puede ser específico de cada chip]. Al final de este, el proceso se lanza el espacio de usuario "init", como en la mayoría de sistemas UNIX, en este punto es montado un disco de memoria (RAM disk) que contiene utilidades esenciales, además de otros scripts y kernels ejecutables.

Una vez que el proceso "init" es lanzado, el kernel de Linux maneja las llamadas al sistema del espacio de usuario, interrupciones y peticiones de otros dispositivos hardware.

Muchas funcionalidades del hardware son mostradas en el espacio de usuario a través del sistema de ficheros virtual sysfs[8]. Por ejemplo, para leer es estado de la batería en Gecko tenemos el siguiente código:

```
1  FILE *capacityFile = fopen("/sys/class/power_supply/battery/capacity", "r");
2  double capacity = dom::battery::kDefaultLevel * 100;
3
4  if (capacityFile) {
5      fscanf(capacityFile, "%lf", &capacity);
6      fclose(capacityFile);
7  }
```

Ilustración 1 – Acceso al estado de la batería del dispositivo

[6] Android Open Source Project
[7] Linux Startup Process
[8] Sysfs

2.2.3.El proceso Init

El proceso *init* en Gonk, maneja el montaje de los sistemas de archivos requeridos e inicia nuevos servicios, una vez que termina ésta tarea, se queda en segundo plano, de forma similar a los *init* de otros sistemas operativos basados en UNIX.

Este proceso interpreta scripts, (archivos *init*.rc*) consistentes en comandos que describen como se deberían iniciar diversos servicios.

El *init.rc* del sistema operativo Firefox OS es el mismo que usa Android para ese equipo en particular, éste incluye los requisitos para ejecutarlo y varía de un dispositivo a otro.

Una tarea fundamental del proceso *init* es el manejo del proceso *b2g,* este es el núcleo del sistema operativo Firefox OS, siendo el código en el *init.rc* similar al del ejemplo:

```
1  service b2g /system/bin/b2g.sh
2      class main
3      onrestart restart media
```

Ilustración 2 – Proceso b2g

2.2.4.La arquitectura del proceso de usuario Userspace

Para tener una idea de como se integran e interactúan los diversos componentes de Firefox OS, vamos a ver un diagrama del userspace del sistema operativo con el que obtendremos una visión de alto nivel.

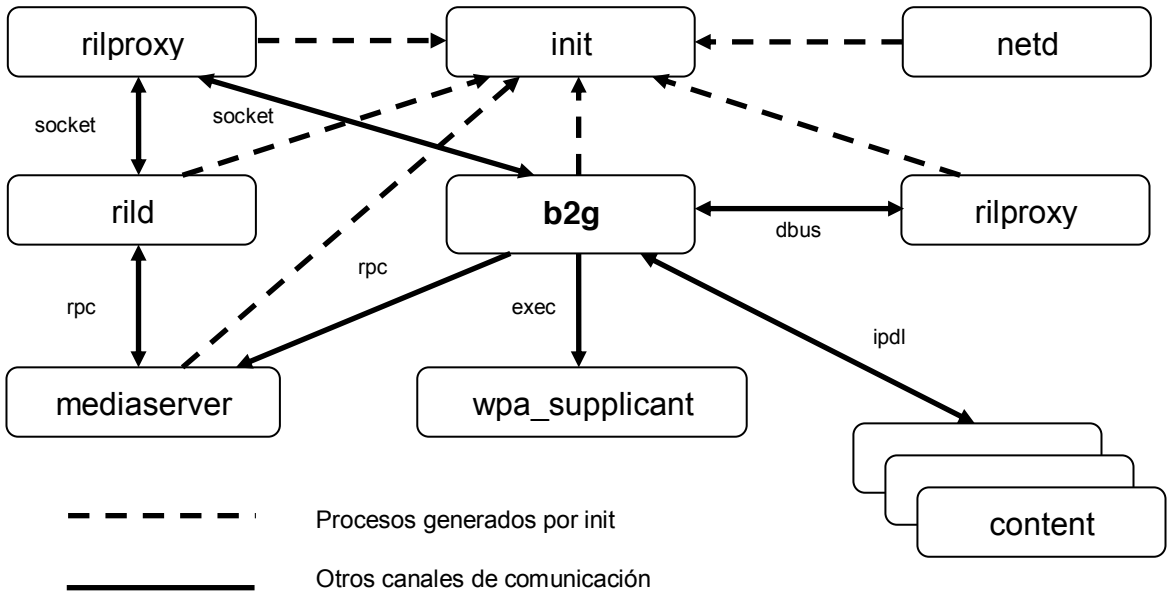


Ilustración 3 - Arquitectura del proceso de usuario

- B2g

El proceso *b2g* es el proceso primario de sistema. Se ejecuta con privilegios; tiene acceso a la mayoría del hardware. *b2g* se comunica con el módem, almacena en el búfer de pantalla e interactúa con el GPS, cámaras y otros dispositivos. Internamemte, se ejecuta en una capa de Gecko.

El proceso *b2g* genera una serie de procesos que albergan el conjunto de aplicaciones web y otros contenidos. Estos procesos se

comunican con el proceso principal del servidor Gecko a través de IPDL, un sistema de envio de mensajes.

Además, *b2g* ejecuta *lixbul*, que hace referencia a b2g/app/b2g.js para obtener las preferencias de fábrica. De las preferencias se abrirá el archivo HTML b2g/chrome/content/shell.html, que se compila en un archivo omni.ja. El shell.html incluye el archivo b2g/chrome/content/shell.js, que activa la aplicación system de Gaia.

- Rild

Este proceso es el demonio que implementa la interfaz de la radio (RIL – Radio Interface Layer). Este código está implementado directamente por el fabricante del hardware. Se inicia con un código como este en el script *init*:

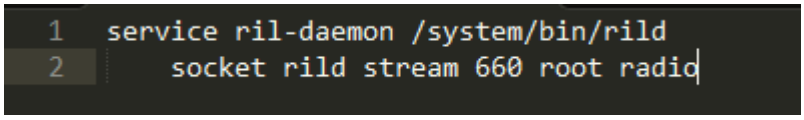


Ilustración 4 - Inicio del servicio rild

- Rilproxy

Es el cliente del proceso *rild*. Este actúa como un proxi inerte de envio (dumb proxy) entre *rild* y *b2g*.

- Mediaserver

El proceso *mediaserver* controla la reproducción de audio y video. Gecko se comunica con él a través de un mecanismo de llamada de procedimiento remoto de Android (Android Remote Procedure Call (RPC)). Algunos de los contenidos multimedia que Gecko puede reproducir (OGG Vorbis audio, OGG Theora video, y WebM video) son decodificados por Gecko y enviados directamente al proceso *mediaserver*. Otros archivos multimedia son decodificados por *libstagefright*, que puede acceder a códecs del fabricante y codificadores del hardware.

Hay que tener en cuenta que el proceso *mediaserver* es un componente provisional de Firefox OS; existe sólo para ayudar en el trabajo de desarrollo inicial pero se espera que se descarte con el tiempo; lo que seguramente no ocurrirá antes de la version 2.0 de Firefox OS.

- Netd

El proceso *netd* se usa para configurar los interfaces de red del dispositivo.

- Wpa_supplicant

El proceso *wpa_supplicant* es un demonio estándar de estilo UNIX que maneja la conectividad con los puntos de acceso WiFi

- Dbus-daemon

El *dbus-daemon* implementa el D-Bus, un sistema de mensajes de bus que Firefox OS emplea para las comunicaciones por Bluetooth.

2.2.5. Gecko

Gecko, como se mencionó previamente, es la implementación de estándares web (HTML, CSS, y JavaScript) que se usa para implementar todo lo que el usuario ve en Firefox OS.

2.2.6. Archivos de Gecko relacionados con Firefox OS

La carpeta *b2g* contiene mayoritariamente funciones relacionadas con Firefos OS.

- b2g/chrome/content: contiene archivos de Javascript ejecutados sobre la aplicación de sistema.
- b2g/chrome/content/shell.html: es el punto de entrada a Gaia.
- b2g/chrome/content/shell.js: es el primer script que se carga en la aplicación de sistema de Gaia. Importa todos los módulos

requeridos, registra los detectores de clave (key listeners), define las funciones *sendCustomEvent* y *sendChromeEvent* para que se cominiquen con Gaia, y provee ayudantes de instalación de aplicaciones web: *indexedDB*, *RemoteDebugger*, el ayudante de teclado, y la herramienta para captura de pantalla. No obstante, la función más importante de *shell.js* es lanzar la aplicación de sistema de Gaia, y después entregarle todo el trabajo general de administración del sistema.

- b2g/app/b2g.js: contiene configuraciones predefinidas, como *about:config* en el navegador, y la misma que *pref.js* de Gaia. Estas configuraciones se pueden cambiar desde la aplicación de congifuraciones y se pueden sobreescribir con *user.js* en el script de construcción de Gaia.

Las nuevas implementaciones de la API (post-b2g) se encuentran en la carpeta *dom/*, las anteriores pasan a la carpeta *dom/base.*

- dom/apps: implementaciones de APIs.
- dom/apps/PermissionsTable.jsm: definición de permisos.

Para definir las web APIs se usa el lenguaje WebIDL. Los archivos relacionados con la capa de adaptación *gonk* se encuentran en el directorio *hal/gonk.* Los archivos de configuración se *encuentran* en el directorio *module/libpref/src/init/all.js*. El paquete de estilos para los recursos del dispositivo se encuentra en los archivos */system/b2g/ omni.ja* y */system/b2g/ omni.js*

2.2.6.1. Proceso de eventos de entrada

La mayor parte de las acciones en Gecko se activan por acciones de usuario. Estas acciones se representan por eventos de entrada (tales como presionar botones o tocar la pantalla). Estos eventos se comunican con Gecko a través de *nsIAppShell*, que es una interfaz de Gecko empleada para representar los puntos de entrada primaria de una aplicación, es decir, el controlador del dispositivo de entrada llama a

métodos en el objeto *nsAppShell* que representa el subsistema de Gecko, para así enviar eventos a la interfaz de usuario. Ejemplo:

```
void GeckoInputDispatcher::notifyKey(nsecs_t eventTime,
                                     int32_t deviceId,
                                     int32_t source,
                                     uint32_t policyFlags,
                                     int32_t action,
                                     int32_t flags,
                                     int32_t keyCode,
                                     int32_t scanCode,
                                     int32_t metaState,
                                     nsecs_t downTime) {
  UserInputData data;
  data.timeMs = nanosecsToMillisecs(eventTime);
  data.type = UserInputData::KEY_DATA;
  data.action = action;
  data.flags = flags;
  data.metaState = metaState;
  data.key.keyCode = keyCode;
  data.key.scanCode = scanCode;
  {
    MutexAutoLock lock(mQueueLock);
    mEventQueue.push(data);
  }
  gAppShell->NotifyNativeEvent();
}
```

Ilustración 5 – Captura de eventos

Estos eventos provienen del sistema estándar Linux *input_event*. Firefox OS emplea *light abstraction layer* sobre eso; lo que provee algunas características útiles como filtrar los eventos. Se puede ver el código que crea eventos de ingreso en el método *EventHub::getEvents()* que se encuentra en *widget/gonk/libui/EventHub.cpp*.

Una vez que Gecko recibe los eventos, se envía a DOM por *nsAppShell*:

```
static nsEventStatus sendKeyEventWithMsg(uint32_t keyCode,
                                         uint32_t msg,
                                         uint64_t timeMs,
                                         uint32_t flags) {
    nsKeyEvent event(true, msg, NULL);
    event.keyCode = keyCode;
    event.location = nsIDOMKeyEvent::DOM_KEY_LOCATION_MOBILE;
    event.time = timeMs;
    event.flags |= flags;
    return nsWindow::DispatchInputEvent(event);
}
```

Ilustración 6 - Envío de eventos

Una vez en el DOM, los eventos son consumidos o enviados a aplicaciones web para ser procesados posteriormente.

2.2.6.2. Gráficos

A bajo nivel, Gecko emplea *OpenGL ES 2.0* para establecer un contexto. Se realiza dentro de la implementación de Gonk en *nsWindow* por medio de un código similar:

```
1  gNativeWindow = new android::FramebufferNativeWindow();
2  sGLContext = GLContextProvider::CreateForWindow(this);
```

Ilustración 7 - Inicio de gráficos

La clase *FramebufferNativeWindow se* obtiene directamente desde Android. Esto emplea la API *gralloc* para acceder al controlador de gráficos con el fin de mapear los búfers del dispositivo *framebuffer* a la memoria del dispositivo.

Gecko emplea su sistema de capas para componer el contenido dibujado en la pantalla. En resumen, ocurre lo siguiente:

Gecko dibuja regiones distintas de las páginas en los búfers de memoria, A veces, estos búfers están en la memoria del sistema; otras veces, son texturas mapeadas en el espacio de direcciones de Gecko, lo que siginifica que Gecko está dibujando directamente en la memoria de video. Esto se realiza generalmente mediante el método *BasicThebesLayer::PaintThebes()*.

Entonces, Gecko, compone todas estas texturas en la pantalla empleando comandos OpenGL. La composición se realiza mediante *ThebesLayerOGL::RenderTo()*.

2.2.6.3. Capa de abstracción hardware (HAL)

La capa de abstracción de hardware de Gecko es una de sus capas de adaptación (porting). Gestiona los accesos de bajo nivel a las interfaces del sistema a través de múltiples plataformas que emplean

una API de C++ a la que se accede en los niveles superiores de Gecko. Estas APIs se implementan de plataforma a plataforma dentro de la HAL de Gecko. Esta capa de abstracción de hardware no se expone directamente a código javaScript dentro de Gecko.

Para describir el funcionamiento de HAL tomaremos como ejemplo la API de vibración (vibration). La HAL para esta API se define en *hal/Hal.h*. Simplificando el método de firma para hacerlo más claro, resulta esta función:

```
1  void Vibrate(const nsTArray<uint32> &pattern);</uint32>
```

Ilustración 8 - HAL del API de vibración

Esta es la función que el código de Gecko llama para activar la vibración del dispositivo de acuerdo con un patrón específico, existe otra para cancelar la vibración activa. La implementación de GONK para este método está en *hal/gonk/GonkHal.cpp*:

```
1  void Vibrate(const nsTArray<uint32_t> &pattern) {
2    EnsureVibratorThreadInitialized();
3    sVibratorRunnable->Vibrate(pattern);
4  }
5  </uint32_t>
```

Ilustración 9 - Implementación del método para la vibración

Este código envía la petición para el inicio de la vibración a otro conjunto de procesos, que se implementa en *VibratorRunnable::Run()*. El bucle principal de este conjunto se ve así:

```
1  while (!mShuttingDown) {
2    if (mIndex < mPattern.Length()) {
3      uint32_t duration = mPattern[mIndex];
4      if (mIndex % 2 == 0) {
5        vibrator_on(duration);
6      }
7      mIndex++;
8      mMonitor.Wait(PR_MillisecondsToInterval(duration));
9    }
10   else {
11     mMonitor.Wait();
12   }
13 }
```

Ilustración 10 - Procesamiento de la petición de vibración

vibrator_on() es la API HAL de Gonk que activa el motor de vibración. Internamente, este método envía un mensaje al controlador del núcleo (kernel driver), escribiendo un valor en un objeto de kernel mediante *sysfs*.

Las APIs HAL de Gecko tienen soporte en todas las plataformas. Cuando se construye Gecko para una plataforma que no tiene una interfaz para los motores de vibración (como una computadora de escritorio), se usa una implemenación de *fallback* de la API de HAL. Para la vibración, se emplea en *hal/fallback/FallbackVibration.cpp*.

```
1  void Vibrate(const nsTArray<uint32_t> &pattern) {
2  }</uint32_t>
```

Ilustración 11 - Fallback para vibración

La mayoría del contenido de la red se ejecuta en procesos con privilegios bajos, por lo que no podemos suponer que esos procesos tienen los privilegios necesarios para poder (por ejemplo) activar o desactivar el motor de vibración. Además, queremos tener una ubicación central para controlar las posibles condiciones de carrera (*race conditions*). En la HAL de Gecko, esto se realiza por medio de la implementación de una *sandbox*. Esta *sandbox* simplemente funciona como un proxy para la peticiones realizadas por los procesos y las reenvía al proceso del servidor Gecko. Las peticiones de proxy se envían empleando IPDL. Para la vibración, la función *Vibrate()* se encarga de la gestión y se implementa en *hal/sandbox/SandboxHal.cpp*:

```
1  void Vibrate(const nsTArray<uint32_t>& pattern, const WindowIdentifier &id) {
2    AutoInfallibleTArray<uint32_t, 8=""> p(pattern);
3
4    WindowIdentifier newID(id);
5    newID.AppendProcessID();
6    Hal()->SendVibrate(p, newID.AsArray(), GetTabChildFrom(newID.GetWindow()));
7  }</uint32_t,></uint32_t>
```

Ilustración 12 - Implementación de la vibración en Sandbox

Este código envía un mensaje definido por la interfaz PHal, descrita por IPDL en *hal/sandbox/PHal.ipdl*. El método se describe aproximadamente de la siguiente manera:

```
1  Vibrate(uint32_t[] pattern);
```

Ilustración 13 – Método Vibrate()

El receptor de este mensaje es *HalParent::RecvVibrate(),* un método incluido en *hal/sandbox/SandboxHal.cpp*:

```
1  virtual bool RecvVibrate(const InfallibleTArray<unsigned int="">& pattern,
2                const InfallibleTArray<uint64_t> &id,
3                PBrowserParent *browserParent) MOZ_OVERRIDE {
4
5    hal::Vibrate(pattern, newID);
6    return true;
7  }</uint64_t></unsigned>
```

Ilustración 14 – Método de recepción de Vibrate()

Se omiten algunos detalles que no son relevantes pero se puede observar como el mensaje progresa de un proceso de Gecko hasta Gonk, luego a la implementación de la HAL de Gonk *Vibrate()*, y finalmente al controlador de vibración.

2.2.6.4. APIs DOM

Las interfaces DOM son la forma en la que el contenido web se comunica con Gecko. Las interfaces DOM se definen empleando IDL, que compone una interfaz de función foránea (foreign function interface) o FFI un objeto modelo (OM) entre JavaScript y C++.

La API de vibración se ofrece al contenido web por medio de una interfaz IDL, que se provee en *nsIDOMNavigator.idl*

```
1  [implicit_jscontext] void mozVibrate(in jsval aPattern);
```

Ilustración 15 – Interfaz IDL de vibración

El argumento *jsval* indica que *mozVibrate()* acepta como parámetro cualquier valor de JavaScript. El compilador IDL, *xpidl*, genera una

interfaz C++ que es implementada por la clase *Navigator* en *Navigator.cpp*.

```
1  NS_IMETHODIMP Navigator::MozVibrate(const jsval& aPattern, JSContext* cx) {
2    // ...
3    hal::Vibrate(pattern);
4    return NS_OK;
5  }
```

Ilustración 16 - Interfaz MozVibrate

Este método tiene más código pero no es importante y se ha eliminado para centrarnos en el tema que nos ocupa. La llamada a *hal::Vibrate()* transfiere el control de DOM a la HAL Gecko. Desde allí, entramos en la implementación de HAL tratada en la sección previa y continuamos hacia el controlador. Además, la implementación DOM no se preocupa de la plataforma en la que está corriendo (Gonk, Windows, OS X, o cualquier otra). Tampoco le interesa si el código está corriento en un proceso de contenido o en un proceso del servidor Gecko. Esos detalles se dejan de lado para que los gestionen los procesos de bajo nivel del sistema.

La API de vibración es muy simple, lo que la convierte un ejemplo excelente. La API de SMS es más compleja porque emplea su propia capa de envío remoto que conecta los procesos de contenido con el servidor.

2.2.6.5. Interfaz de radio RIL

La interfaz RIL ya fue mencionada en el apartado de arquitectura. En esta sección vamos a ver una serie de aspectos para analizarla con mayor detalle.

- rild: es el demonio que se comunica con el firmware del componente.
- rilproxy: es el demonio que hace de proxy entre *rild* y *Gecko*, implementado en el proceso *b2g*. Con este se solucionan problemas de permisos debidos a intentos de interactuar

directamente con *rild*, ya que *rild* sólo puede comunicarse con los integrantes del grupo *radio*.

- b2g: el código perteneciente a la capa de interfaz de radio está en el fichero *dom/system/gonk/ril_worker.js*, que implementa un subproceso para la comunicación entre *rild* y *rilproxy,* y el estado de la radio y la interfaz *nsIRadioInterfaceLayer*, que es el hilo principal del servicio *XPCOM*, cuya función fundamental es el intercambio de mensajes entre el hilo *ril_worker.js* y otros componentes de Gecko.

2.2.6.6. Datos 3G

A través de un mensaje RIL se inicia una llamada de datos, esto permite activar el modo de transferencia de datos en el módem. Esta llamada de datos crea y activa una interfaz mediante protocolo punto a punto (PPP) con el dispositivo, en el kernel de Linux, que se puede configurar mediante las interfaces habituales.

2.3. Fundamentos de una aplicación Firefox OS

Una aplicación Firefox OS es simplemente una Open Web App. Las Open Web Apps permiten a los desarrolladores crear aplicaciones HTML5 que se ejecutan en múltiples dispositivos (escritorio, móvil, tableta, etc.), mediante el uso de estándares web y tecnologías abiertas como HTML5, CSS y JavaScript. Asímismo, pretenden crear un rico ecosistema distribuido en tiendas de aplicaciones HTML5, incluido el Marketplace operado por Mozilla. Con esto Mozilla se propone conseguir que los desarrolladores vuelvan a tener el control de todos los aspectos de las aplicaciones: desde el desarrollo fácil para distribución hasta la gestión de la relación con los clientes directos.

Las aplicaciones se desarrollan usando las tecnologías web estándares a las que se agregan metadatos que permiten al usuario descubrir, instalar, lanzar y otorgarles privilegios adicionales.

Es importante señalar que, aunque Mozilla ofrezca su Marketplace para ayudar a distribuir sus aplicaciones, no es obligatorio distribuir las aplicaciones mediante el Marketplace.

2.3.1. Tipos de aplicaciones en Firefox OS

Las aplicaciones de Firefox OS están clasificadas en distintos tipos atendiendo al modo de instalación/ejecución y a los privilegios que el sistema operativo les otorga.

- **Aplicaciones Hosted:** son Webapps publicadas en modo ejecutable en un servidor Web. Pueden ejecutarse desde un navegador o publicarse en una tienda a través de su URL para su instalación en un dispositivo Firefox OS. Tienen ciertas restricciones pues sólo pueden ser ejecutadas con conexión a Internet y sólo puede haber una hosted Webapp por dominio Web.
- **Aplicaciones Packaged:** son Webapps empaquetadas en un fichero ZIP. Al igual que las Hosted se pueden publicar en tiendas para su instalación en dispositivos Firefox OS pero a direfencia de éstas no se pueden ejecutar directamente en un navegador. Son autónomas, no necesitan conexión a Internet para funcionar. Según los privilegios que tengan sobre el hardware del dispositivo se pueden clasificar en:
  - *Plain:* aplicaciones Web convencionales
  - *Certified:* con acceso a elementos restringidos
  - *Privileged:* con acceso total a todos los recursos del móvil

2.3.2. Tiendas de Apps

Una tienda de aplicaciones para Firefox OS puede crearse en cualquier servidor que aloje Apps. Las tiendas en Firefox OS compiten entre sí, serán más o menos populares dependiendo de la aceptación de los usuarios y más o menos seguras según el control de seguridad que se haya realizado. La tienda oficial de la fundación Mozilla es: https://marketplace.firefox.com

2.3.3. Requerimientos de las aplicaciones

2.3.3.1. Hosted Apps

Para crear una hosted App es necesario disponer de lo siguientes componentes:

- El código de la aplicación en cuestión, es decir, los ficheros HTML, Javascript y CSS que la componen.
- Instalador, un fichero HTML que contendrá el código para la instalación de la aplicación en el dispositivo
- Un conjunto de iconos que se usarán para el acceso directo a la aplicación desde el escritorio
- El archivo de manifiesto *manifest.webapp*

2.3.3.2. Packaged Apps

Los componentes necesarios para crear una packaged App serán los siguientes:

- Instalador, un fichero HTML que contendrá el código para la instalación de la aplicación en el dispositivo
- Archivo de manifiesto *package.manifest*
- App empaquetada en un fichero *zip*, que contendrá los mismos ficheros que la hosted App

2.3.3.3. Estructura de los archivos de manifiesto

Estructura del archivo *manifest.webapp* para una hosted App:

```
{
  "version": "1.0",                                    //version
  "name": "MoDe",                                      //nombre que aparece en el escritorio
  "description": "Monitor Deportivo",                  //descripción para una tienda
  "launch_path": "/app.html",                          //URL de Webapp en servidor
  "icons": {                                           //iconos de escritorio
    "128": "/images/app_icon_128.png",
    "60": "/images/app_icon_60.png"
  },
  "developer": {                                       //equipo o empresa desarrolladora
    "name": "Juan A. Romero Olmo",
    "url": ""
  },
  "permissions": {                                     //permisos
    "geolocation": {
        "description" : "Marking out user location"
    }
}
```

Ilustración 17 - Ejemplo de fichero manifest.webapp

Estructura del archivo *package.manifest* para una packaged App:

```
{
  "version": "1.0",                                    //version
  "name": "MoDe",                                      //nombre que aparece en el escritorio
  "description": "Monitor Deportivo",                  //descripción para una tienda
  "package_path": "/app.html",                         //URL a paquete en tienda
  "developer": {                                       //equipo o empresa desarrolladora
    "name": "Juan A. Romero Olmo",
    "url": ""
  },
  "permissions": {                                     //permisos
    "geolocation": {
        "description" : "Marking out user location"
    }
}
```

Ilustración 18 – Ejemplo de fichero package.manifest

2.3.3.4. Instalador de Apps

El instalador es un archivo HTML que contendrá el script de instalación. Mediante este, podemos testear en el simulador aplicaciones que estén alojadas en un servidor remoto. A modo de ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Instalador de Rutas</title>
        <meta charset="UTF-8">
    </head>

    <body>

        <p>Página de instalación</p>

        <script>

            (function() {

                var manifestUrl = location.href.replace("install.htm", "") + "manifest.webapp";

                if (! navigator.mozApps.installPackage) { //Comprobación de compatibilidad
                    alert("ERROR: Esta aplicación no es compatible con tu dispositivo");
                    return;
                }

                var req = navigator.mozApps.installPackage(manifestUrl); //instalación del paquete
                req.onsuccess = function(){ //instalación correcta
                    alert("Instalación completada."+this.result.origin);
                }
                req.onerror = function(){ //instalación errónea
                    alert("Error de instalación: "+this.error.name);
                }
            )();
        </script>
    </body>
</html>
```

Ilustración 19 - Ejemplo de fichero de instalación en una packaged App

Para aplicaciones que se encuentren en las misma máquina, bastará con añadir el directorio desde el Dashboard de Firefox OS Simulator.

2.3.4. Firefox OS Simulator

El complemento Simulador de Firefox OS, es una herramienta que permite probar y depurar aplicaciones para Firefox OS en el escritorio, a través del navegador de Firefox, **sólo para la versión 1.1** del sistema operativo. El ciclo de «programar–probar–depurar» es mucho más rápido con el Simulador que con un dispositivo de hardware real y, obviamente, no necesitará uno para utilizarlo. En esencia, este complemento consiste en:

El simulador: éste incluye el cliente de escritorio de Firefox OS, que es una versión de las capas superiores de Firefox OS que se ejecuta en su sistema operativo de escritorio. El Simulator también incluye algunas funciones de emulación adicionales que no están disponibles en las compilaciones de Firefox OS para escritorio estándares.

El dashboard: una herramienta ubicada en el navegador Firefox que permite iniciar y detener el Simulador; e instalar, desinstalar y depurar las aplicaciones que tiene instaladas. El dashboard permite también, instalar aplicaciones en un dispositivo real, y realiza comprobaciones a los archivos manifest para detectar problemas comunes.

La siguiente captura de pantalla muestra una sesión de depuración utilizando el Simulador.

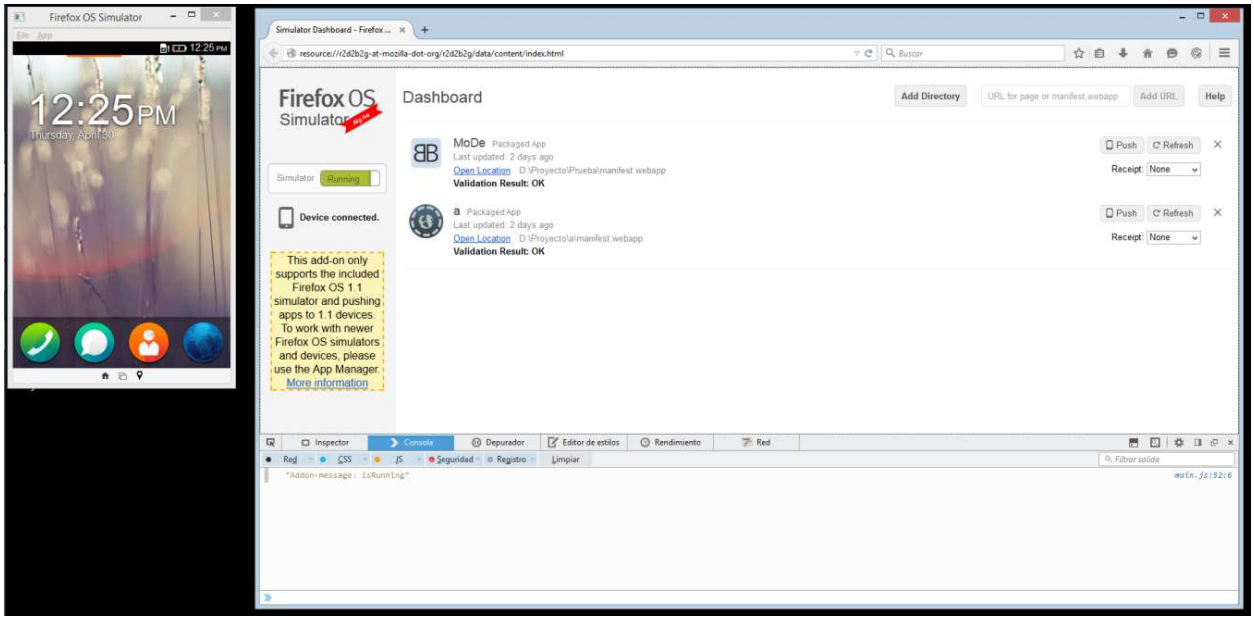


Ilustración 20 - Complemento Firefox OS simulator en el navegador Firefox

El dashboard se encuentra en el cuadrante superior derecho de la imagen, dentro del navegador y ubicado en una pestaña de Firefox. En él, podemos observar las aplicaciones que se han añadido. Hemos añadido dos aplicación packaged, llamada “a” y “MoDe”. Asímismo, hemos conectado las herramientas de depuración que podemos ver en la parte inferior del navegador. En la parte izquierda temenos el simulador ejecutándose.

2.3.5. App Manager

El *App Manager* es una herramienta disponible en Firefox para escritorio, que provee una cantidad de útiles herramientas para ayudarte a probar, desarrollar y depurar aplicaciones HTML5 en teléfonos Firefox OS y el Simulador de Firefox OS, directamente en tu navegador. Similar a Firefox OS Simulator para **versiones** del sistema operativo **posteriores a la 1.1**.

Elementos del App Manager:

- *Apps panel*, que maneja apps locales (aplicaciones cuyo código fuente está en tu equipo) y apps alojadas externamente, perimitiéndote empaquetarlas e instalarlas en tu dispositivo o simulador, y depurarlas usando *Toolboxes.*

- *Device panel*, que muestra información sobre el dispositivo conectado, incluyendo la versión de Firefox OS instalada, los permisos requeridos para usar las APIs en el dispositivo, y las apps instaladas.

- *Toolboxes*, que son conjuntos de aplicaciones para el desarrollador (consola web, inspecto, depurador, etc.) que pueden ser conectadas a una aplicación en ejecución vía el *Apps panel* para realizar operaciones de depuración.

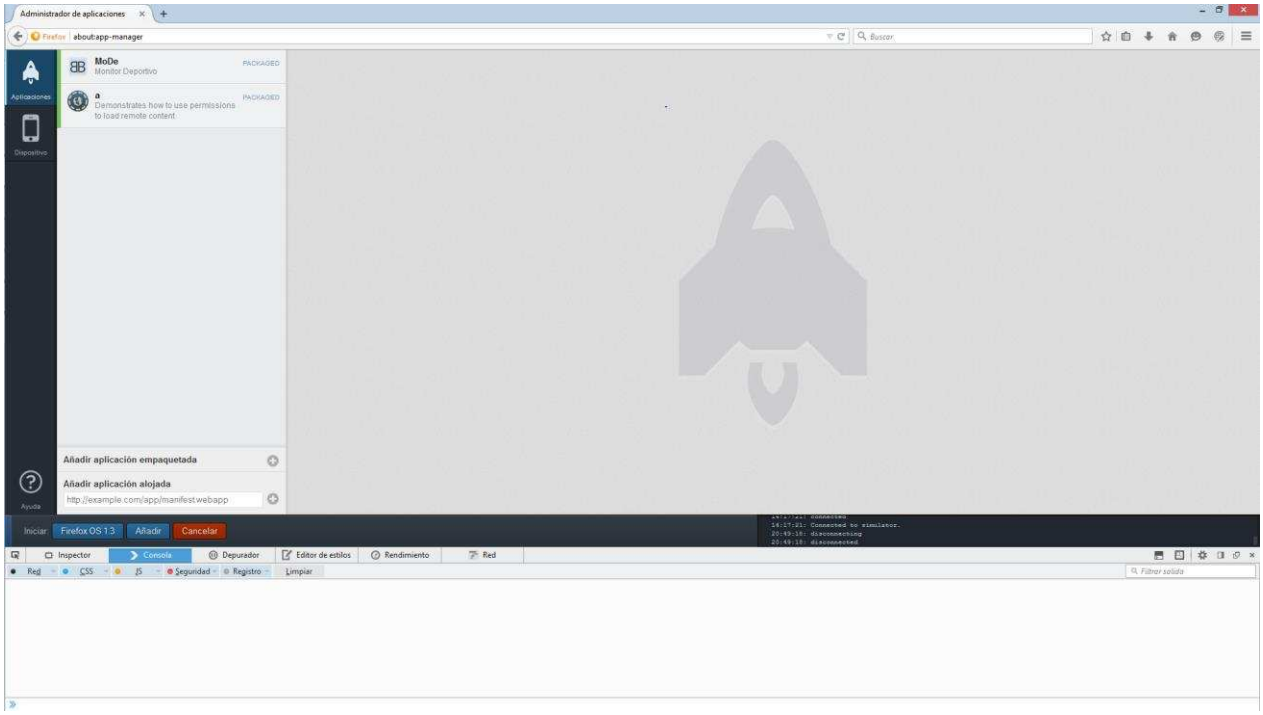


Ilustración 21 – App Manager

En la ilustración 21 se pueden observar los elementos del App Manager descritos anteriormente. En la barra situada encima del panel de depuración se pueden observar las versiones del sistema operativo con las que podemos iniciar el simulador. Si necesitamos alguna distinta se puede instalar pulsando el botón *Añadir,* que nos llevará a la web que muestra las opciones posibles. Las opciones de depuración son las mismas que en *Firefox OS Simulator* para la versión 1.1.

2.3.6. WebIDE

WebIDE reemplaza a *App Manager*. Al igual que *App Manager*, este permite ejecutar y debugear aplicaciones *Firefox OS* utilizando *Firefox OS Simulator,* para la versión 1.2 y posteriores o un dispositivo con *Firefox OS* real también con la versión 1.2 o superior.

WebIDE, proporciona un entorno de edición para crear y desarrollar aplicaciones para *Firefox OS*, incluye una vista de árbol jerárquica de todos los archivos de la aplicación con la posibilidad de editarlos y salvar los cambios realizados, y también dos plantillas de aplicación para ayudar a comenzar a desarrollar.

Para desarrollar y depurar aplicaciones utilizando *WebIDE*, todo lo que se necesita es el navegador *Firefox* en su versión 33 o posterior. Para hacer pruebas en un dispositivo de Firefox OS real, necesita un dispositivo que ejecute *Firefox OS 1.2* o posterior, y un cable USB.

Para abrir *WebIDE* podemos usar distintos métodos. Hacer clic en *WebIDE,* en el menú *Web Developer* (Desarrollador Web), usar el atajo de teclado ***Shift+F8*** o hacer clic en el icono ubicado en la barra de herramientas, que estará siempre visible cuando se haya abierto *WebIDE* al menos una vez.

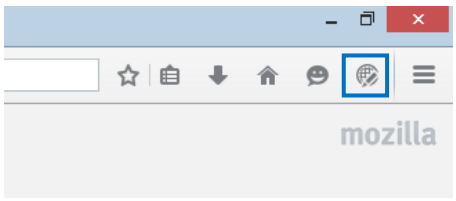


Ilustración 22 - Icono de acceso a WebIDE

Una vez abierto veremos la siguiente ventana:

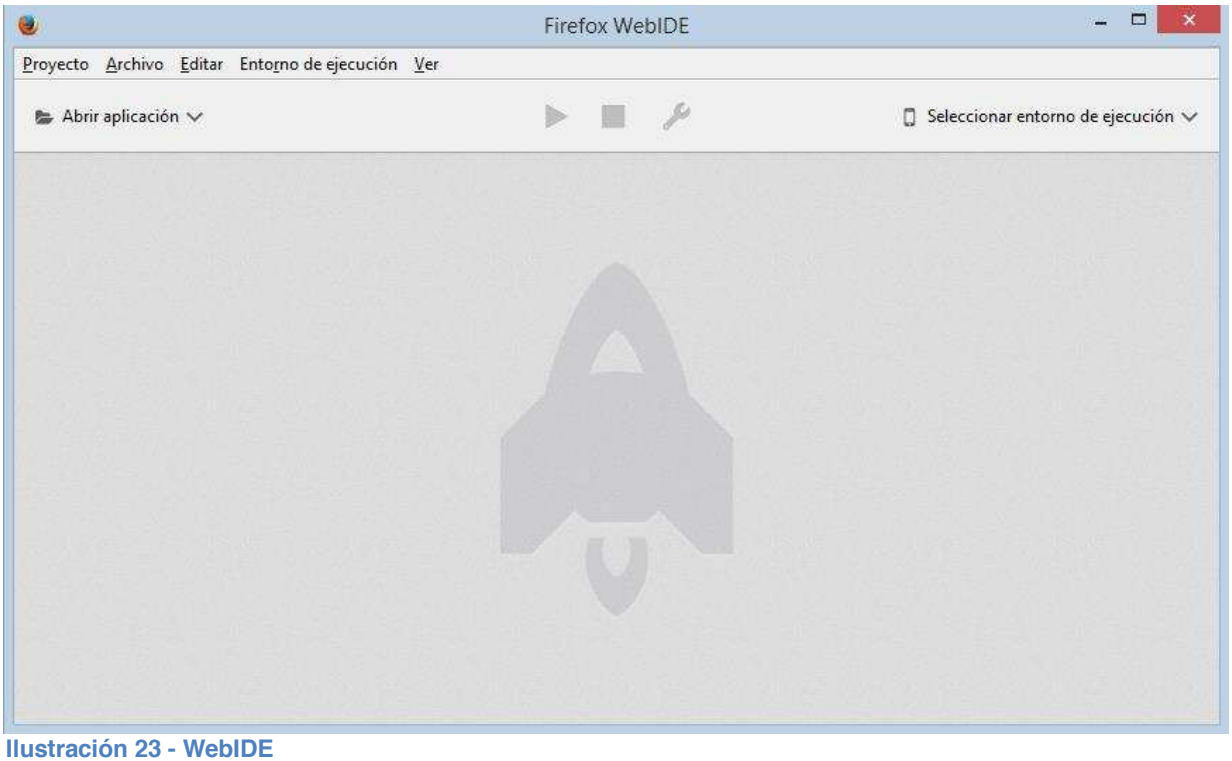


Ilustración 23 - WebIDE

El menú desplegable de la izquierda con la etiqueta *Abrir aplicación,* permite abrir aplicaciones existentes o crear nuevas aplicaciones mientras que el de la derecha, con la etiqueta *Seleccionar entorno de ejecución* permite

seleccionar el entorno donde vamos a ejecutar la aplicación. Los botones del centro, ejecutan, detienen, y depuran la aplicación, estos sólo se activan cuando se ha abierto una aplicación y se selecciona un entorno de ejecución.

Desde la versión 36 de *Firefox*, se puede cambiar el tamaño de la fuente de todo el *WebIDE* usando los atajos de teclado estándares:

Ctrl o *Command +,* aumenta el tamaño de la fuente

Ctrl o *Command -,* disminuye el tamaño de la fuente

Ctrl o *Command 0*, vuelve al tamaño por defecto

2.3.6.1. Entornos de ejecución.

En el el menú desplegable *Seleccione entorno de ejecución*, se muestran los tipos de entorno que podemos elegir:

- **Dispositivos USB**: Dispositivos con sistema operativo *Firefox OS* conectados a través de USB.
- **Simuladores**: Instancias del simulador *Firefox OS* que hemos instalado en WebIDE.
- **Otros**: Entornos de ejecución remota para conectarse a *WebIDE* utilizando un nombre de servidor y un puerto.

Antes de conectar un dispositivo *Firefox OS*, se deben revisar algunas cuestiones:

La versión de *Firefox OS* del dispositivo debe ser la 1.2 o superior. Para revisar la versión, se debe acceder a *Configuración* en el dispositivo y verificarla en *Información*. Además, se debe habilitar la depuración remota, para ello accederemos, en la barra superior del teléfono, a *Configuración->Más información->Desarrollador* y seleccionamos la casilla *Depuración remota*. En la versión 1.3 y anteriores será suficiente marcar la casilla, mientras que en la versión 1.4 y superiores *Depuración*

Remota pedirá habilitar *ADB* o *ADB y DevTools*. Se debe seleccionar *ADB y DevTools*. Es buena idea, además, deshabilitar el bloqueo de pantalla en el dispositivo, pues cuando la pantalla se bloquea, la conexion del teléfono se pierde, esto significa que no esta disponible para depuración. Para deshabilitarlo se accede a *Configuración->Bloqueo de pantalla* y se desactiva.

En el apartado de *Simuladores* se pueden añadir las versiones que se vayan a usar. Bastará con seleccionar la opción *Instalar Simulator,* en el menú de *Entorno de ejecución remoto,* que nos desplegará una ventana en la que se podrán escoger los simuladores a instalar.

En el apartado *Otros* se puede usar un nombre de host y un puerto arbitrario para conectarse a un dispositivo remoto. Por defecto, *WebIDE* usa un complemento llamado *ADB Helper* que configura un puerto para que el escritorio de herramientas de Firefox pueda intercambiar mensajes con el dispositivo. Esta opción es adecuada en la mayoría de casos, pero a veces se necesitará usar *ADB* fuera de *WebIDE*: por ejemplo, se podría estar usando *ADB* directamente desde línea de comandos. En ese caso, se conectará al dispositivo especificando un host y un puerto, usando el comando *adb forward*. Si, además, se quiere conectar *WebIDE*, se debe desactivar el complemento *ADB Helper* y conectar *WebIDE* usando la opción *Entorno de ejecución remoto*, introduciendo el host y puerto que se ha asignado a *adb forward*.

Cuando se selecciona un entorno de ejecución, en el menú desplegable *Seleccionar entorno de ejecución* aparecen tres opciones adicionales:

- **Información del entorno de ejecución:** muestra información del entorno de ejecución seleccionado.
- **Tabla de permisos:** tabla de permisos de la app para el entorno de ejecución usado, para cada API y tipo de app. Los valores son, (✓) permitido, (✗) denegado, o (!) si se le pide al usuario.

- **Captura de pantalla:** que permite realizar capturas de pantalla desde el entorno de ejecución.

2.3.6.2. Menú Abrir aplicación

En este menú se pueden encontrar tres opciones: *Nueva aplicación*, *Abrir aplicación empaquetada* y *Abrir aplicación alojada*.

Si se elige la opción *Nueva aplicación,* se abre una ventana en la que hay posibilidad de elegir tres opciones *HelloWorld, Privileged Empty App* o *Privileged App*. Estas plantillas proporcionan una estructura básica para empezar a realizar una app. A la app se le debe asignar un nombre, todos los archivos relacionados se guardarán en una carpeta seleccionada por el usuario.

Para abrir una aplicación empaquetada, se selecciona la opción *Abrir aplicación empaquetada* y se selecciona la carpeta contenedora del proyecto. Si se desea abrir una aplicación alojada, se selecciona la opción *Abrir aplicación alojada* y se introduce la URL del archivo manifest correspondiente.

2.3.6.3. Editor

WebIDE integra un editor con el que se pueden escribir y modificar las aplicaciones.

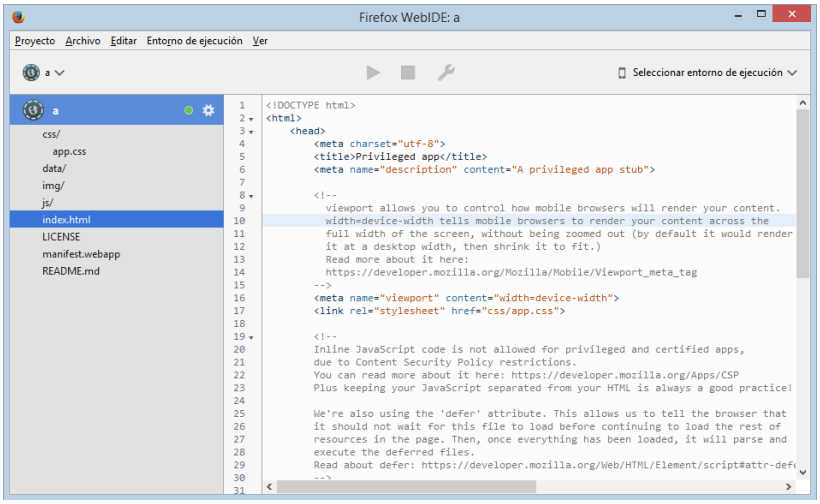


Ilustración 24 - Editor de WebIDE

En la parte izquierda del panel se muestra el árbol de ficheros y directorios de la aplicación. En la parte derecha se observa el código del fichero que se haya seleccionado, con la numeración de líneas y se ofrece la posibilidad de editar el código. Si el fichero seleccionado es una imagen, esta se visualiza en el panel de la derecha. En el menú *Archivo* se encuentran los comandos para guardar (Ctrl + S/Cmd + S) o crear un nuevo archivo.

Cuando se abre un proyecto, WebIDE valida automáticamente el archivo *manifest* y nos indica si existe algún error. Además se muestra información del tipo, descripción, ruta e id de la aplicación. Para volver a ver esta información una vez editado algún fichero bastará con hacer clic en el nombre del proyecto en el árbol de directorios.

El editor ofrece la posibilidad de usar atajos de teclado:

	Windows	OS X	Linux
Ir a la línea…	Ctrl + J	Cmd + J	Ctrl + J
Buscar en fichero	Ctrl + F	Cmd + F	Ctrl + F
Buscar de nuevo	Ctrl + G	Cmd + G	Ctrl + G
Seleccionar todo	Ctrl + A	Cmd + A	Ctrl + A
Cortar	Ctrl + X	Cmd + X	Ctrl + X
Copiar	Ctrl + C	Cmd + C	Ctrl + C
Pegar	Ctrl + V	Cmd + V	Ctrl + V
Deshacer	Ctrl + Z	Cmd + Z	Ctrl + Z
Rehacer	Ctrl + Y	Cmd + Y	Ctrl + Y
Indentar	Tab	Tab	Tab
Desindentar	Shift + Tab	Shift + Tab	Shift + Tab
Mover línea arriba	Alt + Arriba	Alt + Arriba	Alt + Arriba
Mover línea abajo	Alt + Abajo	Alt + Abajo	Alt + Abajo
Comentar/Descomentar líneas	Ctrl + /	Ctrl + /	Ctrl + /

Además, cuando se edita código CSS o JavaScript proporciona sugerencias de autocompletado. Las sugerencias de código JavaScript se pueden activar o desactivar pulsando Ctrl + Espacio.

2.3.6.4. Ejecución y depuración

WebIDE, como en el navegador *Firefox*, dispone de una serie de herramientas de depuración. Para activarlas primero se debe ejecutar la aplicación, para ello seleccionaremos un entorno de ejecución y ejecutaremos la aplicación pulsando el botón ▶ *(instalar y ejecutar)*, del panel central. Una vez ejecutada, en este mismo panel, se activa el botón 🔧 *(Activar depuración)*, al pulsarlo aparece en la parte inferior el panel de depuración.

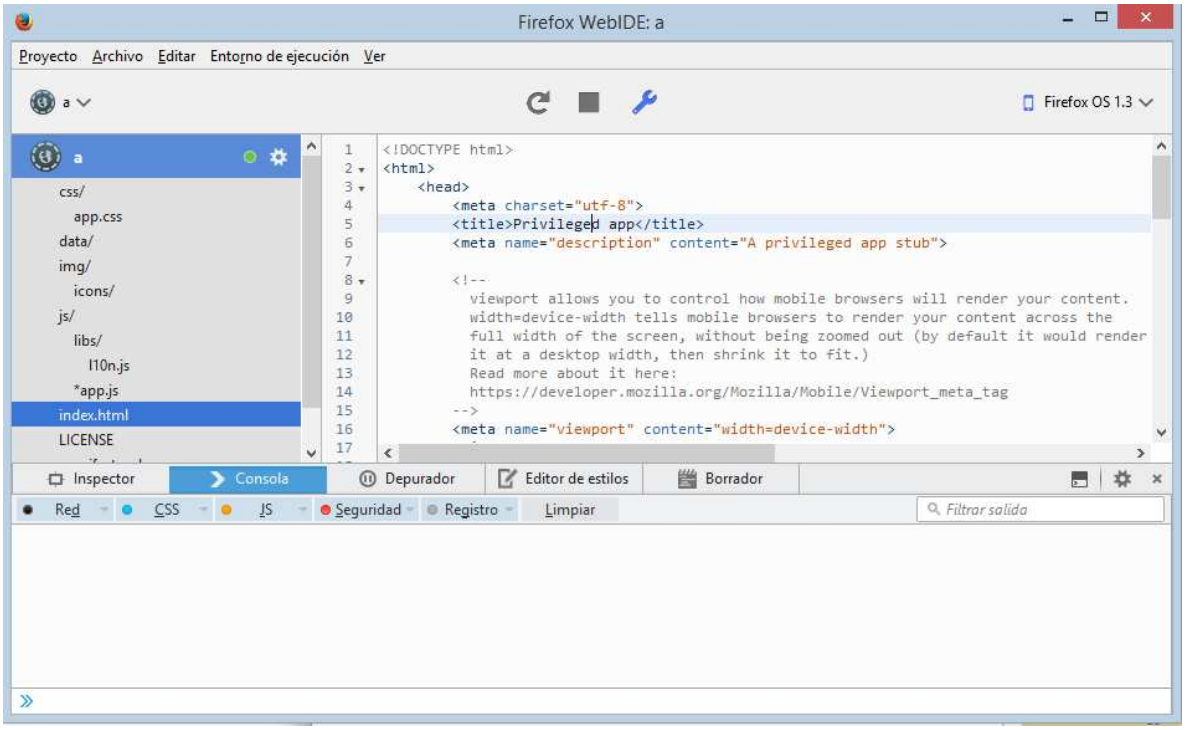


Ilustración 25 - Panel de depuración en WebIDE

Si se conecta un dispositivo real a WebIDE, en el panel de panel de aplicaciones aparecerán todas las que tenga instaladas. Desde este panel se podrán ejecutar, se podrá ver y editar su código desde el editor y depurar en el depurador.

2.3.7. Instalación de aplicaciones en el dispositivo.

Instalar una nueva aplicación en un dispositivo utilizando *WebIDE* es secillo. Primero, habrá que conectar el dispositivo por USB y este será reconocido por *WebIDE,* en la pantalla del smartphone nos aparecera un cuadro de dialogo pidiendo permiso para realizar la conexión. Una vez

aceptado, bastará con ejecutar el proyecto que esté abierto, usando como entorno de ejecución el nombre de dispositivo, que se mostrará en la lista de entornos de ejecución y la aplicación quedará instalada en el dispositivo.

2.4. Directivas de seguridad CSP

Las directivas de seguridad *(Content Security Policy),* son una serie de restricciones que pueden ser usadas por los administradores para establecer el nivel de seguridad de su aplicación, utilizando diferentes combinaciones según sean sus necesidades. Persiguen fundamentalmente evitar ataques externos maliciosos.

Por defecto, las aplicaciones *certified* y *privileged,* tienen una configuración predeterminada de estas directivas. Hay que tener especial cuidado con este aspecto, pues su desconocimiento puede llevar a no comprender por qué aplicaciones que en otros entornos funcionaban correctamente, no lo hagan en Firefox OS.

La configuración de CSP se incluye en el archivo *manifest* y automáticamente se mezcla con la establecida por defecto. Hay que tener en cuenta que en ningún caso se pueden reducir las restricciones establecidas por defecto; sólo se puede añadir una configuración distinta para aumentar las restricciones de uno o varios tipos.

2.4.1. Directivas configurables

base-uri	Especifica las URIs que se pueden usar
child-src	Fuentes válidas en etiquetas como <iframe>
connect-src	Fuentes válidas para conexiones *XMLHttpRequest*, *WebSocket* y *EventSource*
default-src	Define una política de seguridad para contenidos que no son especificados en otras directivas. Abarca las siguientes: *child-src, connect-src, font-src, img-src, media-src, object-src, script-src, style-src*

font-src	Especifica orígenes válidos para fuentes cargadas con *@font-face*
form-action (experimental)	Especifica destinos válidos a los que se accede mediante *<form>*
frame-ancestors (experimental)	Filtra los contenidos que se pueden incrustar usando *<frame>* e *<iframe>*
frame-src	Obsoleta
img-src	Establece fuentes válidas para imágenes e iconos
media-src	Especifíca fuentes válidas para *<audio>* y *<video>*
object-src	Establece fuentes válidas para *<object>, <embed>* y *<applet>*
plugin-types	Establece plugins que pueden ser invocados
referrer	Especifíca información en la cabecera de links externos
reflected-xss *valor*	Permite activar o desactivar heurísticas para filtrar o bloquear ataques *cross-site scripting.* Sus valores pueden ser: *allow* (permitir), *block* (bloquear) y *filter* (filtrar)
report-uri	Envía un infome JSON de intentos de violación de CSP a la URI especificada vía HTTP POST.
sandbox	Aplica restricciones para evitar popups y la ejecución de plugins y scripts
script-src	Especifíca fuentes JavaScript válidas. Cuando están incluidas *script-src* o *default-src* no se permite el uso ni de scripts inline, ni el uso de *eval(),* a menos que se especifique con *'unsafe-inline'* y *'unsafe-eval'*
style-src	Especifica fuentes válidas de estilos. Por defecto, si se usa *style-src* o *default-src*, el uso inline de *<style>* está desactivado, a menos que se active mediante *'unsafe-inline'.*

2.4.2. Valores de configuración

'none': Sin valor específico.

'self': Ámbito restringido a su misma URL.

'unsafe-inline': Permite el uso de recursos en línea.

'unsafe-eval': Permite el uso de *eval()* y métodos similares.

2.4.3. Configuración CSP por defecto

Privileged apps:

default-src *;

script-src 'self';

object-src 'none';

style-src 'self' 'unsafe inline'

Certified apps:

Default-src *;

script-src 'self';

object-src 'none';

style-src 'self'

2.4.4. Ejemplos

Algunos ejemplos de configuración de CSP.

- Ejemplo 1: Usar sólo contenido incluido en el origen de la página

```
1 Content-Security-Policy: default-src 'self'
```

Ilustración 26 - Ejemplo de configuración CSP 1

- Ejemplo 2: Se aceptan contenidos de un dominio y todos sus subdominios.

```
Content-Security-Policy: default-src 'self' *.dominio.com
```

Ilustración 27 - Ejemplo de configuración CSP 2

- Ejemplo 3: Se permite incluir imágenes desde cualquier origen, se restringe el audio y el video a los dominios *media1.com* y *media2.com* (excluidos sus subdominios). Además, sólo se permite ejecutar scripts de *scripts.ejemplo.com*.

```
Content-Security-Policy: default-src 'self';
    img-src *;
    media-src media1.com media2.com;
    scritp-src scripts.ejemplo.com
```

Ilustración 28 - Ejemplo de configuración CSP 3

- Ejemplo 4: Directiva para asegurar que se carga contenido de un dominio en particular usando SSL.

```
Content-Security-Policy: default-src https://dominio.com
```

Ilustración 29 - Ejemplo de configuración CSP 4

- Ejemplo 5: Permite incluir HTML, así como imágenes desde cualquier origen pero no permite JavaScript u otros contenidos potencialmente peligrosos.

```
Content-Security-Policy: default-src 'self' *.mail.com;
    image-src *
```

Ilustración 30 - Ejemplo de configuración CSP 5

2.5. Análisis de las APIs de Firefox OS

Web API es el término usado para referirse al conjunto de APIs compatibles y de acceso a los dispositivos, que permite a las Web apps y su contenido, acceder al hardware del dispositivo (como el estado de la batería o la vibración), y a la información almacenada en el dispositivo (como el calendario o la lista de contactos).

En este apartado se describirá el conjunto de WebAPIs existentes para Firefox OS, algunas de las cuales se usarán en la implementación de este proyecto.

2.5.1. Web APIs de Firefox OS[9]

Para desarrollar APPs en Firefox OS, se utilizan las APIs propias de los dispositivos que usan este sistema, además de estándares web.

Permiso en manifiesto	Nombre	Descripción	Tipo App requerida	Versión que la soporta
alarms	Alarm	Programa una notificación o la ejecución de una aplicación	hosted	FxOS 1.0.1
audio-capture	GetUserMedia	Obtiene un *MediaStream* de audio de un dispositivo de entrada, por ejemplo el micrófono	hosted	FxOS 1.2 Firefox 20+
audio-channel-alarm	Audio Policy	Alarmas del reloj y del calendario	privileged	FxOS 1.0.1
audio-channel-content	Audio Policy	Música, video	hosted	FxOS 1.0.1
audio-channel-normal	Audio Policy	Sonidos de UI, contenido web, música, radio	Hosted	FxOS 1.0.1
audio-channel-notification	Audio Policy	Correo o SMS entrante	Privileged	FxOS 1.0.1
browser	Browser	Permite a la app implementar un browser en un iframe	Privileged	FxOS 1.0.1
camera	Camera	Permite tomar fotos, video, grabar audio y controlar la cámara	Privileged FxOS 2.0+ Certified FxOS 1.0.1-1.4	FxOS 1.0.1

[9] Se puede profundizar en el uso de cada API en https://developer.mozilla.org/en-US/Apps/Reference/Firefox_OS_device_APIs

Permiso en manifiesto	Nombre	Descripción	Tipo App requerida	Versión que la soporta
contacts	Contacts	Añadir, leer o modificar contactos en el dispositivo y en la SIM	Privileged	FxOS 1.0.1 Firefox 18+
desktop-notification	mozNotification para Gecko <22 Notification para Gecko 22+	Muestra una notificación.	Hosted	mozNotification FxOS 1.0.1 Notification FxOS 1.2
device-storage:music	Device Storage	Añade, lee o modifica un fichero de música almacenado en el dispositivo	Privileged	FxOS 1.0.1
device-storage:pictures	Device Storage	Añade, lee o modifica una imagen almacenada en el dispositivo	Privileged	FxOS 1.0.1
device-storage:sdcard	Device Storage	Añade lee o modifica ficheros almacenados en la tarjeta SD	Privileged	FxOS 1.0.1
device-storage:videos	Device Storage	Añade, lee o modifica archivos de video almacenados en el dispositivo	Privileged	FxOS 1.0.1
fmradio	FM Radio	Controla de la radio FM	Hosted	FxOS 1.0.1
geolocation	Geolocation	Obtiene la localización actual de un usuario	Hosted	FxOS 1.0.1
input	Keyboard	Permite a una app interactuar con un teclado virtual	Privileged	FxOS 1.0.1
mobilenetword	Mobile Network	Obtiene información sobre la red móvil (MCC, MNC, etc)	Privileged	FxOS 1.0.1
push	Simple Push	Pone una aplicación en estado de espera para recibir notificaciones	Hosted	FxOS 1.1
storage	Storage	Almacenamiento (appcache, pinned apps, indexedDB)	Hosted	FxOS 1.0.1
systemXHR	SystemXHR	Intercambio de datos con otras URLs	Privileged	FxOs 1.0.1
tcp-socket	TCP Socket	Crea sockets TCP y comunicación a través de ellos	Privileged	FxOs 1.0.1
video-capture	GetUserMedia	Streams de video de dispositivos de entrada de video, por ejemplo la cámara	Hosted	FxOS 1.4

2.5.2. Web APIs generales[10]

- Envío y recepción de datos con XMLHttpRequest

  XMLHttpRequest es un objeto JavaScript usado por *Mozilla*, *Apple* y *Google* y contenido en el estándar *W3C*. Ofrece un mecanismo para recuperar información de una URL sin necesidad de recargar la página.

  No sólo es válido para recuperar datos XML, si no que admite otros formatos además de HTTP, como *file* y *ftp*.

- Eventos *del DOM*

  Eventos que pueden ser enviados y recibidos para interactuar con los objetos del DOM.

- *Eventos de toque*

  Eventos táctiles para utilizar en aplicaciones utilizadas en dispositivos táctiles.

- Eventos *online y offline*

  Eventos que permiten a las aplicaciones identificar cambios de estado en la red.

- Historial

  Permite la manipulación y el movimiento a través del historial, atrás, adelante, ir a un punto específico del historial, etc.

- Audio

  Permite la integración y manipulación de contenido de audio usando la etiqueta *<audio>* de *HTML5*.

[10] Documentación completa en https://developer.mozilla.org/es/Apps/Reference

- Video

Permite la integración y manipulación de contenido de video usando la etiqueta *<video>* de *HTML5.*

- Geolocalización

Permite al dispositivo realizar la petición de si se desea usar la geolocalización. Una vez activada realiza la ubicación del usuario y el seguimiento de sus cambios.

- Orientación *de la pantalla*

Detecta los cambios de la posición de la pantalla entre vertical y horizontal.

- Orientación *del dispositivo*

Permite a las apps detectar cambios de orientación mediante los sensores del dispositivo.

- IndexedDB

Es una base de datos web capaz de albergar gran cantidad de información en el dispositivo, ofrece una interfaz para su manejo.

- Almacenamiento

La API de almacenamiento provee de varias formas de almacenar cantidades liminatadas de información en el dispositivo, *localStorage, sessionStorage, Blob*.

- *Web* Workers

Permite a las aplicaciones ejecutar código JavaScript, que no puede interactuar con el DOM o con el objeto Window, en segundo plano. Estas tareas permiten optimizar el rendimiento y desbloquear el renderizado de la interfaz de usuario.

2.5.3. Comunidad WebAPI

Se puede obtener más ayuda con estas APIs, a través de los foros existentes, contactando con otros desarrolladores que las usan, por alguno de estos medios

- Lista de correo
- Grupo de noticias
- Grupo de Google
- RSS

Canal de IRC de WebAPI: #webapi

2.6. APIs usadas en este proyecto

En este apartado se hará un estudio más profundo de las APIs que se ha utilizado para realizar el proyecto que se documentará en el punto 3.

2.6.1. API de Geolocalización

Permite a una aplicación conocer la ubicación del dispositrivo. Se pide al usuario que confirme es uso de la geolocalización por razones de privacidad.

Se accede al API de gelocalización a través del objeto ***navigator.geolocation****, s*i existe, el servicio está disponible, lo que nos permite comprobar si es soportado por el dispositio.

```
1  if ("geolocation" in navigator) {
2    /* geolocalización disponible */
3  } else {
4    /* geolocalización no disponible */
5  }
```

Ilustración 31 - Disponibilidad de geolocalización

Con el método ***getCurrentPosition()***, se obtiene la ubicación actual. Cuando se realiza la llamada, el método consulta el hardware para obtener la información requerida e intenta responder lo más rápidamente posible con un resultado de baja precisión basado en IP o WIFI. Si se necesita una respuesta

de alta precisión, esta puede ser más lenta, pues el hardware de posicionamiento puede tardar minutos en obtener una posición. Una vez obtenida una posición esta se representa con un objeto ***Position*** referenciando a un objeto ***Coordinates***.

```
navigator.geolocation.getCurrentPosition(function(position) {
  do_something(position.coords.latitude, position.coords.longitude);
});
```

Ilustración 32 - Ejemplo de getCurrentPosition

En este ejemplo, la función *do_something* se ejecutará una vez obtenida la posición.

Si el dispositivo se encuentra en movimiento, los datos de ubicación cambian. Para ratrear los datos, se usa la función ***watchPosition()***, que recibe los mismos parámetros que *getCurrentPosition(),* y devuelve un identificador para el rastreador de posición solicitado. Pasando este identificador como parámetro a la función ***clearPosition()***, se deja de rastrear la posición.

```
var watchID = navigator.geolocation.watchPosition(function(position) {
  do_something(position.coords.latitude, position.coords.longitude);
});

navigator.geolocation.clearWatch(watchID);
```

Ilustración 33 - Uso de watchPosition() y clearWatch()

Tanto ***getCurrentPosition()*** como ***watchPosition()***, aceptan una función callback opcional por si ocurre algún error y un objeto ***PositionOptions*** para especificar el nivel de precisión, la antigüedad de la petición o el tiempo de espera.

```
function geo_success(position) {
  do_something(position.coords.latitude, position.coords.longitude);
}

function geo_error() {//función si se produce un error
  alert("Sorry, no position available.");
}

var geo_options = {//objeto PositionOptions
  enableHighAccuracy: true,
  maximumAge        : 30000,
  timeout           : 27000
};

var wpid = navigator.geolocation.watchPosition(geo_success, geo_error, geo_options);
```

Ilustración 34 - - watchOptions() con PositionOptions

Con todo lo visto anteriormente, se puede construir el siguiente ejemplo:

Código HTML

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, user-scalable=no" />
  <title>Geo</title>
</head>

<body>
    <div id="mapa"></div>

    <script src="geo.js"></script>
</body>

```

Código JS

```javascript
function geoLocalizaMe() {
  var output = document.getElementById("mapa");

  var geo_options = {
  enableHighAccuracy: true,
  maximumAge        : 30000,
  timeout           : 27000
  };

  if (!navigator.geolocation){
    output.innerHTML = "<p>Su dispositivo no soporta geolocalización</p>";
    return;
  }

  function success(position) {
    var latitude  = position.coords.latitude;
    var longitude = position.coords.longitude;

    output.innerHTML = "<p>La latitud es " +
    + latitude + "<br>la longitud es " + longitude + "</p>";

    var img = new Image();
    img.src = "http://maps.googleapis.com/maps/api/staticmap?center=" +
    + latitude + "," + longitude + "&zoom=13&size=300x300&sensor=false";

    output.appendChild(img);
  };

  function error() {
    output.innerHTML = "Imposible obtener su ubicación";
  };

  output.innerHTML = "<p>Geolocalizando…</p>";

  navigator.geolocation.getCurrentPosition(success, error, geo_options);
}

geoLocalizaMe();

```

Ilustración 35 - Ejemplo de geolocalización

Que produciría el siguiente resultado en el navegador:

Ilustración 36 - Resultado de geolocalización

2.6.2. IndexedDB

IndedexedDB es un API de cliente para el almacenamiento de datos de forma estructurada y que permite realizar búsquedas mediante índices. Provee APIs para aceeso síncrono o asíncrono, el acceso síncrono está indicado para ser usado dentro de *Web Workers* (procesos en segundo plano que no interactúan con el DOM), mientras que el asíncrono puede trabajar en cualquier tipo de aplicaciones. A continuación se detallan sus principales características:

- Esta base de datos almacena pares clave-valor, los valores pueden ser objetos y las claves pueden ser propiedades de esos objetos. Se pueden crear índices que usen cualquier propiedad de un objeto o una enumeración ordenada.
- Se basa en un modelo transaccional, por lo que las acciones siempre se incluyen en el contexto de una transacción, que tiene un

ciclo de vida bien definido. Este modelo es útil para controlar escenarios en los que se abran varias instancias de la aplicación, puesto que sin operaciones transaccionales, las modificaciones de una instancia podrían sobreescribir modificaciones de otras.

- Es principalmente asíncrona. No devuelve valores, devuelve una notificación a través de un evento DOM cuando una operación ha sido realizada.
- Usa eventos DOM para nortificar que los resultados están disponibles. Los eventos DOM tienen unas propiedades asociadas, en el caso de *IndexedDB,* suelen ser *"success"* y "*error"*, en los que se realizan las acciones convenientes en cada caso.
- Está orientada a objetos. No es una base de datos relacional, con tablas de filas y columnas, sino que se crean almacenes de objetos de un tipo de datos. Cada almacén de objetos puede tener un conjunto de índices para realizar las consultas e iterar sobre sus datos. Esta cuestión se debe tener en cuenta a la hora de diseñar y construir las aplicaciones.
- No usa SQL. Se realizan consultas en un índice, a las que se aplica un cursor, que es un iterador que va recorriendo el conjunto de resultados..
- Vincula las bases de datos al origen. El origen es el dominio, el protocolo de la capa de aplicación y el puerto de la URL de la página donde se ejecuta el script. Cada origen tiene su conjunto de bases de datos asociado y cada base de datos tiene un nombre que la indentifica dentro de un origen.

Veamos a continuación las principales operaciones que necesitaremos para trabajar con esta base de datos.

2.6.2.1. Abrir la base de datos

Para abrir la base de datos se usará el método *open*, pasándole como parámetros el nombre de la base de datos y la versión de la base de datos. Si la base de datos no existe es creada automáticamente.

Cuando se llama a esta función se crea un objeto *IDBOpenDBRequest* con tres posible resultados que pueden ser tratados como eventos:

- *success:* acciones a realizar en caso de éxito
- *error:* acciones a realizar en caso de error
- *onupgradeneeded:* actualiza la versión de la base de datos en caso de que sea necesario

```
const nombreDB = "Nombre de la base de datos";
var db;

if (window.indexedDB) {//Testea si el navegador soporta IndexedDB
    console.log("IndexedDB soportado");
}
else {
    alert("Indexed DB no soportado");
}

// Abriendo la base de datos
//1º parametro : Nombre, 2º parametro version
request = indexedDB.open(nombreDB, 1);//Parámetros nombre y versión

request.onsuccess = function (e) {
    db = e.target.result;
    console.log("Base de datos inicializada");
};

request.onerror = function (e) {
    console.log("Se ha producido un error: " + e.value);
};

request.onupgradeneeded = function (e) {//Actualización de versión
    db = e.target.result;

    //Borrado
    if (db.objectStoreNames.contains("ejercicios")) {
        db.deleteObjectStore("ejercicios");
    }

    //Creación, clave id con autoincremento
    var objectStore = db.createObjectStore('ejercicios',
    { keyPath: 'id', autoIncrement: true });

    console.log("Base de datos actualizada");
};

```

Ilustración 37 – Ejemplo de apertura de base de datos en IndexedDB

2.6.2.2. Añadir registros

Una vez creada la base de datos se pueden añadir registros usando el método *add,* como vemos en el siguiente ejemplo:

```
1  //Creación de la transacción
2  var transaccion = this.db.transaction([ 'ejercicios' ], 'readwrite'),
3      store = transaccion.objectStore('ejercicios'),
4      request = store.add(ejercicioActual);//Elemento a guardar en la BD
5
6  //Callbacks para éxito y error
7  request.onsuccess = function (e) {
8      console.log("El ejercicio ha sido guardado");
9  };
10
11 request.onerror = function (e) {
12     console.log("Error guardando el ejercicio. Razón : " + e.value);
13 };
```

Ilustración 38 - Añadir un registro

2.6.2.3. Recuperar registros

El proceso para recuperar un registo de la base de datos sigue una estructura similar al anterior:

```
1  //Creación de la transacción
2  var transaccion = this.db.transaction([ 'ejercicios' ], 'readwrite'),
3      store = transaccion.objectStore('ejercicios'),
4      request = store.get(id);//Elemento a recuperar usando el id
5
6  //Callbacks para éxito y error
7  request.onsuccess = function (e) {
8      //Acciones a realizar con el objeto recuperado
9      console.log("El nombre del ejercicio recuperado es: " + request.result.nombre);
10 };
11
12 request.onerror = function (e) {
13     console.log("Error recuperando el ejercicio. Razón : " + e.value);
14 };
```

Ilustración 39 - Recuperar un registro

2.6.2.4. Eliminar registros

Para eliminar un registro se usará el siguiente código. Si lo que queremos es eliminar todos los registros de la base de datos bastará sustituir el método *delete()* por *clear()*.

```
//Creación de la transacción
var transaccion = this.db.transaction([ 'ejercicios' ], 'readwrite'),
    store = transaccion.objectStore('ejercicios'),
    request = store.delete(id);//Elemento a eliminar usando el id

//Callbacks para éxito y error
request.onsuccess = function (e) {
    console.log("Ejercicio eliminado");
};

request.onerror = function (e) {
    console.log("Error eliminando el ejercicio. Razón : " + e.value);
};
```

Ilustración 40 - Eliminar un registro

2.6.2.5. Editar registros

Para editar un registro primero lo recuperaremos como se ha descrito en el apartado anterior. Una vez que tenemos acceso a el procederemos a editarlo y finalmente se actualizará en la base de datos.

```
//Creación de la transacción
var transaccion = this.db.transaction([ 'ejercicios' ], 'readwrite'),
    store = transaccion.objectStore('ejercicios'),
    request = store.get(id);//Elemento a recuperar usando el id

//Callbacks para éxito y error
request.onsuccess = function (e) {
    //La variable ejercicio contiene el registro recuperado
    var ejercicio = request.result;

    //Edición
    ejercicio.nombre = "Jaén-Torredelcampo-Jaén";

    //Se actualiza en la base de datos
    var requestUpdate = store.put(ejercicio);

    //Callbacks para éxito y error
    request.onsuccess = function (e) {
        console.log("Ejercicio actualizado");
    };

    request.onerror = function (e) {
        console.log("Error actualizando el ejercicio. Razón : " + e.value);
    };
};

request.onerror = function (e) {
    console.log("Error recuperando el ejercicio. Razón : " + e.value);
};
```

Ilustración 41 - Editar un registro

2.6.2.6. Uso de cursores

Un cursor no es más que un iterador que nos permite recorrer secuencialmente los objetos almacenados en la base de datos. Su uso se ejemplifica en la siguiente ilustración:

```
1  //Creación de la transacción
2  var transaccion = this.db.transaction([ 'ejercicios' ], 'readwrite'),
3      store = transaccion.objectStore('ejercicios');
4
5      store.openCursor().onsuccess = function (e) {
6          var cursor = e.target.result;
7
8          if(cursor){
9              //Acciones a realizar con los elementos
10             console.log("Nombre: " + cursor.value.nombre);
11             cursor.continue();
12         }else{
13             console.log("No existen más entradas");
14         }
15     };
16
```

Ilustración 42 - Iterar sobre los elementos de la base de datos

2.7. LIBRERIAS

2.7.1. Introducción. Librerías y Frameworks.

Existen gran cantidad de librerías dedicadas a ayudarnos en el desarrollo de una Webapp. Ya que el concepto de desarrollo para la web es muy amplio se pueden clasificar atendiendo a una serie de criterios.

- **Tipo de proyecto**: Bibliotecas que ayuden a implementar aplicaciones web con una lógica de negocio compleja. Se descartan las webs típicas de comunicación, que normalmente requieren otro tipo de destreza.
- **Problema a resolver.** Se pretenden encontrar bibliotecas JS orientadas a solucionar problemas de programación, más que diseño y maquetación.

Partiendo de estos dos criterios, los puntos principales a valorar de cara a clasificar estos proyectos de JavaScript son los siguientes:

- **Arquitectura de la aplicación:** *Frameworks* completos que sirven de esqueleto de aplicaciones web y dotan de estructura a las páginas y a sus distintas partes.
- **Manipulación del DOM, AJAX y utilidades varias:** bibliotecas de funciones misceláneas que amplían o corrigen la API JavaScript básica del navegador y proporcionan abstracciones útiles o notación más concisa para necesidades comunes de aplicaciones web modernas.
- **Componentes visuales y gráficas:** bibliotecas de *widgets* y controles de UI (interfaz de usuario).

Otras características secundarias a tener en cuenta para evaluar las bibliotecas son: precio, tamaño en bytes (fundamental para aplicaciones diseñadas para dispositivos con recursos limitados), licenciamiento, grado de soporte a navegadores móviles, documentación, popularidad, sociabilidad con otras herramientas, curva de aprendizaje, aspectos destacados y aspectos que se echan de menos.

2.7.1.1. Características de los Frameworks

En la siguiente tabla se pueden observar las características de los Frameworks más usados:

	Arquitectura	DOM, AJAX, utils	Componentes visuales
Angular JS	✓	✓	
Backbone.js	✓		
Ember	✓		
Knockout	✓		
RequireJS		✓	
JQuery		✓	
Underscore		✓	
Bootstrap			✓
JQuery UI			✓
JQuery Mobile			✓
Kendo UI	✓	✓	✓

YUI	✓	✓	✓
Prototype		✓	
Sencha	✓	✓	✓
Zepto		✓	
Script.aculo.us		✓	
AngularUI			✓
Dojo	✓	✓	✓
Closure Library		✓	✓
Google Charts			✓
MailChimp's Pattern Library			✓
Building Blocks			✓

Teniendo en cuenta estas características para el proyecto de Webapp que se documentará en el siguiente capítulo se usarán las librerías *Zepto* para la interacción con el DOM por su ligereza y *Building Blocks* para el diseño, ya que contiene estilos similares a los usados por *Firefox OS*:

2.7.2. Librería Zepto

Zepto es una librería ligera, para navegadores modernos y con gran compatibilidad con las APIs de *jQuery*. Los usuarios que usan *jQuery* no tendrán problema para usarla.

Las APIs proporcionadas por *Zepto* coinciden con sus homólogas de *jQuery*. El objetivo es tener una biblioteca modular, de entre 5 y 10K, que se descarga y ejecuta muy rápidamente, con una API familiar y versátil. *Zepto* es un software de código abierto bajo licencia *MIT*.

La compilación por defecto incluye los siguientes módulos *Core*, *Ajax*, *Event*, *Form* e *IE.*

Para incluir la librería *Zepto* en nuestra página añadiremos la siguiente línea cerca del final del fichero HTML:

```
1  ...
2  <script src=zepto.min.js></script>
3  </body>
4  </html>
```

Ilustración 43 - Incluyendo la librería Zepto

Es soportada al 100% por los siguientes navegadores:

- Safari 6+ (Mac)
- Chrome 30+ (Windows, Mac, Android, iOS, Linux, Chrome OS)
- Firefox 24+ (Windows, Mac, Android, Linux, Firefox OS)
- iOS 5+ Safari
- Android 2.3 Browser
- Internet Explorer 10+ (Windows, Windows Phone)

Zepto permite personalización para maximizar su eficiencia mediante el sistema de construcción incluido en su código fuente. Este permite seleccionar módulos y realizar pruebas de ejecución. Con *Uglify.JS* se puede obtener una estimación de la compresión que resultará en *zepto.min.js*

2.7.3. Librería GMaps[11]

GMaps.js es una librería JavaScript, que permite usar de una forma sencilla los mapas de Google. Incluye una serie de funciones que permiten visualizar mapas e interactuar con ellos, añadir marcadores, seleccionar áreas del mapa, dibujar rutas, etc.

[11] Página oficial: https://hpneo.github.io/gmaps/

Veamos algunos ejemplos:

- *Visualizar un mapa centrado en unas coordenadas*

Ilustración 44 - Ejemplo de mapa centrado en coordenadas

Se puede observar en el código cómo se crea el mapa usando la función *GMaps*, a la que se le indica en qué parte de la página se va a colocar, ***div: '#map',*** la latitud, ***lat: 37.788796*** y la longitud ***lng: -3.777908***. Se podría definir también el zoom del mapa, con ***zoom: valor***, por defecto 15, el ancho ***width: valor*** y el alto ***height: valor***, ambos en píxeles.

- *Geolocalizar*

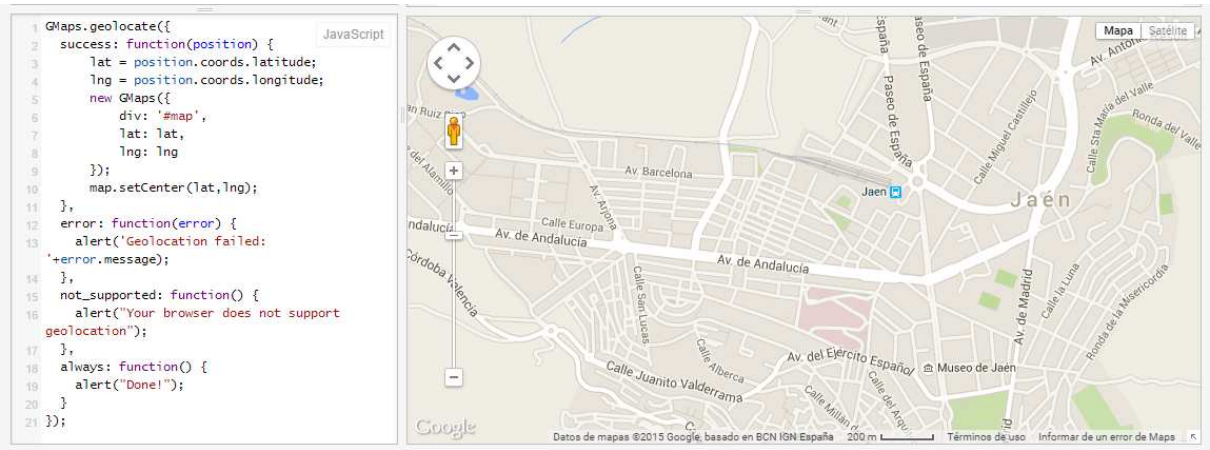


Ilustración 45 - Ejemplo de mapa geolocalizado

Mediante *GMaps.geolocate* se obtiene la posición actual y se usan la latitud ***position.coords.latitude*** y la longitud ***position.coords.longitude*** para dibujar el mapa con el método anterior.

Con el método ***setCenter(latitud, longitud)*** se centra el mapa en unas coordenadas determinadas, en este caso son las mismas de la geolocalización. En caso de la geolocalización falle, o no esté soportada se terminará con los eventos ***error*** o ***not_suported***.

- *Dibujar líneas entre puntos*

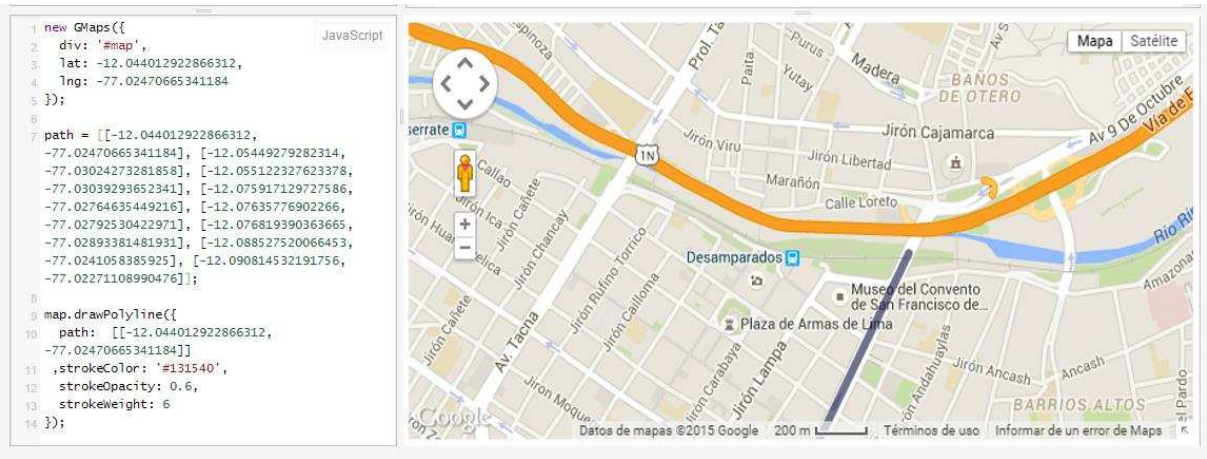


Ilustración 46 - Ejemplo de mapa con línea entre puntos

Con esta función se pueden dibujar líneas entre un conjunto de puntos. Los puntos se pasan en la variable ***path***, y ***map.drawPolyline*** va uniendo cada punto con el siguiente. Esta función une cada pareja de puntos en línea recta, sin tener en cuenta los edificios representados en el mapa.

- *Dibujar rutas*

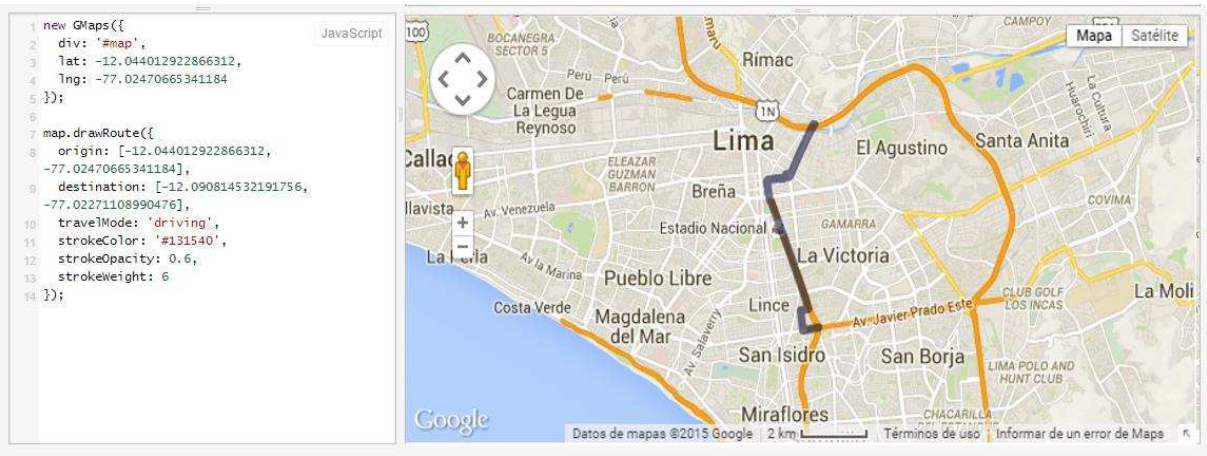


Ilustración 47 - Ejemplo de mapa con ruta

map.drawRoute une dos puntos en el mapa, especificados en ***origin*** y ***destination***. Al contrario que la función anterior, esta no los une en línea recta, si no que calcula la ruta sorteando los obstáculos físicos que encuentra a su paso. En ***travelMode***, se indica el modo en que realizamos la travesía, pudiendo pasar como valores, ***driving***, ***bicycling*** o ***walking***, con lo que además tendrá en cuenta, si procede, los sentidos de circulación de las calles por las pasa la ruta.

- *Mapas estáticos*

```javascript
url = GMaps.staticMapURL({
  size: [1400,400],
  lat: -12.043333,
  lng: -77.028333
});

$('<img/>').attr('src', url)
  .appendTo('#map');
```

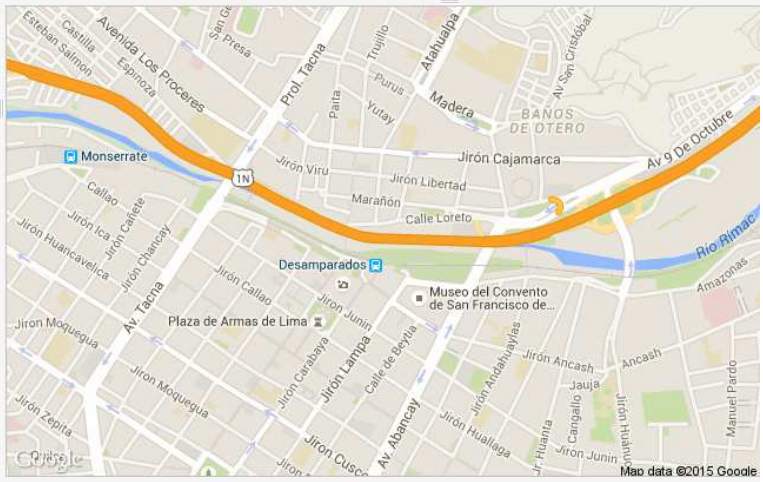


Ilustración 48 - Ejemplo de mapa estático

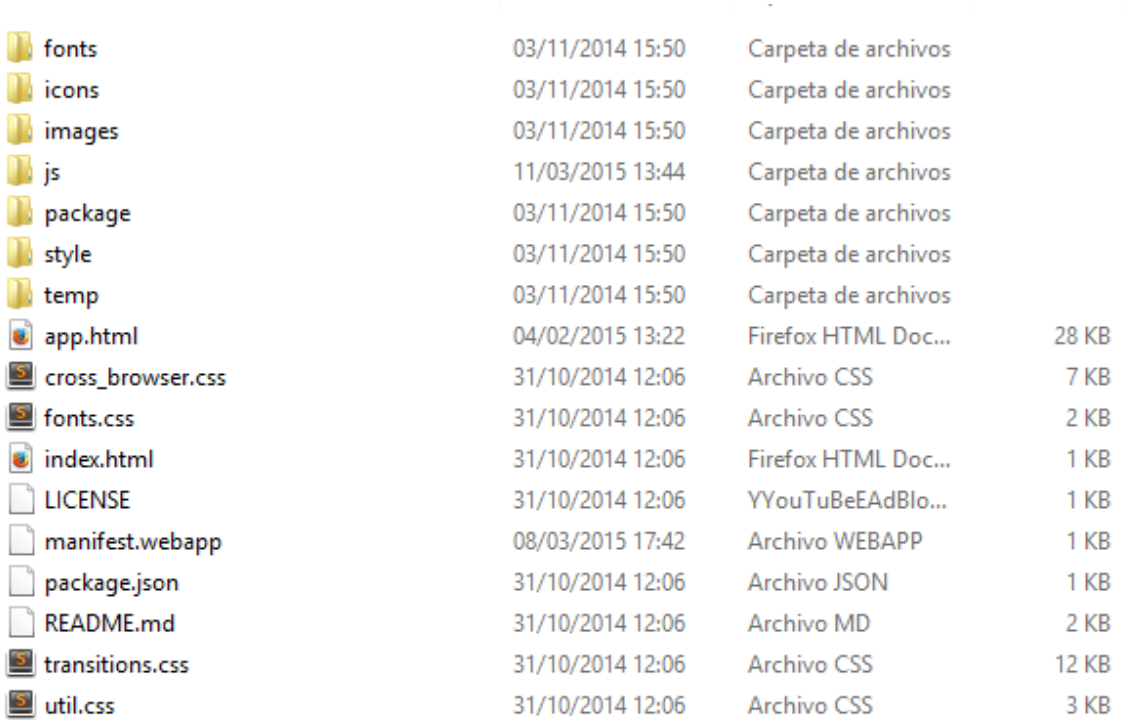
Esta función muestra un mapa centrado en ***lat*** y ***lng***, pero a diferencia del primer caso con ***GMaps***, este es estático, es decir, una fotografia, no hay ninguna posibilidad de interactuar con él.

2.7.4. Building Blocks[12]

Building Blocks es una colección de componentes de interfaz de usuario, reusables, que pueden ser usados de manera sencilla para construir una app de Firefox OS o una aplicación web convencional. Todos los componentes incluidos se ajustan a la guía de estilos de Firefox OS, y proporcionan el código HTML, JavaScript y CSS necesario para su implementación.

[12] Descripción de componentes en http://buildingfirefoxos.com/building-blocks/action-menu.html

Para incluir componenentes en una app bastará con copiar los ficheros JavaScript y CSS necesarios y copiar el código HTML del componente en la aplicación.



fonts	03/11/2014 15:50	Carpeta de archivos	
icons	03/11/2014 15:50	Carpeta de archivos	
images	03/11/2014 15:50	Carpeta de archivos	
js	11/03/2015 13:44	Carpeta de archivos	
package	03/11/2014 15:50	Carpeta de archivos	
style	03/11/2014 15:50	Carpeta de archivos	
temp	03/11/2014 15:50	Carpeta de archivos	
app.html	04/02/2015 13:22	Firefox HTML Doc...	28 KB
cross_browser.css	31/10/2014 12:06	Archivo CSS	7 KB
fonts.css	31/10/2014 12:06	Archivo CSS	2 KB
index.html	31/10/2014 12:06	Firefox HTML Doc...	1 KB
LICENSE	31/10/2014 12:06	YYouTuBeEAdBlo...	1 KB
manifest.webapp	08/03/2015 17:42	Archivo WEBAPP	1 KB
package.json	31/10/2014 12:06	Archivo JSON	1 KB
README.md	31/10/2014 12:06	Archivo MD	2 KB
transitions.css	31/10/2014 12:06	Archivo CSS	12 KB
util.css	31/10/2014 12:06	Archivo CSS	3 KB

Ilustración 49 - Árbol de ficheros y directorios de la app de ejemplo de Building Blocks

En la ilustración anterior se puede observar el árbol de ficheros y directorios de la aplicación de ejemplo de *Building Blocks* que podemos descargar o instalar desde http://buildingfirefoxos.com/downloads/, aquí se incluye todo el código necesario para implementar cualquier complonente.

2.7.4.1. Componentes

- *Menú de acciones: muestra una lista de acciones, el usuario puede seleccionar una.*

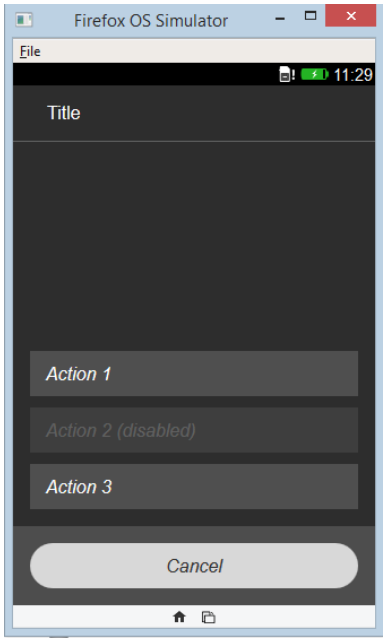


Ilustración 50 – Menú de acciones

Código CSS

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

Código HTML

```
<form role="dialog" data-type="action">
  <header>Title</header>
  <menu>
    <button>Action 1</button>
    <button disabled>Action 2
(disabled)</button>
    <button>Action 3</button>
    <button>Cancel</button>
  </menu>
</form>
```

- *Botones: distintos estilos de botones en función de uso que se les vaya a dar.*

Código CSS

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

Código HTML

```
<button>Default</button>
<a class="bb-button recommend"
href="#">Recommend</a>
<button class="danger">Danger</button>
```

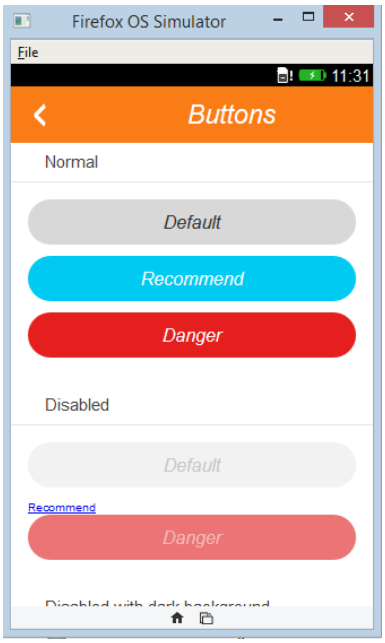


Ilustración 51 - Botones

- *Confirmación: ventana de confirmación o información de evento.*

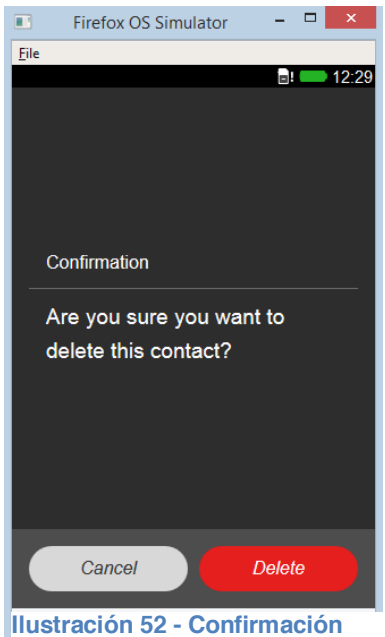


Ilustración 52 - Confirmación

Código CSS

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

Código HTML

```
<form role="dialog" data-type="confirm">
  <section>
    <h1>Confirmation</h1><!-- opcional -->
    <p>Are you sure you want to delete this contact?</p>
  </section>
  <menu>
    <button>Cancel</button>
    <button class="danger">Delete</button>
  </menu>
</form>
```

- *Menú lateral: menú emergente del lateral con opciones elegibles por el usuario.*

Código CSS

```
<link href="…/headers.css" rel="stylesheet" type="text/css">
<link href="…/drawer.css" rel="stylesheet" type="text/css">
```

Código HTML

```
<section data-type="sidebar">
  <header>
    <menu type="toolbar">
      <a href="#content">Done</a>
    </menu>
    <h1>Drawer demo</h1>
  </header>
  <nav>
    <ul>
      <li><a href="#content">item 1</a></li>
      <li><a href="#content">item 2</a></li>
      <li><a href="#content">item 3</a></li>
    </ul>
    <h2>This is a subtitle</h2>
    <ul>
      <li><a href="#content">item 4</a></li>
      <li><a href="#content">item 5</a></li>
      <li><a href="#content">item 6</a></li>
    </ul>
  </nav>
</section>
<section id="drawer" role="region">
  <header>
    <a href="#content"><span class="icon icon-menu">hide sidebar</span></a>
    <a href="#drawer"><span class="icon icon-menu">show sidebar</span></a>
    <h1>Building Blocks <em>demo</em></h1>
  </header>
  <div role="main"></div>
</section>
```

Ilustración 53 - Menú lateral

- *Modo edición: ventana para contenidos editables, por ejemplo, borrados.*

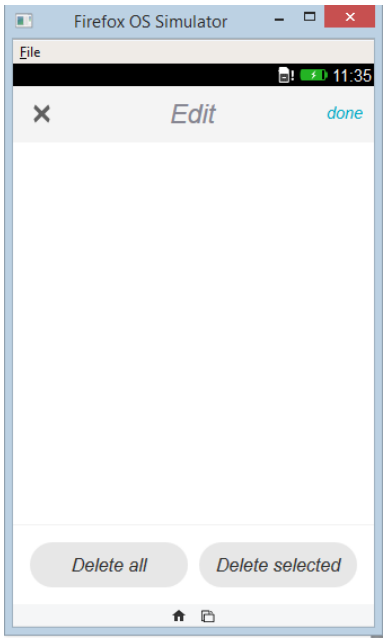


Ilustración 55 - Modo edición

Código CSS

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

Código HTML

```
<form role="dialog" data-type="edit">
  <section role="region">
    <header>
      <button><span class="icon icon-
close">close</span></button>
      <menu type="toolbar">
        <button>done</button>
      </menu>
      <h1>Edit</h1>
    </header>
  </section>
  <menu>
    <button>Delete all</button>
    <button>Delete selected</button>
  </menu>
</form>
```

- *Cabeceras: distinto tipos de cabeceras.*

Código CSS

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

Código HTML

```
<section role="region">
  <header>
    <menu type="toolbar">
      <a href="#"><span class="icon icon-
edit">edit</span></a>
      <a href="#"><span class="icon icon-
add">add</span></a>
    </menu>
    <h1>Messages</h1>
  </header>
</section>
<section role="region">
  <header>
    <button><span class="icon icon-
menu">menu</span></button>
    <menu type="toolbar">
      <button><span class="icon icon-
add">add</span></button>
    </menu>
    <h1>Inbox <em>(2)</em></h1>
  </header>
</section>
<section role="region">
  <header>
    <button><span class="icon icon-
close">close</span></button>
    <menu type="toolbar"><button>done</button></menu>
    <h1>Title</h1>
  </header>
  <header>
    <h2>Subheader text</h2>
  </header>
</section>
```

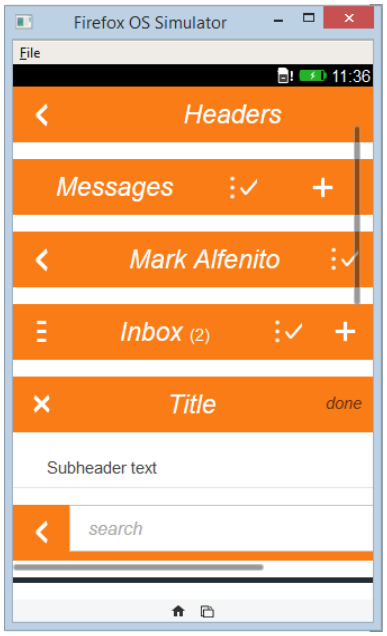


Ilustración 54 - Cabeceras

- *Entrada de datos: ventanas con campos de entrada de datos.*

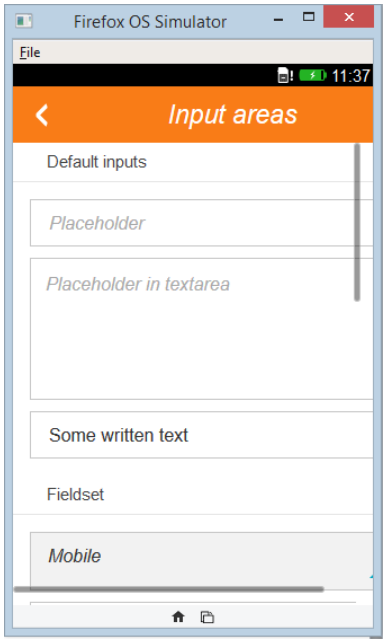


Ilustración 56 - Entrada de datos

Código CSS

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

Código HTML

```
<form>
  <p>
    <input type="text"
placeholder="Placeholder" required="">
    <button type="reset">Clear</button>
  </p>
  <p>
    <textarea placeholder="Placeholder in
textarea" required=""></textarea>
  </p>
  <p>
    <input type="text"
placeholder="Placeholder" value="Some
written text" required="">
    <button type="reset">Clear</button>
  </p>
</form>
```

- *Listas: para mostrar conjuntos de elementos de forma consecutiva.*

Código CSS

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

Código HTML

```
<section data-type="list">
  <header>Title</header>
  <ul>
    <li>
      <p>Travis Gray</p>
    </li>
    <li>
      <a href="#">
        <p>Travis Gray</p>
        <p>Clickable item</p>
      </a>
    </li>
    <li>
      <aside class="pack-end">
        <img alt="placeholder"
src="myimage.jpg">
      </aside>
      <a href="#">
        <p>Travis Gray</p>
        <p>Beginning of message</p>
      </a>
    </li>
  </ul>
  <header>Another title</header>
  <ul>
    <li aria-disabled="true">
      <aside class="pack-end">
        <img alt="placeholder"
src="myimage.jpg">
      </aside>
      <a href="#">
        <p>Travis Gray</p>
        <p>Disabled item</p>
      </a>
    </li>
    <li>
      <a href="#">
        <aside data-icon="call-outgoing">
          asidecall
        </aside>
        <p>Travis Gray <em>(2)</em></p>
        <p>
          <time
datetime="17:43">5:43PM</time>
          Mobile
        </p>
      </a>
    </li>
  </ul>
</section>
```

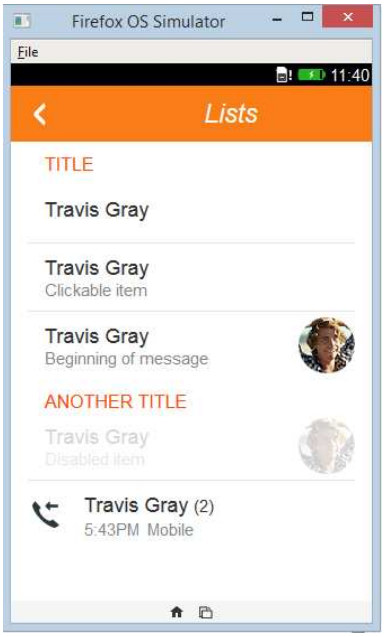


Ilustración 57 - Listas

- *Progreso y actividad: efectos visuales para mostrar el progreso o la ejecución de un proceso.*

Código CSS

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

Código HTML

```
<section role="region" id="progress" data-
position="right">
    <header class="fixed">
      <a id="btn-progress-back" href="#"><span
class="icon icon-back">back</span></a>
      <h1>Progress and activity</h1>
    </header>
    <article class="content scrollable header">
      <header><h2>Spinner</h2></header>
      <div class="center">
        <progress></progress>
      </div>

      <header><h2>Activity bar</h2></header>
      <div>
        <progress class="pack-activity"
value="0" max="100"></progress>
      </div>

      <header><h2>Progress bar</h2></header>
      <div>
        <progress value="80"
max="100"></progress>
      </div>

      <header><h2>Progress + activity
bar</h2></header>
      <div>
        <progress class="pack-activity"
value="80" max="100"></progress>
      </div>

      <header><h2>Light progress + activity
bar</h2></header>
      <div>
        <progress class="pack-activity light"
value="0" max="100"></progress>
      </div>
    </article>
  </section>
```

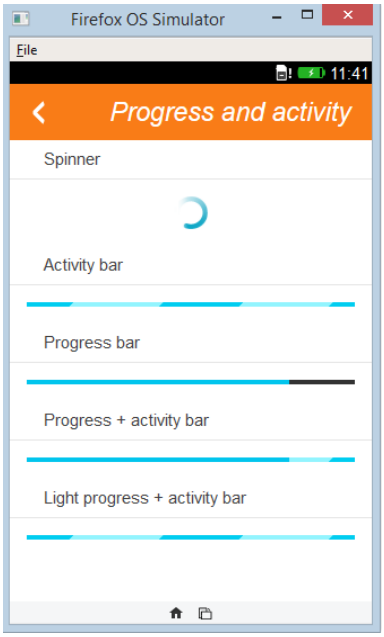


Ilustración 58 - Progreso y actividad

- *Scrolling: lista de texto, imágenes y/o video con scroll vertical*

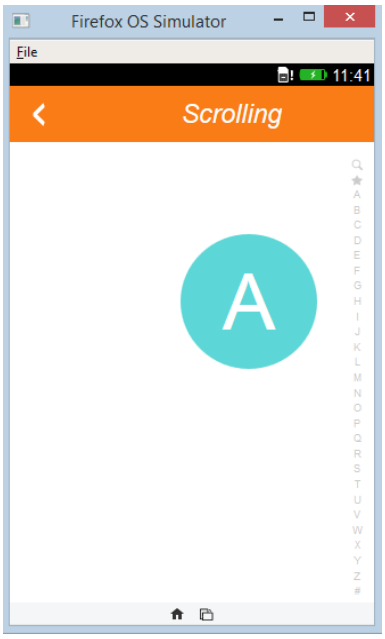


Ilustración 59 - Scrolling

Código CSS

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

Código HTML

```
<section role="region" id="main">
  <nav data-type="scrollbar">
    <p>A</p>
    <ol>
      <li><a href="#"><span class="pack-
icon-search">Search</span></a></li>
      <li><a href="#"><span class="pack-
icon-favorites">favorites</span></a></li>
      <li><a href="#">A</a></li>
      <li><a href="#">B</a></li>
      <li><a href="#">C</a></li>
      <li><a href="#">D</a></li>
      <li><a href="#">E</a></li>
      <li><a href="#">F</a></li>
      <li><a href="#">G</a></li>
      <li><a href="#">H</a></li>
      <li><a href="#">I</a></li>
      <li><a href="#">J</a></li>
      <li><a href="#">K</a></li>
      <li><a href="#">L</a></li>
      <li><a href="#">M</a></li>
      <li><a href="#">N</a></li>
      <li><a href="#">O</a></li>
      <li><a href="#">P</a></li>
      <li><a href="#">Q</a></li>
      <li><a href="#">R</a></li>
      <li><a href="#">S</a></li>
      <li><a href="#">T</a></li>
      <li><a href="#">U</a></li>
      <li><a href="#">V</a></li>
      <li><a href="#">W</a></li>
      <li><a href="#">X</a></li>
      <li><a href="#">Y</a></li>
      <li><a href="#">Z</a></li>
      <li><a href="#">#</a></li>
    </ol>
  </nav>
</section>
```

- *Barras de búsqueda: barras de búsqueda y ajuste*

Código CSS

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

Código HTML

```
<section role="region">
  <div role="slider" aria-valuemin="0" aria-
valuenow="80" aria-valuemax="100" aria-
valuetext="slider description">
    <div>
      <progress value="80"
max="100"></progress>
      <button>handler</button>
    </div>
  </div>
</section>
<section role="region">
  <div role="slider" aria-valuemin="0" aria-
valuenow="80" aria-valuemax="100" aria-
valuetext="slider description">
    <label>0</label>
    <label>100</label>
    <div>
      <progress value="80"
max="100"></progress>
      <button>handler</button>
    </div>
  </div>
</section>
<section role="region">
  <div role="slider" aria-valuemin="0" aria-
valuenow="80" aria-valuemax="100" aria-
valuetext="slider description">
    <label data-icon="moon">0</label>
    <label data-
icon="brightness">100</label>
    <div>
      <progress value="80"
max="100"></progress>
      <button>handler</button>
    </div>
  </div>
</section>
```

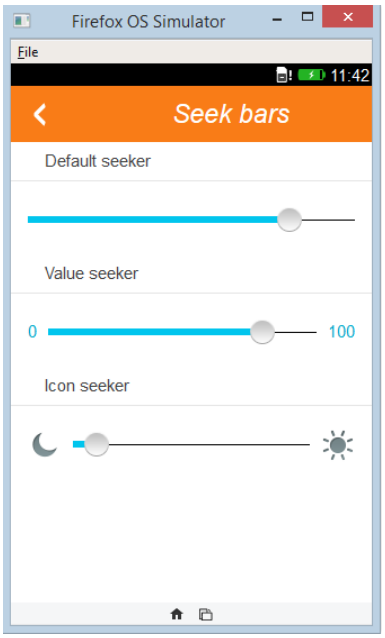


Ilustración 60 - Barras de búsqueda

- *Estado: muestra información en una ventana durante unos segundos*

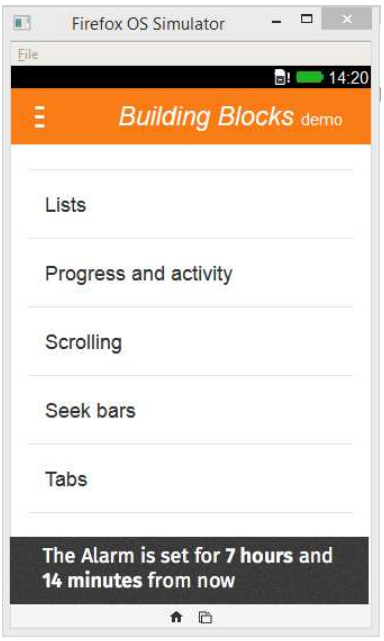


Ilustración 61 - Estado

Código CSS

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

Código HTML

```
<section role="status">
  <p>The Alarm is set for <strong>7 hours</strong> and <strong>14 minutes</strong> from now</p>
</section>
```

- *Selectores: elementos de activación/desactivación y selección de elementos*

Código CSS

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

Código HTML

```
<header>Checkbox, commonly used in edit
mode</header>
<div>
  <label class="pack-checkbox">
    <input type="checkbox" checked>
    <span></span>
  </label>
  <label class="pack-checkbox">
    <input type="checkbox">
    <span></span>
  </label>
  <label class="pack-checkbox danger">
    <input type="checkbox" checked>
    <span></span>
  </label>
  <label class="pack-checkbox danger">
    <input type="checkbox">
    <span></span>
  </label>
</div>

<header>Radio, commonly used in settings</header>
<div>
  <label class="pack-radio">
    <input type="radio" name="example" checked>
    <span></span>
  </label>
  <label class="pack-radio">
    <input type="radio" name="example">
    <span></span>
  </label>
  <label class="pack-radio danger">
    <input type="radio" name="example2" checked>
    <span></span>
  </label>
  <label class="pack-radio danger">
    <input type="radio" name="example2">
    <span></span>
  </label>
</div>

<header>Switch, commonly used in
settings</header>
<div>
  <label class="pack-switch">
    <input type="checkbox" checked>
    <span></span>
  </label>
  <label class="pack-switch">
    <input type="checkbox">
    <span></span>
  </label>
</div>
```

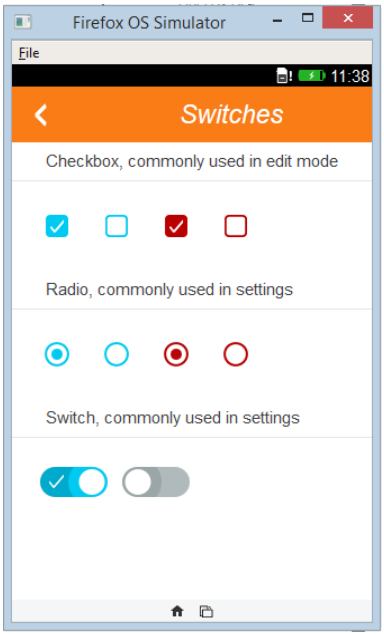


Ilustración 62 - Selectores

- *Pestañas*

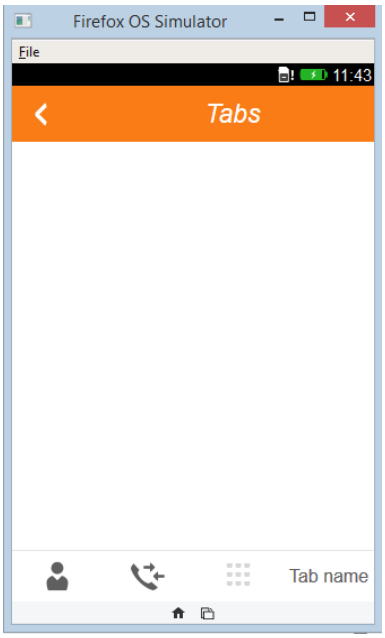


Ilustración 63 - Pestañas

Código CSS

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

Código HTML

```
<!–si las pestañas estan arriba, eliminar
class="bottom" -->
<ul role="tablist" class="bb-tablist">
  <li id="panel1" role="presentation">
    <a id="tab1" href="#panel1" data-
icon="contacts" role="tab" aria-
controls="tabpanel1">contacts</a>
    <div id="tabpanel1" class="bb-tabpanel"
role="tabpanel" aria-labelledby="tab1">Tab Panel
1</div>
  </li>
  <li id="panel2" role="presentation">
    <a id="tab2" href="#panel2" data-
icon="recent-calls" role="tab" aria-
controls="tabpanel2">recent calls</a>
    <div id="tabpanel2" class="bb-tabpanel"
role="tabpanel" aria-labelledby="tab2">Tab Panel
2</div>
  </li>
  <li id="panel3" role="presentation" aria-
disabled="true">
    <a id="tab3" data-icon="dialpad" role="tab"
aria-controls="tabpanel3">dialpad</a>
    <div id="tabpanel3" class="bb-tabpanel"
role="tabpanel" aria-labelledby="tab3">Tab Panel
3</div>
  </li>
  <li id="panel4" role="presentation">
    <a id="tab4" href="#panel4" role="tab" aria-
controls="tabpanel4">Tab name</a>
    <div id="tabpanel4" class="bb-tabpanel"
role="tabpanel" aria-labelledby="tab4">Tab
Panel 4</div>
  </li>
</ul>
```

3. DESARROLLO DE UNA APLICACIÓN EN FIREFOX OS.

3.1. Introducción

En este punto y en los sucesivos se detallará el proceso de construcción de un aplicación para Firefox OS. Se trata de un monitor deportivo (*MoDe*), que recogerá datos de distintas magnitudes en la realización de actividades deportivas al aire libre y los almacenará. Además ofrecerá la posibilidad de interactuar con ellos.

3.2. Objetivos del proyecto

El objetivo de este desarrollo consiste en analizar el proceso de creación de aplicaciones para el sistema operativo Firefox OS y especialmente la API de acceso a sus distintos componentes. Se analizará, diseñará e implementará una aplicación siguiendo el proceso de Ingeniería del Software para mostrar un caso real de utilización del terminal.

3.3. Metodología

La metodología que se usará en este proyecto se basa en el *Proceso Unificado*. Por tanto, el flujo de trabajo que se va a seguir en la realización de la aplicación es:

- **Análisis**. En esta fase se capturarán y describirán en detalles los requisitos del sistema y por último se definirá y describirá la interacción de los usuarios con el sistema.
- **Diseño**. A partir del análisis se describirá la solución del problema.
- **Implementación**. En esta fase se detallará la construcción del sistema.
- **Prueba**. Se indicarán las pruebas que se realizan al sistema para comprobar que los requisitos descritos en la etapa de análisis se cumplen.

En los siguientes apartados se detallarán las citadas etapas de la construcción de esta aplicación.

3.4. Planificación

El proceso de planificación va a comenzar con el establecimiento de las tareas que desarrollaremos hasta llegar a la consecución del mismo, para lo cuál se realiza la siguiente tabla de tareas:

ACTIVIDADES
Búsqueda bibliográfica
Revisión bibliográfica
Formación en las tecnologías a usar
Elaboración de la memoria
Análisis de requisitos
Diseño de la aplicación
Desarrollo y prueba
Evaluación de la interfaz
Finalización de la memoria

3.5. Estimación de tiempos

Una vez conocidas las tareas necesarias para la consecución del proyecto, se estima la duración de cada una de ellas, con lo que podremos obtener una estimación temporal completa. Vemos las duraciones en la siguiente tabla:

ACTIVIDAD	DURACIÓN (En días)
Búsqueda bibliográfica	2
Revisión bibliográfica	1
Formación en las tecnologías a usar	5
Elaboración de la memoria	5
Análisis de requisitos	2
Diseño de la aplicación	6
Desarrollo y prueba	15
Evaluación de la interfaz	3
Finalización de la memoria	3

3.5.1.Diagrama PERT

Una vez establecidas las tareas se procede a determinar el flujo de realización de estas. Teniendo en cuenta que el proyecto va a ser realizado por una sola persona, no existe posibilidad de paralelizar tareas por lo que se realizarán una a continuación de otra, tal y como muestran la siguiente tabla y diagrama:

	TAREA A REALIZAR	TAREA PREDECESORA	DURACIÓN (En días)
A	Búsqueda bibliográfica	-	2
B	Revisión bibliográfica	A	1
C	Formación en las tecnologías a usar	B	5
D	Elaboración de la memoria	C	5
E	Análisis de requisitos	D	2
F	Diseño de la aplicación	E	6
G	Desarrollo y prueba	F	15
H	Evaluación de la interfaz	G	3
I	Finalización de la memoria	H	3

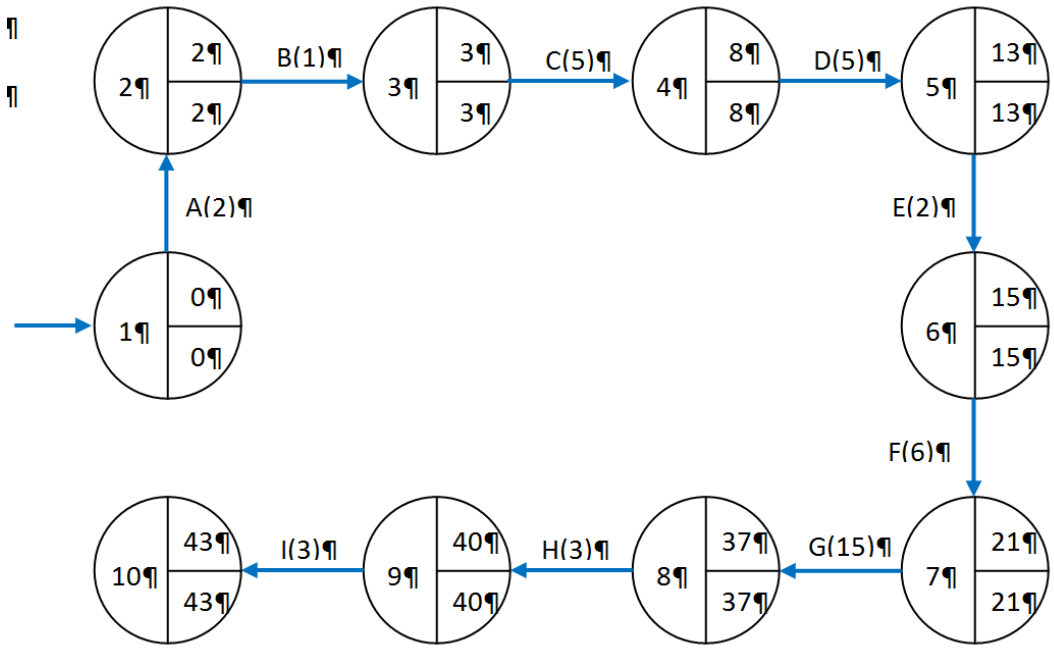


Ilustración 64 - Diagrama Pert

3.5.2. Diagrama de Gant

Se establece como fecha de comienzo del proyecto el día 3 de marzo de 2015 por lo que, aplicando las duraciones estimadas anteriormente, obtenemos el siguiente diagrama de Gant:

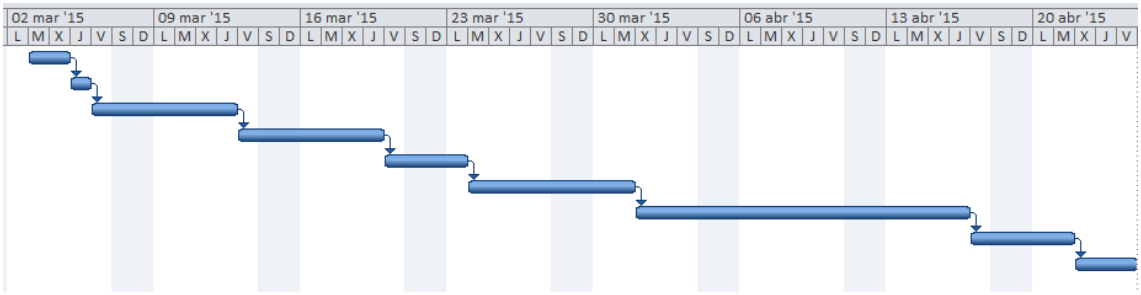


Ilustración 65 - Diagrama de Gantt

Se puede observar que la fecha de terminación estimada para el proyecto es el 25 de abril de 2015.

3.5.3. Calendario

Con todos los datos anteriores obtenemos un calendario detallado para cada una de las tareas, considerando días laborables de lunes a viernes.

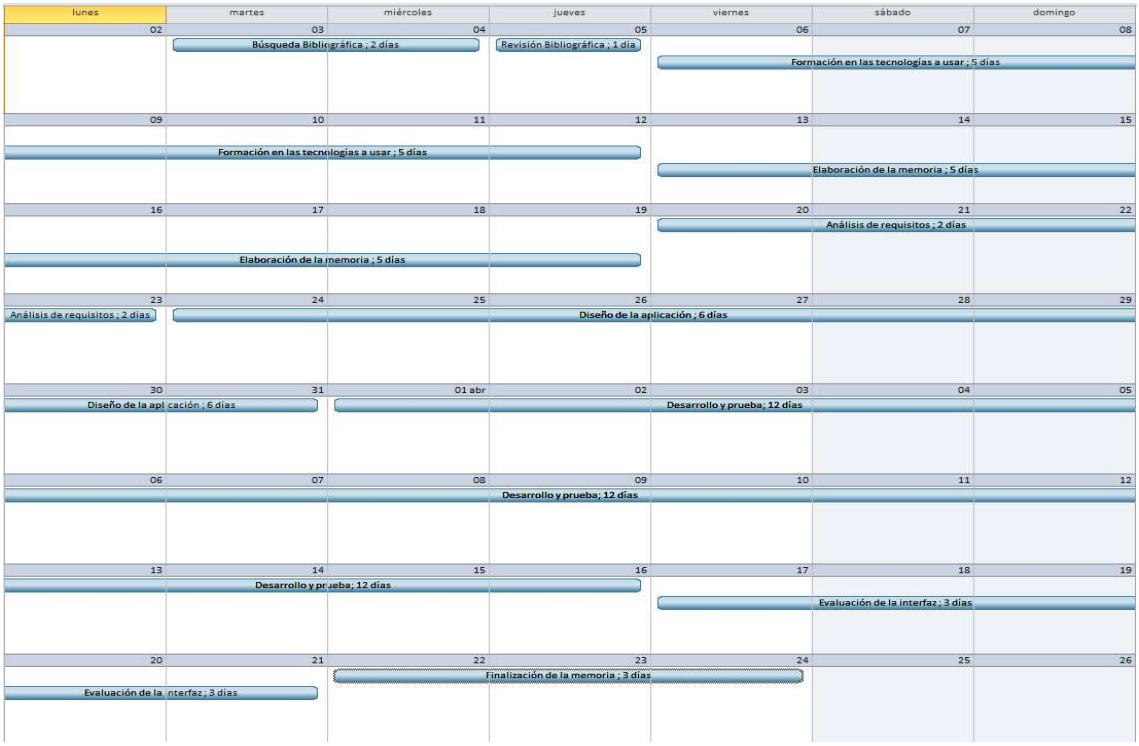


Ilustración 66 - Calendario del proyecto

3.6. Estimación de costes

Para la estimación de costes de desarrollo se usará el modelo COCOMO[13] como método de estimación.

Se estima que el tamaño del proyecto en líneas de código será de aproximadamente 1,5 KLDC (*Kilo Líneas De Código*), por lo que se aplicará el modo orgánico al ser este inferior a 50 KLDC.

MODO	A	B	C	D
Orgánico	2,40	1,05	2,50	0,38

Usando el modelo básico para obtener una primera estimación del esfuerzo de desarrollo tenemos:

$$E_d = a\ (KLDC)^{\ b} = 2{,}40\ (1{,}5)^{\ 1{,}05} = 3{,}67\ \textit{personas por mes}$$

En cuanto al tiempo de desarrollo:

$$Td = c\ (E_d)^{\ d} = 2{,}50\ (3{,}67)^{\ 0{,}38} = 4{,}1\ \textit{meses}$$

Y con los datos anteriores se puede determinar que el número de personas para realizar el proyecto será:

$$P = E_d / T_d = 3{,}67 / 4{,}1 = 0{,}89\ \textit{personas}$$

Es decir, que redondeando los datos, se necesitará una persona durante cuatro meses para realizar el proyecto.

Seguídamente se va a intentar afinar un poco más la estimación aplicando el modelo intermedio. Este usa una serie de modificadores que tienen en cuenta el entorno de trabajo incrementando la precisión de la estimación. En la siguente tabla se enumeran estos modificadores:

[13] **COCOMO**, (**CO**nstructive **CO**st **MO**del)

Modificador	Valor		
	Bajo	Nominal	Alto
Fiabilidad requerida del software	0,88	1,00	1,15
Complejidad	0,85	1,00	1,15
Restricciones en tiempo de ejecución	-	1,00	1,11
Restricciones de almacenamiento	-	1,00	1,06
Tiempo de respuesta del terminal	0,87	1,00	1,07
Capacidad del analista	1,19	1,00	0,86
Experiencia en la aplicación	1,13	1,00	0,91
Capacidad de los programadores	1,17	1,00	0,86
Experiencia en el S.O. usado	1,10	1,00	0,90
Experiencia en los lenguajes de programación usados	1,07	1,00	0,95
Prácticas de programación modernas	1,10	1,00	0,91
Utilización de herramientas software	1,10	1,00	0,91

El valor de la variable FAE será el producto de todos los modificadores elegidos, no se incluyen los que son igual a 1, pues no varían el resultado:

$$FAE = 0,88 * 1,11 * 1,07 * 1,13 * 1,10 * 0,91 * 0,91 = 1,07582924$$

Se recalcula el esfuerzo de desarrollo con el valor de FAE y obtenemos los siguientes resultados:

$$E_d = a\,(KLDC)^{b} = 2,40\,(1,5)^{1,05} * 1,07582924 = 3,95 \textit{ personas por mes}$$

$$Td = c\,(E_d)^{d} = 2,50\,(3,95)^{0,38} = 4,21 \textit{ meses}$$

$$P = E_d / T_d = 3,67 / 4,1 = 0,93 \textit{ personas}$$

Por lo que atendiendo a los resultados obtenidos con el modelo intermedio, se estima que será necesaria una persona durante cuatro meses aproximadamente, para la realización del proyecto.

El salario de un analista y el de un programador son distintos, pero para realizar esta estimación se va a usar un valor intermedio de 1500 € brutos al mes, por lo que el coste total, teniendo en cuenta, los cuatro meses que

necesitará para realizar el proyecto será de 6000 €. Este coste está un poco desvirtuado por la baja experiencia en este tipo de proyectos del desrrollador. Si se contase con un analista/programador experimentado en este tipo de proyectos, el coste se podría reducir notablemente.

Al coste de desarrollo le sumaremos el coste del hardware que se ha usado para realizar las pruebas, en concreto un terminal *ZTE Open* de *Movistar*, que ascendió a 49€.

Por lo tanto, el coste total estimado para la realización del proyecto asciende a 6049 €.

3.7. Proceso Unificado

A continuación se detallan las distintas etapas que componen esta metodología de Ingeniería del Software.

3.7.1. Análisis y especificación de requerimientos

El primer paso en el proceso de Ingeniería del Software debe ser determinar el propósito último del proyecto, las propiedades que debe satisfacer y las restricciones que tiene. Este es, un paso de vital importancia dentro del desarrollo de un proyecto software, ya que sin conocer el propósito del mismo ni establecer las diferentes limitaciones que debe afrontar, será realmente difícil realizar una aplicación software capaz de cumplir dicho propósito.

Dado que nos encontramos ante un proyecto generalista, orientado al público en general, se ha determinado el propósito pensando qué características podrían satisfacer al mayor número de personas. Tras haber definido el propósito del proyecto, pasaremos a especificar los requerimientos del mismo, es decir, el conjunto de propiedades o restricciones, definidas de forma totalmente precisa, que dicho proyecto ha de satisfacer. Existen dos tipos bien diferenciados de requerimientos, funcionales y no funcionales.

3.7.1.1. Requerimientos funcionales

Los requerimientos funcionales son aquellos que se refieren al funcionamiento del sistema, qué funcionalidades debe tener. A continuación se enumeran los que debe cumplir este proyecto:

- La aplicación será de acceso libre, no será necesario darse de alta en la aplicación para a ser usuario de ésta, ni logarse para usarla.
- El usuario podrá elegir la actividad a realizar.
- El usuario podrá activar y desactivar la autopausa en la grabación y elegir el valor deseado.
- El usuario podrá registrar rutas a través de grabación de trayectos mediante el GPS del terminal.
- La aplicación almacenará las rutas a nivel local en el terminal del usuario.
- El usuario podrá pausar o reanudar una ruta en cualquier momento.
- El usuario podrá guardar o descartar una ruta una vez concluida la grabación.
- El usuario podrá ver un listado de las rutas que tiene guardadas.
- El usuario podrá editar el nombre de las rutas.
- El usuario podrá eliminar una ruta individualmente o eliminar todas las rutas a la vez.
- El usuario podrá consultar datos relativos a cualquier ruta.
- El usuario podrá visualizar el mapa de una ruta.
- El usuario podrá añadir comentarios a las rutas.
- El usuario podrá configurar el autoapagado de la pantalla y activar o desactivar la vibración del dispositivo.
- La interfaz de la aplicación se realizará usando la plantilla Building Blocks.

3.7.1.2. Requerimientos no funcionales

Los requerimientos no funcionales son todos aquellos que no tienen relación con el funcionamiento del sistema, sino con factores externos. Este proyecto tendrá los siguientes requerimientos no funcionales:

- **Disponibilidad:** La aplición debe tener una disponibilidad total, y será sólamente reducida por los límites de autonomía del terminal, es decir, sólo dejará de estar disponible si el sistema operativo reduce las prestaciones del terminal cuando se llega a un determinado nivel de batería.
- **Facilidad de uso:** La aplicación será intuitiva y fácil de usar. Así mismo ofrecerá la información necesaria en caso de errores o funcionamiento anómalo de la misma.
- **Instalación:** La aplicación será fácil de instalar. Para ello se utilizarán los medios de instalación que ofrece el sistema operativo sobre el que correrá, *Firefox OS*.
- **Mantenibilidad:** La aplicación se desarrollará teniendo en cuenta una estructura con reglas y estándares que permitan que su mantenimiento y sus posteriores modificaciones sean sencillas.
- **Operatividad:** La aplicación tendrá una baja demanda de soporte por parte de los usuarios.
- **Fiabilidad:** Se buscará un alto nivel de fiabilidad frente a errores y situaciones no esperadas.
- **Portabilidad:** La aplicación estará diseñada para usarse en terminales móviles con *Firefox OS*, pero teniendo en cuenta que los lenguajes que se utilizarán en su implementación son HTML5, CSS3 y JavaScript, podrá portarse a otro tipo de sistemas con facilidad.

3.7.1.3. Requerimientos de la interfaz:

A pesar de que estos requerimientos estarían dentro de los requerimientos no funcionales, al tratarse este proyecto de una aplicación

para un smartphone y teniendo en cuenta la gran importacia de la interfaz en este tipo de sistemas vamos a tratar estos en un apartado distinto:

- **Facilidad de aprendizaje:** Las características de la interfaz deben permitir a los usuarios comprender cómo usar la aplicación. El usuario de percibir fácilmente los elementos de la interfaz y los cambios de estado del sistema.
- **Flexibilidad:** Debe haber distintas formas en que el usuario y el sistema intercambian información.
- **Robustez:** El sistema tiene que ser fiable, es decir, debe ser capaz de tolerar fallos durante la interacción.

3.7.2. Diagrama de casos de uso

Un caso de uso representa una funcionalidad concreta del sistema como un flujo de eventos, una tarea que los actores relacionados con el sistema realizan al utilizarlo.

En este caso, sólo pueden intervenir dos actores en el sistema:

- **Usuario:** Cada uno de los usuarios que pueden usar la aplicación.
- **Base de Datos:** Almacena y proporciona todos los datos relevantes para el funcionamiento del sistema.

Una vez identificados los actores, se crean los diferentes casos de uso en los que se representan y registran los requisitos funcionales que tendrá la aplicación. Vemos en la siguiente ilustración el diagrama resultante:

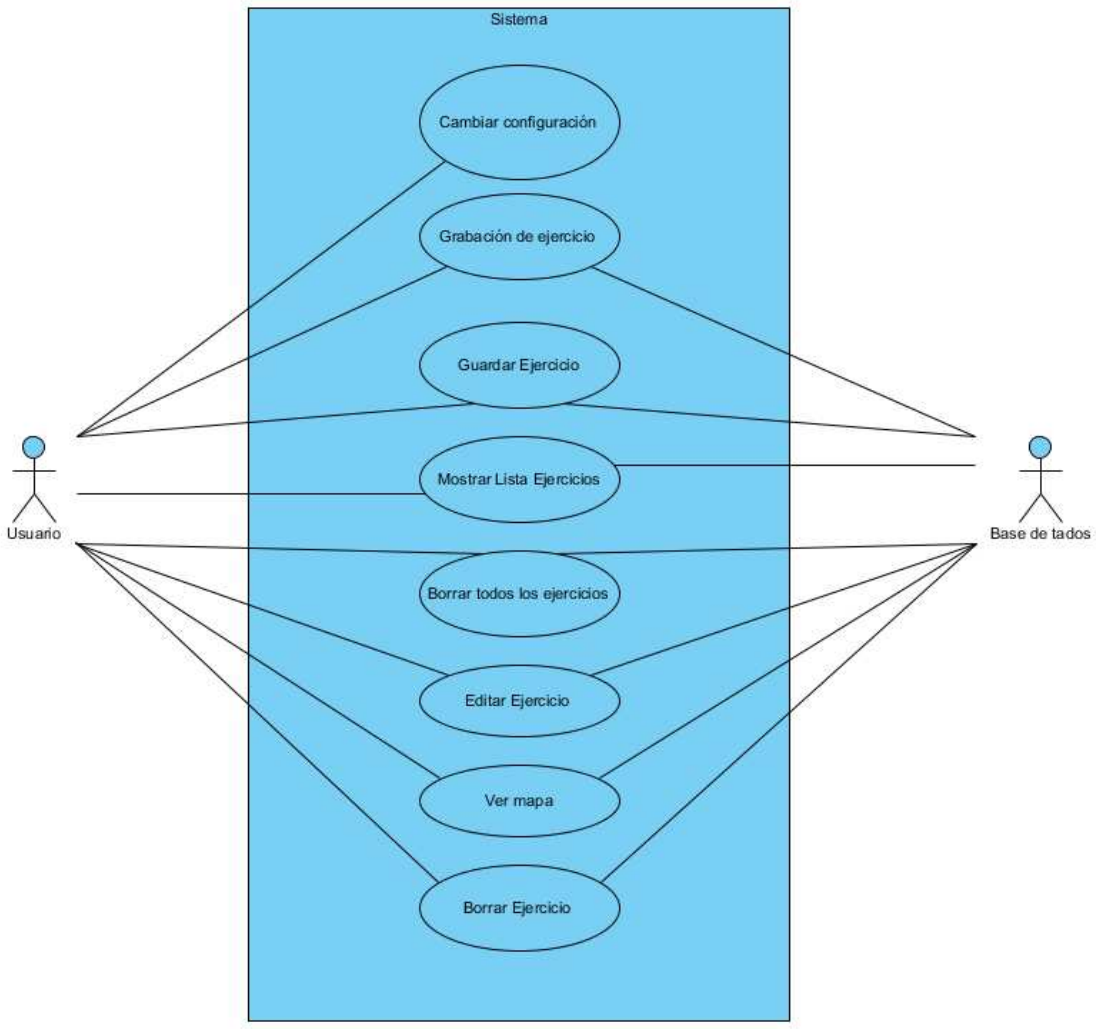


Ilustración 67 - Diagrama de casos de uso

3.7.3. Narrativa de casos de uso

A continuación se detalla cada uno de los casos de uso mostrados en la figura anterior.

CASO DE USO 1	
Nombre	Cambiar configuración
Actor principal	Usuario
Condición de entrada	El usuario inicia la aplicación y pulsa el botón de configuración
Flujo de eventos	1. El usuario pulsa el botón de configuración 2. El usuario modifica o no la configuración de los controles 3. El usuario pulsa el botón *Hecho*
Condición de salida	El sistema almacena la configuración y aplica los cambios correspondientes
Excepciones	No

CASO DE USO 2	
Nombre	Grabación de ejercicio
Actor principal	Usuario
Condición de entrada	El usuario inicia la aplicación, pulsa *Nuevo Ejercicio*, selecciona los valores de *Actividad* y *Autopausa* y pulsa *Continuar*
Flujo de eventos	1. El usuario pulsa el botón *Nuevo Ejercicio* 2. El usuario selecciona los valores de *Actividad* y *Autopausa* 3. El usuario pulsa el botón *Continuar*
Condición de salida	El sistema aplica los parámetros de *Actividad* y *Autopausa*, y muestra la pantalla de grabación de ejercicio
Excepciones	No

CASO DE USO 3	
Nombre	Guardar ejercicio
Actor principal	Usuario
Condición de entrada	El usuario inicia la aplicación e inicia la grabación de un ejercicio
Flujo de eventos	1. El usuario pulsa el botón *Parar* 2. El usuario pulsa el botón *Terminar* 3. El usuario pulsa el botón *Guardar*
Condición de salida	El sistema muestra la pantalla de resumen, guarda los datos en la base de datos y vuelve a la pantalla de inicio
Excepciones	No

CASO DE USO 4	
Nombre	Mostrar lista de ejercicios
Actor principal	Usuario
Condición de entrada	El usuario inicia la aplicación y pulsa *Ejercicios*
Flujo de eventos	1. El usuario pulsa el botón *Ejercicios*
Condición de salida	El sistema muestra el listado de ejercicios almacenados
Excepciones	No

CASO DE USO 5	
Nombre	Borrar todos los ejercicios
Actor principal	Usuario
Condición de entrada	El usuario inica la aplicación, pulsa *Ejercicios* y pulsa *Borrar Todos*
Flujo de eventos	1. El usuario pulsa el botón *Ejercicios* 2. El usuario pulsa el botón *Borrar Todos*
Condición de salida	El sistema elimina todos los datos de la base de datos
Excepciones	No

CASO DE USO 6	
Nombre	Editar ejercicio
Actor principal	Usuario
Condición de entrada	El usuario inica la aplicación, pulsa *Ejercicios* y pulsa *Editar*
Flujo de eventos	1. El usuario pulsa el botón *Ejercicios* 2. El usuario pulsa el botón *Editar*
Condición de salida	El sistema muestra la pantalla de edición de ejercicio del ejercicio correspondiente
Excepciones	No

CASO DE USO 7	
Nombre	Ver mapa
Actor principal	Usuario
Condición de entrada	El usuario inica la aplicación, pulsa *Ejercicios* y pulsa *Ver Mapa*
Flujo de eventos	1. El usuario pulsa el botón *Ejercicios* 2. El usuario pulsa el botón *Ver Mapa*
Condición de salida	El sistema muestra la pantalla de visualización del mapa del ejercicio correspondiente
Excepciones	No

CASO DE USO 8	
Nombre	Borrar ejercicio
Actor principal	Usuario
Condición de entrada	El usuario inicia la aplicación, pulsa *Ejercicios*, pulsa *Editar* y pulsa *Borrar*
Flujo de eventos	1. El usuario pulsa el botón *Ejercicios* 2. El usuario pulsa el botón *Editar* 3. El usuario pulsa el botón *Borrar*
Condición de salida	El sistema borra el ejercicio correspondiente
Excepciones	No

3.7.4.Análisis de la Interfaz

Para el análisis y la realización de la interfaz de usuario se usará la técnica de "*personas*" para representar a los futuros usuarios de la aplicación. Con la utilización de esta técnica se pretenden comprender las reacciones que éstos pueden tener al interaccionar con la aplicación y aclarar posibles aspectos de la aplicación que se presupongan a priori. Junto con el análisis de escenarios se podrán estudiar importantes aspectos de la interfaz del sistema. La aplicación de esta técnica, junto a las contenidas en el proceso unificado aportarán un entorno adecuado para revisiones, usabilidad, pruebas y documentación.

3.7.4.1. Personas

Datos personales:
Nombre: Sebastián
Apellidos: Pulido Guerrero
Edad: 39 años
Localidad: Jaén
Teléfono: BQ E5
S.O.perativo: Android 4.4
Deportes que practica: Fútbol, Running

Empresa: Universidad de Jaén
Puesto: Técnico Especialista
Conocimientos en smartphones: Nivel intermedio

De lunes a viernes Sebastián se levanta a las 7:00 de la mañana. Dispone de una hora para prepararse y desplazarse, dando un paseo, hasta su lugar de trabajo en la Universidad de Jaén. A las 8:00 comienza su jornada laboral en la que realiza servicios de atención a la comunidad universitaria. Alrededor de las 10:00 comienza su descanso de media hora, el cual aprovecha para desayunar con algunos compañeros. Su jornada laboral continúna has las 14:30. Una vez terminada, se dirige al colegio en el que se encuentran sus hijas y las recoge para irse a casa a comer.

Después de comer descansa un poco junto a su familia. Los días que sus hijas tienen actividades extraescolares las lleva y las recoge, si no tienen las ayuda con las tareas del cole.

A Sebastián le gusta prácticar un poco de deporte en su tiempo libre, entre semana a última hora de la tarde o a lo largo de los fines de semana. Las aplicaciones que más usa son Whatsapp y Facebook y cuando sale a hacer deporte le gustaría poder monitorizar sus recorridos para comprobar sus logros.

Datos personales:

Nombre: Alberto
Apellidos: Romero Olmo
Edad: 33 años
Localidad: Jaén
Teléfono: THL 5000
S.O.perativo: Android 4.4
Deportes que practica: Fitness, Ciclismo

Empresa: Universidad de Jaén
Puesto: Técnico Especialista
Conocimientos en smartphones: Nivel intermedio

La jornada comienza para Alberto a las 6:30 de la mañana. Comienza su jornada laboral a las 8:00, pero como tiene que desplazarse desde Mancha Real hasta Jaén, necesita levantarse con antelación suficiente.

Su jornada va desde las 8:00 hasta las 14:30, igual que la de su compañero Sebastián, con el que desayuna habitualmente en el descanso de las 10:00 y en el que aprovecha para actualizar sus redes sociales.

Después de comer, Alberto dispones de bastante tiempo libre, por lo que aprovecha para hacer bastante deporte. Normalmente acude al gimnasio durante un par de horas dos o tres días en semana. Los días que no va al gimnasio y si hace buen tiempo , aprovecha para hacer una ruta en su Mtb de 40 o 50 km. Para estas salidas en bici le gusta monitorizar sus trayectos y sus tiempos, para comprobar su evolución.

Cuando llega a casa sólo queda tiempo para cenar y ver alguna serie hasta las 0:00, hora en que normalmente se va a la cama.

Datos personales:

Nombre: M. Trinidad
Apellidos: Romero Olmo
Edad: 37 años
Localidad: Jaén
Teléfono: THL 4400
S.O.perativo: Android 4.4
Deportes que practica: Fitness

Empresa: Desempleada
Puesto: ---
Conocimientos en smartphones: Nivel básico

Un día en la vida de M. Trinidad, transcurre entre normativas, pues preparra oposiciones con el fin de conseguir la plaza de trabajo estable.

Normalmente comienza a estudiar a las 7:30. Alrededor de las 10:00 realiza un descanso para desayunar y aprovecha para revisar Whatsapp por si tiene mensajes nuevos. Después continúa estudiando hasta que llega la hora de comer.

Después de comer, descansa un rato, aproximádamente hasta las 16:00 y vuelve a retomar el estudio.

Le gusta prácticar un poco de deporte de forma diaria, así que sobre las 20:00 acude al gimnasio y realiza alguna clase colectiva.

Algunos días le gusta ir al gimnasio caminando, por lo que le podría venir bien una aplicación para saber cuanto tarda en llegar y la distancia que ha recorrido.

De regreso en casa sobre las 21:30, cena y charla o hace video conferencia con David, su pareja que trabaja en Bristol (UK), durante unos minutos. Si le queda tiempo sigue alguna serie. A las 23:30 suele irse a la cama.

Datos personales:

Nombre: M. Ángeles
Apellidos: Sánchez Godino
Edad: 36 años
Localidad: Torredelcampo
Teléfono: Samsung Note 3
S.O.perativo: Android 4.4
Deportes que practica:
Patinaje, Running, mantrailing

Empresa: Unicaja
Puesto: Técnico de informática
Conocimientos en smartphones: Nivel avanzado

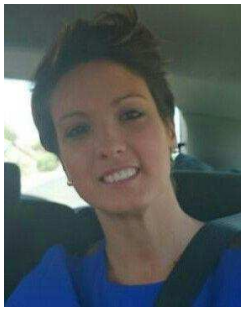
A Mari Ángeles le suena el despertador cada mañana a las 6:10, los días laborables. Se coloca su ropa deportiva y le coloca el arnés a su querida amiga Keira. Salen a pasear o correr entre las 6:30 y las 7:10. Cuando vuelve a casa se da una ducha rápida y se dirige a su lugar de trabajo en la ciudad de Jaén. Como entra a trabajar a las 8:00, de camino mira las noticias y las redes sociales en el móvil. Sobre las 9:30 sale a desayunar con los compañeros, y también aprovecha para actualizar redes sociales.

A las 15:00, termina su jornada laboral y se desplaza a casa, donde Keira la está esperando para su paseo de medio dia, éste muy cortito. Se prepara la comida y descansa hasta las 17:30.

Dependiendo del día, la tarde transcurre entre trabajo, ordenador, casa, amigos, familia…

Sobre las 20:30-21:00, visita a sus padres, (aprovechando la última salida de Keira) y les ayuda con ciertas tareas que ellos ya no pueden realizar. Vuelve a casa sobre 22:30/23:00, prepara la cena, y da de comer a sus animales (a parte de Keira tiene una liebre y un acuario con peces). Organiza un poco la casa, ve la tele un rato y llega la hora de irse a la cama, entre 00:00 - 01:00.

Para ella el móvil es una herramienta fundamental, tanto a nivel de trabajo como a nivel personal y los usa casi para todo, consulta de noticias, redes sociales, correo electrónico, actividades deportivas, etc.



Datos personales:

Nombre: M. Magdalena
Apellidos: Navarrete García
Edad: 31 años
Localidad: Jaén
Teléfono: LG G2
S.O.perativo: Android 4.4
Deportes que practica:
Pilates, paseos

Empresa: Sodexo
Puesto: Jefa de centro
Conocimientos en smartphones: Nivel básico

Magdalena comienza su día a las 6:30, se prepara rápidamente y coge el coche, pues empieza a trabajar a las 7:30 y debe desplazarse desde Mancha Real hasta Jaén.

A su llegada al hospital comprueba su teléfono para ver si tiene nuevas notificaciones y comienza a trabajar. Sobre las 10:00 tiene un descanso de media hora, el cuál aprovecha para desayunar e intercambiar algunos mensajes.

Sale del hospital sobre las 15:00, come y se desplaza de nuevo hasta su casa donde descansa hasta las 17:00.

Después del descanso aprovecha la hora de pasear a sus perritas para dar un paseo y andar un rato. Le gustaría almacenar algunos datos, como los itinerarios, la distancia y el tiempo que emplea, en su teléfono.

Sobre las 21:00 espera la llegada de Alberto, su pareja y cenan juntos y ven algún programa de televisión hasta la hora de irse a dormir, alrededor de las 23:30.

3.7.4.2. Escenarios actuales

- Escenario 1: Sebastián sale a hacer running

Sebastián sale a correr en el parque que tiene justo enfrente de casa. Este está circunvalado por una pista para realizar este deporte. Normalmente, controla el tiempo que dura el ejercicio ayudándose del reloj pero no sabe que distancia logra recorrer ni en cuanto tiempo es capaz de realizar una vuelta.

- Escenario 2: Alberto sale con la bici

En algunas ocasiones a Alberto le gusta pedalear trayectos ya conocidos, para así poder comprobar si va mejorando su forma física. El dispositivo que lleva conectado su bici sólo le indica que distancia ha recorrido y que velocidad máxima ha alcanzado. Otras veces simplemente circula por el primer carril o sendero que encuentra para así descubrir nuevas rutas y nuevos paisajes.

- Escenario 3: Trini camina a la Universidad

Trini se desplaza todos los días a la biblioteca de la Universidad. Los días en los que hace buen tiempo le gusta realizar el trayecto a pie y así aprovecha para hacer algo de deporte. El trayecto de regreso también lo realiza a pie.

- Escenario 4: Mari Ángeles y Keira salen a pasear

Mari Ángeles y Keyra salen a pasear varias veces a lo largo del día pero el paseo de la tarde es el más largo. Aprovechan para caminar por algún sendero cercano a casa y realizan trayectos de hasta 4 km.

- Escenario 5: Magdalena da un paseo

Desde hace algún tiempo, Magdalena no puede realizar actividades físicas muy intensas, por lo que pasea durante aproximádamente una hora por el parque que tiene al lado de casa.

3.7.4.3. Escenarios futuros

- Escenario 1: Sebastián sale a hacer running y usa MoDe

Como de costumbre, Sebastián sale a correr por el parque de enfrente de casa. Ha decidido almacenar su ejercicio usando MoDe, por lo que inicia la aplicación, crea un nuevo ejercicio de atletismo sin autopausa y comienza a correr. Ahora puede comprobar el tiempo que realiza en cada vuelta y las distancias totales y parciales. Almacena el ejercicio para compararlo con siguientes y ocasiones y comprobar sus progresos.

- Escenario 2: Alberto sale con la bici y usa MoDe

En esta ocasión Alberto ha elegido salir con la bici a descubrir lugares nuevos. Coloca el teléfono en el soporte de su bici e inicia un nuevo ejercicio de ciclismo sin autopausa. A medida que va avanzando va comprobando su localización en el mapa, la distancia recorrida y la velocidad. Una vez concluido lo almacena para posteriores revisiones.

- Escenario 3: Trini camina a la Universidad y usa MoDe

Como es habitual Trini se dirige hacia el campus de la Universidad. Hoy ha decidido que va a controlar cuanto tiempo tarda en llegar por la ruta que hace habitualmente. En días sucesivos elegirá otras para comprobar cuál de ellas es la más rápida.

- Escenario 4: Mari Ángeles y Keira salen a pasear usando MoDe

Cuando Mari Ángeles y Keyra salen a pasear, siempre monitoriza en su smartwatch la distancia recorrida y el tiempo que emplea. Esta vez ha decidio utilizar MoDe para contrastar la información con los datos que ya tiene, con el objetivo de determinar los posibles márgenes de error de uno y otro.

- Escenario 5: Magdalena da un paseo y usa MoDe

A Magdalena realmente sólo le interesa el tiempo cuando sale a pasear, pues sólo dispone de una hora, pero hoy ha decidido usar MoDe para comprobar, por curiosidad, qué distancia es capaz de recorrer.

3.7.4.4. Storyboard

El siguente gráfico describe las transiciones de la interfaz:

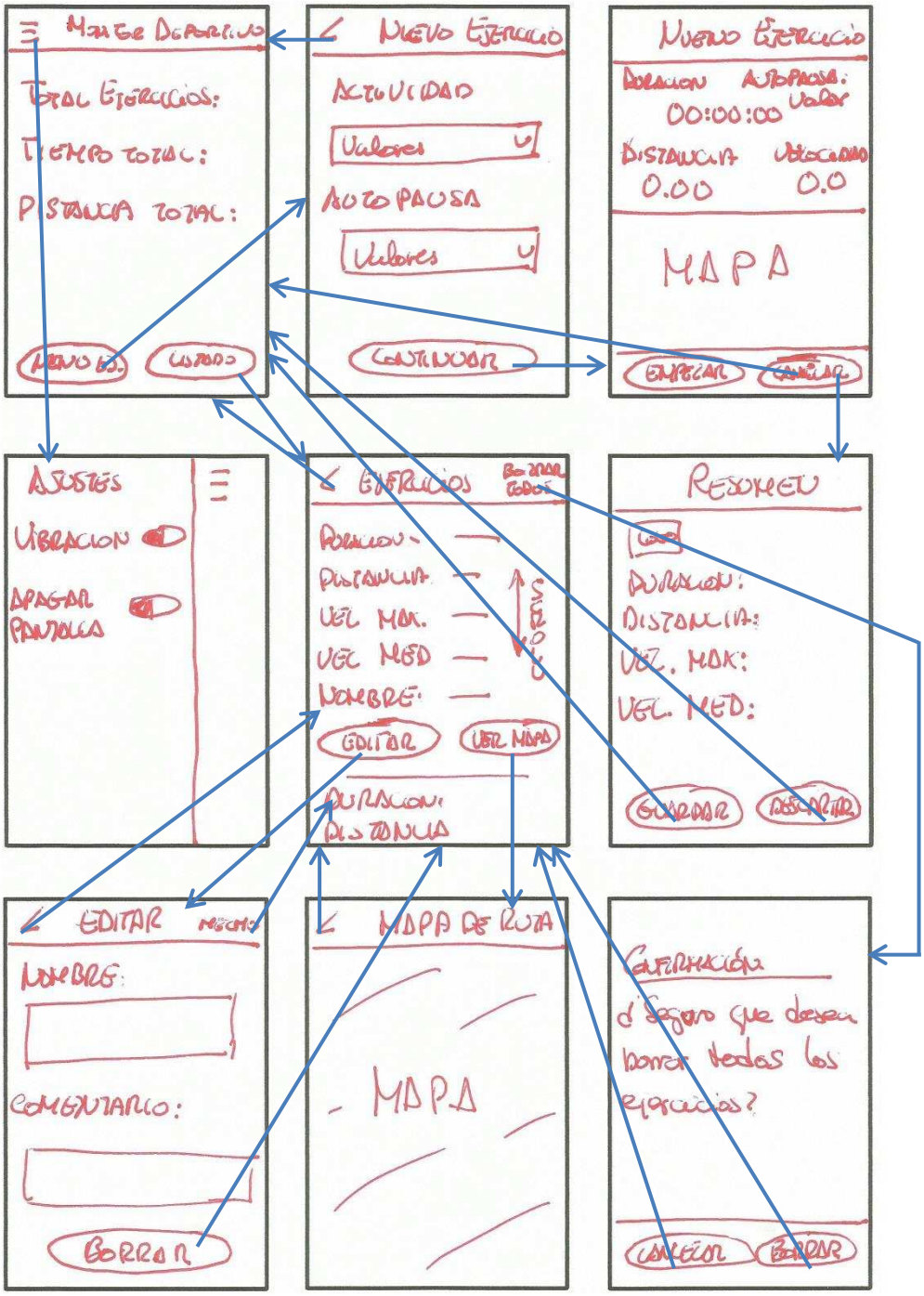


Ilustración 68 - Storyboard

3.7.4.5. Manual de usuario

Para ejecutar la aplicación pulsando el icono de la aplicación MoDe, que se encuentra en el panel de aplicaciones, desplazando el escritorio hacia la izquierda.

Ilustración 69 - Panel de aplicaciones

Una vez pulsado el icono aparece en el terminal la página de inicio.

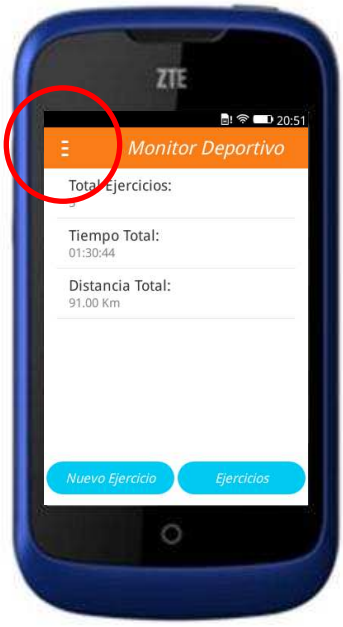


Ilustración 70 - Pantalla de inicio

Como se puede ver en la ilustración anterior, la pantalla de incio nos muestra el número total de ejercicios almacenados, el tiempo total empleado y la distancia total recorrida.

En la parte derecha de la cabecera se encuentra el botón de acceso a la pantalla de configuración.

Ilustración 71 - Pantalla de configuración

Con el botón *Vibración* podemos activar o desactivar la vibración que se produce cuando se pulsa cualquier botón de la aplicación. Con el botón *Apagar Pantalla* se define si la pantalla se apagará o no en la realización de un ejerecicio. Si Apagar Pantalla se encuentra desactivado, la pantalla no se apagará durante la realización de un ejercicio, mientras que si está activo la aplicación tomará el tiempo establecido en la configuracion del teléfono para apagar la pantalla.

Una vez seleccionadas las opciones se pulsa el botón *Hecho* y los cambios son aplicados y guardados.

Para comenzar la grabación de un ejercicio pulsamos el botón *Nuevo Ejercicio* que nos llevará a la pantalla de selección de la actividad y la autopausa

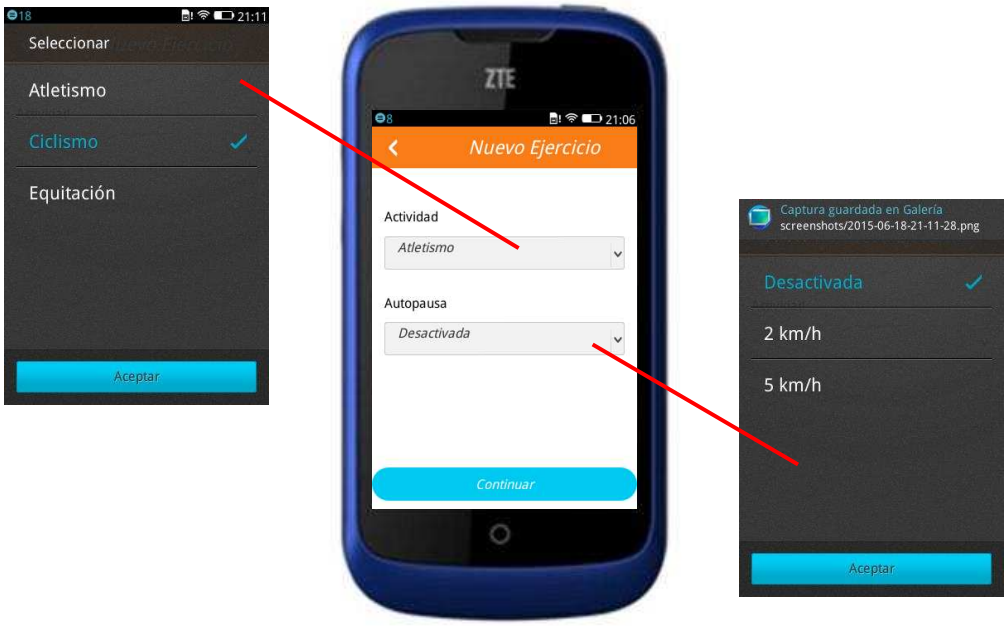


Ilustración 72 - Selección de parámetros para nuevo ejercicio

Pulsando en cada uno de los desplegables se podrá elegir la opción desada. El valor de autopausa hace que la grabación del ejercicio se pause automáticamente en caso de que la velocidad esté por debajo del valor elegido. Pulsando el botón continuar pasaremos a la pantalla de grabación de ejercicio.

Cuando accedemos a la pantalla de grabación de ejercicio por primera vez, el teléfono nos pedirá confirmación de si desamos compartir nuestra ubicación. Activamos el botón *Recordar mi elección* y pulsamos *Compartir* para que la aplicación ubique nuestra posición en el mapa.

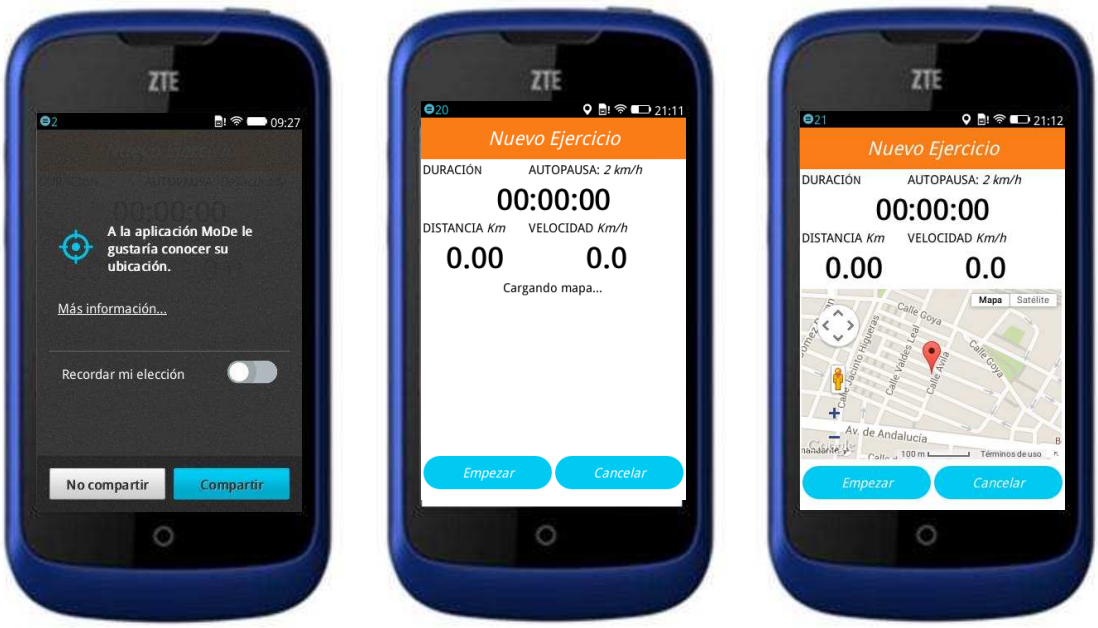


Ilustración 73 – Pantalla de grabación de ejercicio

En esta pantalla se puede observar la información ofrecida por la aplicación, la duración, el valor seleccionado para la autopausa, la distacia recorrida, la velocidad instantánea, así como el mapa y la ubicación actual.

Pulsando el botón *Empezar,* comenzará la grabación, y la aplicación dibujará en el mapa la ruta del desplazamiento registrado.

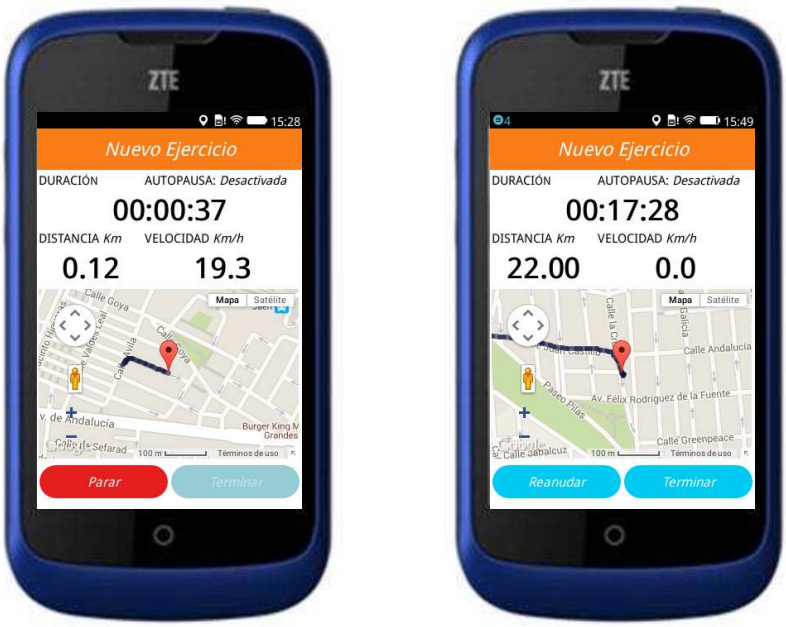


Ilustración 74 - Grabación en curso

Durante la grabación, si pulsamos *Parar,* la grabación se pausará y se mostrará el botón *Reanudar,* que si es pulsado reanudará la grabación. Si una vez pausado el ejercicio, pulsamos *Terminar*, pasaremos a la pantalla de resumen.

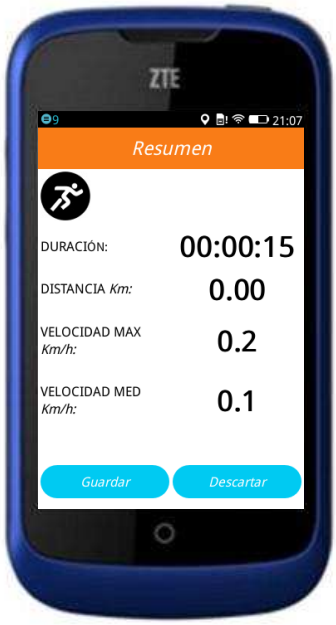


Ilustración 75 - Pantalla de resumen

Si pulsamos *Guardar,* el ejercicio será guardado y la aplicación volverá a la pantalla de inicio, mientras que si pulsamos *Descartar,* se volverá a la pantalla de inicio y el ejercicio no se guardará.

Para ver el listado de ejercicios que tenemos guardados se pulsará el botón *Ejercicios,* lo que nos llevara a la pantalla de listado. En el listado se visualizan los datos almacenados para cada ejercicio. Deslizando el dedo sobre la pantalla de forma vertical podremos movernos a través de dicha lista.

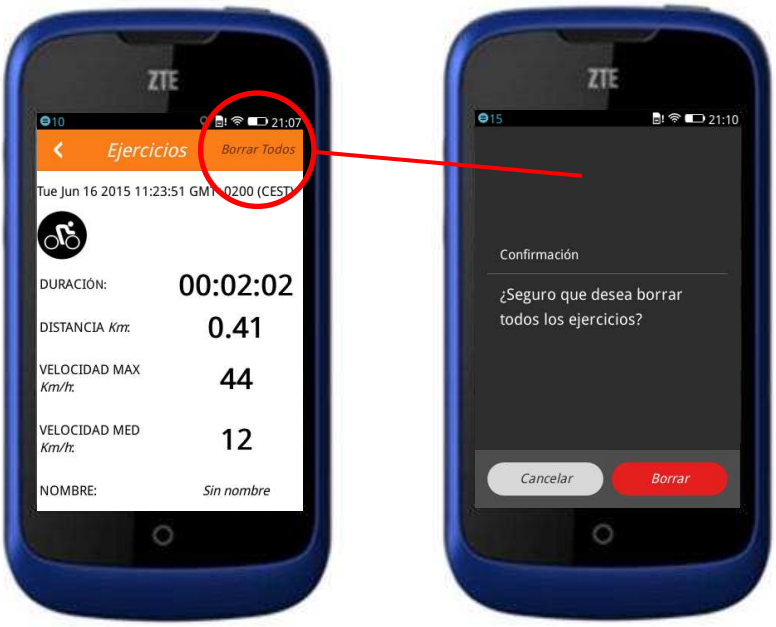


Ilustración 76 - Listado de ejercicios

Pulsando el botón *Borrar Todos* se borrarán todos los ejercicios guardados en memoria.

Cada ejercicio de la lista nos permite tanto editar el nombre y la descripción, como visualizar el mapa de la ruta guardada.

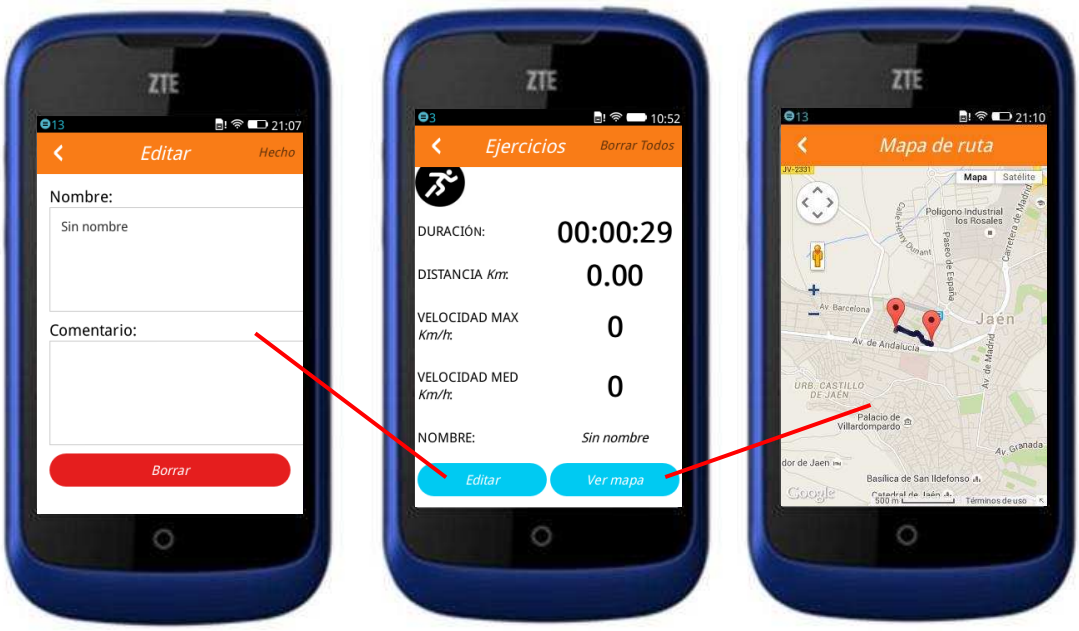


Ilustración 77 - Opciones de edición y visualización de la ruta

Una vez introducidos el nombre y el comentario, pulsando el botón *Hecho,* se almacenarán los datos y se volverá al listado.

En la pantalla de editar un ejercicio se ofrece además la posibilidad de eliminarlo pulsando el botón *Borrar*, igual que en el caso de borrar todos, la aplicación nos pedirá que confirmemos la eliminación.

3.8. DISEÑO.

3.8.1. Introducción

Una vez realizado el análisis de la aplicación se realiza el diseño desde el punto de vista del dominio de la solución.

3.8.2. Diagrama de clases

En el siguiente diagrama de clases, se muestran las clases que contiene la aplicación, así como sus atributos y métodos.

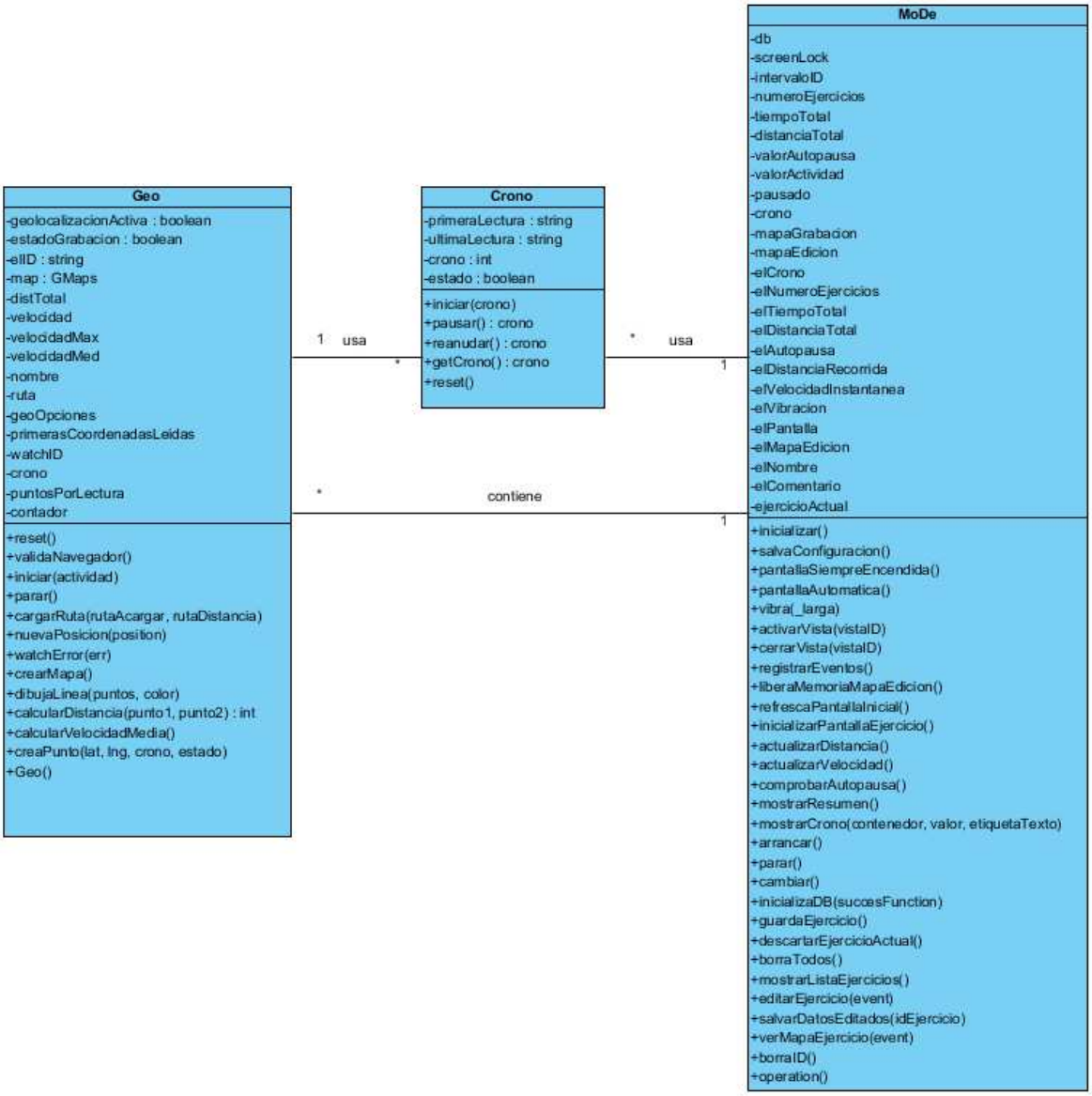


Ilustración 78 - Diagrama de clases

3.8.2.1. Descripción de clases

A continuación se detalla cada una de las clases del diagrama anterior:

- **Crono:** es una clase que implementa un cronómetro. Almacena el tiempo transcurrido desde que se inicia hasta que se para.
- **Geo:** clase que se ocupa de toda la funcionalidad relacionada con la geolocalización, cálculo de distancias y velocidades. Se ocupa también de la creación del mapa y la representación de la información en él.
- **MoDe:** esta clase engloba a toda la aplicación y gestiona el control de la misma, se encarga de la visualización de los datos, las transiciones entre pantallas y la gestión de la base de datos.

3.8.3.Diseño de datos

Para almacenar los datos de la aplicación se va a usar *IndexedDB,* base de datos que fue descrita en apartados anteriores. Esta base de datos almacena colecciones de objetos con una estructura determinada. En concreto, la aplicación que se está diseñando almacenará una colección de objetos con el siguiente formato:

```
ejercicioActual: {
  "ID": undefined,              //valor numérico autoincrementado
  "fecha": new Date(),          //cadena de caracteres
  "actividad": undefined,       //cadena de caracteres
  "duracion": 0,                //valor numérico
  "distancia": 0,               //valor numérico
  "velMedia": undefined,        //valor numérico
  "velMaxima": undefined,       //valor numérico
  "nombre": undefined,          //cadena de caracteres
  "coordRuta": undefined,       //array de puntos
  "comentario": ""              //cadena de caracteres
}
```

Todos los campos almacenan datos de tipos simples, a excepción de *fecha* que almacena un dato de tipo *Date()*, que devuelve la fecha actual del sistema y *coorRuta* que almacena objetos de tipo compuesto con la siguiente estructura:

```
punto: {
  "lat": lat,                                  //valor numérico
  "lng": lng,                                  //valor numérico
  "crono": this.crono.getCrono(),              //valor numérico
  "estado": estado,                            //booleano
  "cronoVelocidad": this.cronoVelocidad.getCrono()   //valor numérico
}
```

En esta estructura todos los campos son tipos simples, *crono* y *cronoVelocidad* almacenan valores numéricos devueltos por el método *getCrono()* de los objetos de la clase *Crono, crono* y *cronoVelocidad.*

3.8.4.Diseño de la interfaz

El diseño de la interfaz se realizará, enteramente, basándose en la plantilla *Building Blocks*, que se describió en apartados anteriores, satisfaciendo de esta forma el requisito funcional expresamente detallado.

3.9. IMPLEMENTACIÓN.

En este punto, se tiene definido el problema y su solución, por lo que se comienza a construir el sistema que debe satisfacer las necesidades del usuario.

Convertir el diseño en código es la parte más obvia del trabajo de ingeniería del software, siempre y cuando, el análisis y el diseño se hayan realizado correctamente.

3.9.1. Lenguajes de programación

Cómo ya se explicó en el apartado de descripción de *Firefox OS*, los lenguajes de programación que se usan para realizar aplicaciones, para este sistema operativo, son los siguientes:

HTML (*Hyper Text Markup Language*), es un lenguaje de marcas diseñado para estructurar textos y presentarlos en forma de hipertexto, formato estándar de las páginas web. El código en HTML se escribe en forma de etiquetas, rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta cierto punto, la apariencia de un documento, y puede incluir o hacer referencia a programas llamados *"script"*, los cuales pueden interactuar con los navegadores web.

JavaScript, es un lenguaje de programación interpretado, dialecto del estándar *ECMAScript*. Es orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente *(client-side)*, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. JavaScript usa una sintaxis similar al *C*, aunque adopta nombres y convenciones del lenguaje de programación *Java*. Sin embargo *Java* y *JavaScript* no están relacionados y tienen semánticas y propósitos diferentes. Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje *JavaScript* de una implementación del *Document Object Model (DOM)*. *JavaScript* se interpreta en el agente de usuario, al mismo tiempo que las sentencias van descargándose junto con el código *HTML*.

CSS *(Cascade Style Sheets)*, es un lenguaje formal que ayuda a separar la estructura interna de un documento de su presentación externa. Las etiquetas de estilo *CSS* pueden presentarse tanto dentro de un documento *HTML* (entre las etiquetas *<style type="text/css"></style>* en la cabecera), como en un documento aparte (con extensión *.css*), que el documento HTML se encarga de llamar cuando sea necesario. Con el uso de *CSS*, no sólo se consigue separar la estructura de la presentación sino que también se consigue la centralización de los estilos, ya que una sola hoja de estilos *CSS* puede ser invocada por diferentes páginas de la aplicación Web, lo que ayuda de manera sustancial al mantenimiento de la coherencia y consistencia del diseño de la aplicación.

En el desarrollo del proyecto que nos ocupa, al ser una aplicación desarrollada para el sistema operativo *Firefox OS*, se deben tener muy en cuenta las restricciones a que este obliga, sobre todo en el apartado de directivas de seguridad *CSP*, que fueron detalladas en la exposición del sistema operativo en puntos anteriores, ya que limitan el uso de ciertas prácticas, que sí admiten los navegadores tradicionales.

En cuanto al apartado de estilos, el uso de la plantilla *Building Blocks,* que también se desarrollo en apartados anteriores, soluciona directamente la adaptación de la aplicación a la guia de estilos de *Firefox OS*, no siendo necesario prácticamente ningún cambio o añadido para la realización de la aplicación.

3.9.2.Herramientas de desarrollo.

Para la realización de esta aplicación se ha usado, además del entorno integrado, *WebIDE,* que proporciona *Firefox,* y que fue descrito en apartados anteriores, el editor de textos *Sublime Text,* en su versión 3. Cabe resaltar que la aplicación se podría haber desarrollado integramente usando *WebIDE,* pues dispone de todas las herramientas necesarias para ello, tanto de edición de ficheros como de depuración de la aplicación. El uso de *Sublime Text* ha sido sólo por familiaridad con el editor.

Sublime Text[14] es un editor de texto centrado principalmente en código que soporta Snippets, plugins varios y sistemas de construcción de código, pero también tiene lo necesario para escribir artículos o textos de manera habitual. En cualquier caso, donde *Sublime Text* brilla es en la cantidad y calidad de sus prestaciones, entre las que podemos encontrar algunas tan interesantes como la multi selección, el multi cursor y el multi layout, gracias a las que podremos editar mucho más fácilmente (y sobre todo rápidamente) cualquier código. Muy interesante también resulta el gran soporte nativo que tiene para diferentes lenguajes, como *Clojure*, *Perl*, *Javascript*, *Haskell*, *Erlango* o *Escala* entre otros. Además, podemos crear y guardar macros en cualquier momento para que nos faciliten aún más el trabajo, contando con muchas posibilidades para ello. También permite configurar todos los atajos de teclado que queramos, opción muy importante, ya que una vez tengamos creado nuestro propio estilo, llevar a cabo cualquier acción que en otros programas nos podía llevar casi un minuto, en *Sublime Text* apenas nos llevará unos segundos.

[14] http://www.sublimetext.com/

3.10. PRUEBAS

Una vez concluido el proceso de implementación, se comienza con la siguiente actividad asociada al método de ingeniería del software que estamos usando, esta actividad es la prueba. En este apartado se detallarán las pruebas realizadas al software desarrollado, se intentará valorar la calidad de la aplicación generada, así como detectar y corregir posibles errores. Las pruebas del software son un elemento crítico para la garantía de calidad del software y representan una revisión final del diseño y de la codificación.

La prueba es el proceso de ejecución de un programa con el intento deliberado de encontrar errores. Su objetivo es la verificación y validación del producto. La verificación se refiere al conjunto de actividades que aseguran que el software implementa correctamente una función específica y la validación se refiere a un conjunto diferente de actividades que aseguran que el software construido se ajusta a los requisitos del cliente.

La fase de prueba es un proceso consistente en diseñar casos de prueba, mediante los cuales se intentan encontrar puntos débiles del software para descubrir errores y corregirlos.

El diseño de pruebas está caracterizado por la imposibilidad de probar exhaustivamente el software. Puesto que no se pueden probar todas las posibilidades de funcionamiento del software, se eligen aquellas pruebas que por sus características puedan considerarse representativas del resto.

En este caso se diseñarán las pruebas atendiendo a un enfoque funcional o pruebas de caja negra, que consiste en analizar todas las funciones desarrolladas.

3.10.1. Enfoque funcional o pruebas de caja negra

Las pruebas de caja negra, también denominadas pruebas de comportamiento, permiten obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa.

Los casos de prueba, son un conjunto de condiciones o variables bajo las cuáles se determinará si un requisito de la aplicación es parcial o completamente satisfactorio.

Se pueden realizar muchos casos de prueba para determinar que un requisito es completamente satisfactorio. Con el propósito de comprobar que todos los requisitos de la aplicación son revisados, debe haber al menos un caso de prueba para cada requisito funcional de los definidos durante el análisis.

A continuación se muestra la lista de los grupos de pruebas realizadas para comprobar los anteriormente citados requisitos funcionales:

- Validación de configuraciones de pantalla y vibración.

Prueba 1	Descripción	El usuario accede al menú de configuraciones pulsando el botón situado en la parte superior izquierda de la pantalla.
	Condiciones de entrada	El usuario establece los valores del autoapagado de la pantalla y de la vibración a través de los controles y pulsa el botón *Hecho* en la parte superior derecha.
	Salida esperada	El sistema almacena los valores y activa o desactiva el autoapagado de la pantalla y la vibración según corresponda.
	Salida real	Igual que la esperada.

Ilustración 79 - Prueba 1

- Validación de elección de actividad y autopausa.

Prueba 2	Descripción	El usuario accede a la pantalla de nuevo ejercicio pulsando el bóton *Nuevo Ejercicio.*
	Condiciones de entrada	El usuario elige los valores de *actividad* y de *autopausa.*
	Salida esperada	El sistema establece el número de puntos gps a guardar y activa o desactiva la autopausa.
	Salida real	Igual que la esperada.

Ilustración 80 - Prueba 2

- Validación de grabación, pausa, y reanudación de un ejercicio.

Prueba 3	Descripción	El usuario accede a la pantalla de grabación pulsando el botón *Continuar.*
	Condiciones de entrada	El usuario pulsa los botones *Empezar* o *Cancelar.*
	Salida esperada	1. Si el usuario pulsa el botón *Cancelar*, se vuelve a la pantalla de incio. 2. Si el usuario pulsa el botón *Empezar*, este se pone de color rojo, el texto del botón cambia a *Parar*, se desactiva el botón *Terminar* y se incia la grabación, el cronómetro y el cálculo de distancia y velocidad. La ruta se dibuja en gris. 3. Si el usuario pulsa *Parar* el botón se pone de color azul, el texto cambia a *Reanudar*, se activa el botón *Terminar* y se detiene la grabación, el cronómetro y el cálculo de distancia y velocidad. La ruta se dibuja en rojo. 4. Si el usuario pulsa el botón *Reanudar* se vuelve al punto 1. 5. Si el usuario pulsa el botón *Terminar* se muestra la pantalla de Resumen.
	Salida real	Igual que la esperada.

Ilustración 81 - Prueba 3

- Validación de almacenamiento o descarte de un ejercicio.

Prueba 4	Descripción	El usuario accede a la pantalla de Resumen pulsando el botón *Terminar.*
	Condiciones de entrada	El usuario pulsa el botón *Guardar* o *Cancelar.*
	Salida esperada	1. Si el usuario pulsa *Guardar*, el ejercicio será guardado y se volverá a la pantalla inicial. 2. Si el usuario pulsa *Descartar*, el ejercico será descartado y se volverá a la pantalla inicial.
	Salida real	Igual que la esperada.

Ilustración 82 - Prueba 4

- Validación de acceso al listado de ejercicios y sus datos

Prueba 5		
	Descripción	El usuario accede a la pantalla Ejercicios pulsando el botón *Ejercicios*
	Condiciones de entrada	El usuario pulsa el botón atrás, el botón *Borrar Todos*, el botón *Editar* de un ejercicio o el botón *Ver Mapa* de un ejercicio.
	Salida esperada	1. Si el usuario pulsa el botón de atrás, situado arriba a la izquierda, se vuelve a la pantalla anterior. 2. Si el usuario pulsa el botón *Borrar Todos*, todos los ejercicios son borrados. 3. Si el usuario pulsa el botón *Editar* de un ejercicio, se accede a la pantalla Editar. 4. Si el usuario pulsa el botón *Ver Mapa* de un ejercicio, se accede a la pantalla Ver Mapa y se carga el mapa de la ruta de ese ejercicio.
	Salida real	Igual que la esperada.

Ilustración 83 - Prueba 5

- Validación de edición del nombre y/o comentarios de un ejercicio y de borrado del mismo.

Prueba 6		
	Descripción	El usuario accede a la pantalla Editar pulsando el botón *Editar* de un ejercicio.
	Condiciones de entrada	El usuario sustituye o no el nombre y/o introduce o no un comentario y pulsa el botón *Hecho* o pulsa el botón *Borrar*.
	Salida esperada	1. Si el usuario pulsa el botón *Hecho* el sistema almacena el nombre y del comentario actual. 2. Si el usuario pulsa el botón *Borrar*, el sistema borra el ejercicio actual.
	Salida real	Igual que la esperada.

Ilustración 84 - Prueba 6

- Validación de visualización del mapa asociado a un ejercicio.

Prueba 7		
	Descripción	El usuario accede a la pantalla Ver mapa pulsando el botón Ver mapa de un ejercicio.
	Condiciones de entrada	El usuario pulsa el botón atrás o interactúa con el mapa.
	Salida esperada	1. Si el usuario pulsa el botón de atrás, situado arriba a la izquierda, se vuelve a la pantalla anterior. 2. El usuario puede ver la ruta realizada e interactuar con el mapa.
	Salida real	Igual que la esperada.

Ilustración 85 - Prueba 7

4. CONCLUSIONES

4.1. Conclusiones generales

Una vez estudiado *Firefox OS* y después de haber desarrollado una aplicación, se puede concluir que éste sistema operativo y las herramientas que *Mozilla* pone a disposición de los desarrolladores ofrecen un entorno simple, completo y de fácil utilización a la hora de realizar aplicaciones.

Sin duda, etre sus ventajas más apreciables tenemos que el código sea abierto, lo que facilita que cualquiera, con unos moderados conocimientos de programación, pueda desarrollar su propia aplicación. Otra gran ventaja es la facilidad para portar las aplicaciones a otros dispositivos, de hecho, cualquier aplicación desarrollada para *Firefox OS* se puede ejecutar en un navegador moderno y cualquier aplicación que se pueda ejecutar en un navegador moderno no suele necesitar de mucha adaptación para poder convertirse en una app de *Firefox OS.*

Cabe destacar también, los bajos requerimientos de hardware para ejecutar aplicaciones similares a las que se usan en otros dispositivos, cuyas prestaciones son mucho más elevadas, uno de los pilares en los que se apoyó la creación de este sistema operativo.

En cuanto a la planificación estimada, se produjo una desviación comparándola con la real, debido a la falta de experiencia en los lenguajes de programación y a algunos problemas técnicos relacionados con las políticas de seguridad de *Firefox OS.* Finalmente, el tiempo de ejecución se elevó a 63 días, aunque éste retraso no ha influido en la planificación de costes, que presuponía un tiempo de desarrollo superior, unos cuatro meses.

4.2. Mejoras y ampliaciones de la aplicación

Con los datos que actuálmente recoge la aplicación se pueden calcular gran cantidad de resultados y estadísticas. A continuación se ofrecen una serie de ampliaciones relacionadas con los datos recogidos:

- Cálculo de tiempo por kilómetro
- Cálculo de tiempo medio por kilómetro
- Calculo de la energía empleada en el desrrollo del ejercicio, introduciendo datos según edad, sexo y peso del usuario.

Además, se podrían recoger más datos conectando la aplicación a un monitor de ritmo cardíaco a través de Bluetooth. Con ello podríamos calcular estadísticas relacianadas con el rendimiento físico como: frecuencia máxima, frecuencia mínima, frecuencia media.

En cuanto a la compartición de la información, se podría desarrollar una funcionalidad para compratir ejercicios, vía bluetooth, por email o a través de redes sociales, con otros usuarios y una funcionalidad para que la aplicación mostrase información de cómo seguir una ruta recibida.

4.3. Conclusiones finales

Firefox OS es un sistema operativo, que nació a finales de 2012, sin pretensiones de ser una competencia directa para los grandes acaparadores del mercado mundial *iOS* y *Android.* Se concebió realmente para ser implantado en mercados emergentes en combinación con terminales de muy bajo coste, para llevar de esta forma la conectividad móvil a todo el mundo.

A lo largo de sus poco más de dos años de vida, su implantación ha sido bastante lenta, aunque poco a poco va ampliando posiciones.

En la actualidad se está impulsando el proyecto con el apoyo de grandes multinacionales como *Telefónica*, que apostó por *Firefox OS* desde su nacimiento, y la creación de nuevas alianzas.

En Estados Unidos, la compañia *Verizon* desarrollará una nueva gama de smartphones con este sistema operativo con el objetivo de ganar cuota de mercado. Los nuevos dispositivos dispondrán de todas las capacidades de los móviles avanzados, entre ellas las tecnologías LTE y VoLTE.

Por su parte *Orange,* la gran operadora francesa, tiene pensado llevar en el próximo trimestre sus primeros terminales *Orange Klif*, equipados con *Firefox OS*, a África, continente en el que tiene gran presencia. En total, *Orange,* opera en 13 mercados a nivel mundial y ha anunciado su intención de distribuir terminales con un coste inferior a 40 euros.

En cuanto a otro tipo de dispositivos, *Panasonic*, ha presentado su primera línea de televisores de "ultra alta definición" con éste sistema operativo, los *Panasonic Life+ 4K*.

Simultáneamente, en el MWC[15] 2015 celebrado en Barcelona, se ha producido el anuncio de la creación de una nueva versión del software para dispositivos "wearables", que según Joe Cheng, directivo de Mozilla, ofrecerá la posibilidad de interconectar una gran cantidad de dispositivos inteligentes, desde relojes hasta lavadoras. Aún no se sabe cuando se realizará el lanzamiento de esta nueva versión, aunque Cheng adelantó que se está negociando con varias compañias que están interesadas.

Teniendo en cuenta todo lo anterior, parece sensato pensar que 2016 puede ser un año crucial para todos estos proyectos, pero como bien es sabido, el mundo de la tecnología es complicado, así que habrá que ver cuál es el recorrido real de todas estas iniciativas.

[15] Mobile World Congress

Bibliografía

- Firefox OS – Todo lo que necesitas – Grandes características, apps y mucho más para smartphones

  https://www.mozilla.org/es-ES/firefox/os/2.0/

- 4 librerías JS que facilitan la programación de Webapps

  http://nosmoke.cycle-it.com/2014/02/03/4-librerias-js-que-facilitan-la-programacion-de-webapps/

- IndexedDB | MDN

  https://developer.mozilla.org/es/docs/IndexedDB

- Building Blocks: markup & examples

  http://buildingfirefoxos.com/building-blocks/action-menu.html

- Gmaps.js – Google maps api with less pain and more fun

  https://hpneo.github.io/gmaps/

- Maps API for work: Application Development & Analytics

  https://www.google.com/intx/en_uk/work/mapsearth/products/mapsapi.html?utm_source=HouseAds&utm_medium=cpc&utm_campaign=2015-Geo-EMEA-LCS-GEO-MAB-DeveloperTargetedHouseAds&utm_content=Developers&gclid=CjwKEAjwtYSsBRCDx6rM1v_uqmsSJAAZgf2qPTdjkzWdlk96TLr7DsE3ZDoZx0eOdpgkXf3AStjjphoCYWTw_wcB

- Zepto.js: the aerogel weight JQuery compatible JavaScript Library

  http://zeptojs.com/

- HTML (5), CSS, JavaScript Tutorials

  http://www.w3schools.com/