UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

INGENIERÍA INDUSTRIAL

"DISEÑO Y DESARROLLO DE UNA APLICACIÓN ANDROID CON FUNCIONALIDADES GPS TRACKING"

PROYECTO FINAL DE CARRERA

JUNIO - 2014

AUTOR: Saúl Irles Quiles
DIRECTORA: María Asunción Vicente Ripoll

VISTO BUENO Y CALIFICACIÓN DEL PROYECTO

Título proyecto:

Proyectante: _____

Director/es: _____

_____

V°B° director/es del proyecto: Fdo.: Fdo.:

Lugar y fecha: _____

CALIFICACIÓN NUMÉRICA ☐ MATRÍCULA DE HONOR ☐

Méritos justificativos en el caso de conceder Matrícula de Honor:

_____

_____

_____

_____

Conforme presidente: Fdo.:	Conforme secretario: Fdo.:	Conforme vocal: Fdo.:

Lugar y fecha: _____

0. Índice

1. Introducción

1.1. Resumen y objetivos del pfc

Con la aparición de los teléfonos inteligentes, también llamados “smartphones”, la tecnología ha pasado a formar parte de nuestra forma de vivir llegando a puntos insospechados hace únicamente unos pocos años. Estos aparatos ya están presentes en muchos aspectos de nuestra vida cotidiana. Además de su “principal” funcionalidad, la comunicación, también aparecen en aspectos como ir a hacer la compra, consultar precios a través de Internet, utilizarlos como cámara de fotos, como videoconsola, como lector de revistas, etc. Y en lo que nos ocupa este proyecto: Seguimiento de ubicación mediante GPS (Global Positioning System).

La miniaturización de los dispositivos móviles, menor peso y volumen, ha permitido que se conviertan en unos perfectos acompañantes para todo tipo de actividades, incluso las deportivas dónde se premia la comodidad.

Por ello, el siguiente proyecto tratará de utilizar esta potente herramienta, y en particular sus funcionalidades GPS, para hacer un seguimiento de la localización del usuario programando bajo la plataforma Android.

Los objetivos a alcanzar se enumeran a continuación:

- Diseño de una interfaz gráfica amigable y completa.
- Seguimiento de la ubicación del usuario instantáneamente.
- Representación de la secuencia de posiciones del usuario sobre un mapa utilizando Google Maps API v.2.
- Extracción de los datos más significativos y mostrarlos en pantalla.
- Almacenamiento de las actividades para su posterior visualización y análisis.
- Cumplir con los estándares necesarios para poder ser publicado en Google Play.

1.2. Introducción histórica a la tecnología móvil

Durante muchas décadas se le otorgó la invención del teléfono al escocés-americano Alexander Graham Bell, que patentó la idea en 1876. Sin embargo, en 2002, el Congreso de Estados Unidos aprobó una resolución que reconocía como inventor del mismo al italiano Antonio Meucci, que en 1871 presentó la idea pero por problemas económicos no pudo formalizar la patente.

Lamentablemente, muchos de los avances matemáticos, científicos y tecnológicos se producen a partir de una necesidad en medio de un conflicto bélico para obtener ventajas estratégicas sobre el enemigo. Y la idea de la comunicación inalámbrica nace bajo la sombra de la segunda guerra mundial, donde Motorola proporcionó al ejército estadounidense el aparato Handie Talkie para la comunicación entre tropas vía ondas de radio con una banda de frecuencia de 600kHz.

A partir de finales de los años 40, las empresas pioneras en el sector de la comunicación telefónica iniciaron una carrera por conseguir la tan ansiada telefonía móvil.

Los avances respecto la comunicación inalámbrica eran enormes, se conseguían señales con pocas interferencias y mayor alcance, sin embargo el verdadero conflicto era hacer esta tecnología realmente portátil. Y esto se consiguió de la mano de Motorola el 3 de Abril de 1973, cuando Martin Cooper (director de Motorola) realizó la primera llamada de la historia a través de un dispositivo móvil, el teléfono DynaTAC 8000X.

En la foto siguiente se muestra a Martin Cooper en 2013, con un modelo del terminal DynaTAC 8000X en su mano derecha y en la otra el Moto G, unos de los últimos terminales de la compañía Motorola.

Imagen 1. Martin Cooper en 2013.

A continuación se muestra una tabla que compara las especificaciones físicas de los terminales anteriormente citados:

	DynaTAC 8000X	Moto G
Peso	800 gr	143 gr
Dimensiones	33.02 x 4.445 x 8.89 cm	12.99 x 6.59 x 1.16 cm

Tabla 1. Comparación de características físicas entre terminales.

A día de hoy han pasado más de cuarenta años desde aquella llamada y la evolución tecnológica ha ido ligada al estilo de vida de los llamados países del primer mundo, donde el consumo ha ido creciendo y permitiendo así el desarrollo a pasos agigantados de la tecnología.

1.3. Introducción a la tecnología GPS

El GPS, cuyas siglas corresponden a su terminología en ingles “Global Positioning System”, es un sistema global de navegación por satélites (GNSS) que permite localizar la posición del receptor mediante la triangulación de como mínimo tres puntos de referencia, los satélites.

Esta idea surgió gracias al lanzamiento del primer satélite artificial de la historia, el Sputnik I (1957), cuyo seguimiento se realizaba mediante el estudio del efecto Doppler sobre la señal que transmitía. Por consecuencia e inversamente, conociendo la posición exacta del satélite se podría conocer la posición exacta de un observador en la superficie terrestre utilizando como mínimo tres satélites. Así, el primero en desarrollar, instalar y emplear esta idea fue el Departamento de Defensa de los Estados Unidos con TRANSIT.

TRANSIT estuvo operativo para su uso comercial en el año 1967. En este sistema, las actualizaciones de posición se realizaban cada 40 minutos y el usuario debía estar estático para que los datos fuesen exactos. Por ello se desarrolló otro sistema basado en PRN (Pseudo Random Noise) que proveía unos datos más precisos y se añadió a cada satélite un reloj atómico sincronizado desde el mismo punto para conseguir mayor exactitud en la localización de la ubicación. Así nació la actual constelación de satélites utilizadas para el GPS, NAVSTAR.

NAVSTAR consta de un total de 21 satélites más tres de apoyo orbitando a 20.200km de la tierra de manera sincronizada para cubrir toda la superficie planetaria.

El funcionamiento del posicionamiento del receptor sigue los siguientes pasos:

- ✧ El receptor detecta automáticamente como mínimo 3 satélites de la red.

- Recibe unas señales indicando la identificación y la hora de cada uno de los satélites.
- Se sincroniza con el reloj del satélite y calcula el tiempo que tarda en llegar la señal.
- En base al punto anterior calcula la distancia a los tres satélites.
- Un vez halladas las distancias, se determina la posición relativa respecto los satélites mediante triangulación.
- Y conociendo la ubicación real de cada uno de los satélites sobre el globo, se calcula la posición absoluta o coordenadas reales del receptor.

Actualmente las grandes potencias mundiales están desarrollando sus propios sistemas GPS, creando sus propias constelaciones de satélites. Los equivalentes al NAVSTAR de Estados Unidos son GLONASS para Rusia, Galileo para la Unión Europea y Beidou para China.

En cuanto a los receptores GPS, siguiendo la evolución tecnológica de las últimas décadas, han ido reduciendo el tamaño y la energía consumida, facilitando la tarea a los fabricantes de telefonía móvil de integrar estos chips en sus dispositivos. Además, a los desarrolladores de software se les abre un gran abanico de posibilidades creando aplicaciones que van desde las tradicionales de seguimiento punto a punto de la ubicación, hasta las que ofrecen LBS (Servicios Basados en la Localización).

1.4. SO más popular: Android

Para hacer uso de la tecnología GPS disponible en los terminales móviles se necesita un sistema operativo que haga posible la comunicación de los desarrolladores de software con el hardware del dispositivo. Actualmente existen cuatro sistemas operativos que abarcan casi la totalidad del mercado mundial de telefonía móvil: Android, iOS, WindowsPhone y BlackBerry.

Según IDC (International Data Corporation), una empresa de investigación de mercado, análisis y asesoramiento en tecnología de información, telecomunicaciones y tecnología de consumo, Android se ha consolidado con casi un 80% del total durante el año 2013. En la siguiente tabla se muestran los datos del último trimestre del año anterior:

Sistema Operativo	Android	iOS	Windows Phone	BlackBerry	Otros
Nº terminales (Millones)	211.6	33.8	9.5	4.5	1.7
Porcentaje (%)	**81.0**	12.9	3.6	1.7	0.6

Tabla 2. Estadísticas de SO móviles.

¿Por qué el sistema operativo Android es el más popular? Sencillamente es el más accesible y libre. Es decir, no hay que pagar ni para programar ni para incluirlo en el terminal. Característica que lo hace muy popular entre fabricantes y desarrolladores ya

que los costes de lanzar un teléfono o una aplicación son realmente bajos.

Al ser abierto, cualquiera puede bajarse el código fuente e inspeccionarlo, compilarlo y modificarlo si es necesario, propiedad que agradecen los fabricantes porque pueden adaptar mejor el sistema operativo a los terminales. Además aporta cierta seguridad a los usuarios ya que los errores se solucionan rápidamente.

Por las características anteriormente citadas en este apartado, el desarrollo de la aplicación de seguimiento GPS se realizará bajo el sistema operativo Android.

2. Android

2.1. Cronología

Justo en el momento en que Android salió a la luz, los sistemas operativos reinantes eran Symbian e iOS. Como sucede con todos los nuevos productos, no es nada fácil hacerse hueco entre otros ya establecidos desde hace meses e incluso años, por ello el nuevo sistema debería ofrecer alguna particularidad muy especial para que los fabricantes se decantaran por él. Y ésta fue que Android se presentó bajo la licencia de código abierto Apache 2.0, la cual permitía una modificación del código y su utilización gratuitamente.

Este concepto y la plataforma fue idea de Android Inc., una empresa pequeña de Palo Alto, California, que fue adquirida por Google en 2005. Su finalidad era proporcionar a los fabricantes y a las operadoras telefónicas un sistema operativo ligero, flexible, estable y fácil de actualizar.

La plataforma fue presentada en noviembre de 2007 coincidiendo con el anuncio de la agrupación Open Handset Alliance, un grupo de empresas con el objetivo de promover estándares abiertos para plataformas de dispositivos móviles. Y casi un año después, el 22 de octubre de 2008 salió al mercado de la mano de HTC con su HTC Magic, el primer *smartphone* con Android.

Imagen 2. HTC Magic

El 11 de septiembre de 2012 Android se convierte en el sistema operativo más popular con 500 millones de activaciones. Es obvia la apuesta de los fabricantes por este sistema y la multitud de opciones que barren la gama baja y alta haciendo mella

en sus competidores, especialmente en iOS ya que Symbian está casi desaparecido debido al declive de Nokia.

2.2. Plataforma Android

Android es un sistema operativo inicialmente creado para usarlo en dispositivos móviles pero que se ha extendido a tabletas y a portátiles con pocos recursos. Está compuesto por un núcleo (el kernel) basado en Linux, un sistema operativo libre, gratuito y multiplataforma, numerosas librerías de código, aplicaciones frameworks, soporte multimedia, una interfaz gráfica completa y mucho más. Mientras el código del núcleo está desarrollado en el lenguaje de programación C++, el sistema permite programar aplicaciones en una variación de la plataforma JAVA llamada Dalvik, en ella se utiliza el lenguaje de programación java con la ayuda del lenguaje xml para definir ciertos aspectos gráficos.

Para desarrollar aplicaciones Android es necesario descargarse el paquete SDK (Software Development Kit) desde la página de desarrolladores developer.android.com/sdk. Este paquete incluye las API's y herramientas necesarias para desarrollar y testear el código en java. El SDK de Android debe utilizarse conjuntamente con un IDE (Integrated Development Environment). Los más utilizados son Eclipse de Oracle y NetBeans de SunMicrosystems.

2.3. El papel de Google

Realmente lo que se entiende por sistema operativo es el conjunto de todo el software que hacen útiles y aprovechables por parte del usuario los servicios físicos (hardware) de una computadora. Con la evolución de la tecnología, los teléfonos móviles se han convertido ya en ello y han pasado a llamarse *smartphones*.

Haciendo referencia a Android, se tiene una primera capa, como base, del SO denominada Android Open Source Project (AOSP), a la que se sumarían los Google Mobile Services (Google Play, Google Maps, Google Now, etc.) y las Google Apps (Gmail, Google Drive, etc.).

La base de Android, el citado AOSP, es un proyecto en el que colabora la comunidad de usuarios y que ofrece una alternativa al sistema Android como se entiende a día de hoy ya que ofrece algunas aplicaciones nativas para realizar llamadas, fotos, reproducir

música, etc., y ser autosuficiente dentro de un terminal. Es la parte del sistema Android que le da la fama de ser libre y abierto. Google es la principal responsable de su gestión y desarrollo.

En cuanto a los servicios y aplicaciones que ofrece Google son totalmente cerradas y propietarias. Cualquier fabricante puede tener un teléfono completamente funcional con el AOSP, pero si quiere utilizar cualquier servicio o aplicación que dan acceso a unas prestaciones muy demandadas por los usuarios (Google Play Services, Gmail, Google Maps, son algunos ejemplos de los más conocidos) tendrá que asumir los términos y requisitos de Google, además de pasar una validación técnica.

Como conclusión se tiene que la plataforma Android es libre, pero las distintas capas superficiales que la componen no. La extensión del uso de éstas hacen que sean imprescindibles para los usuarios y para los fabricantes otorgando así a Google una posición privilegiada en el mercado.

2.4. Arquitectura

Para empezar a desarrollar aplicaciones en Android es importante conocer cómo está estructurado el sistema, es decir, conocer su arquitectura. Android está compuesto por diversas capas conectadas entre ellas a través de librerías que permite al programador hacer aplicaciones sin tener que escribir código en un lenguaje bajo de programación. A este tipo de estructura también se le conoce como pila.

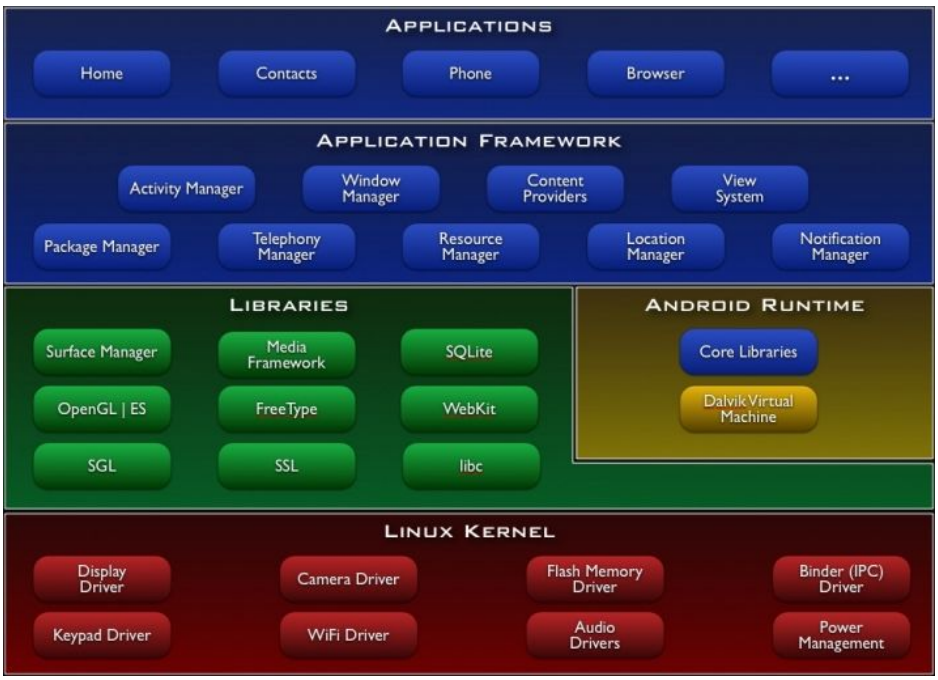

Imagen 3. Pila de Android.

Explicación de cada una de las partes representadas en la imagen anterior:

- El **núcleo** (Linux kernel). El núcleo de Android está basado en el kernel de Linux versión 2.6, como el que pueda llevar cualquiera de las distintas distribuciones, por ejemplo Ubuntu. La única diferencia radica en que se construye para un tipo de máquinas menos potentes, por lo tanto debe ser adaptado para que sea soportado.

  El núcleo actúa como una capa de abstracción entre el hardware y el resto de capas de la pila. El desarrollador no puede acceder directamente a éste, sino que debe utilizar los recursos disponibles en la librerías. Para cada elemento de hardware existe un controlador (*driver*) que permite acceder a él mediante software.

  El kernel también se encarga de gestionar los distintos recursos del teléfono y del sistema operativo: energía, memoria, procesos, servicios de comunicación, etc.

- **Librerías**. La capa justo por encima del núcleo la componen las bibliotecas nativas de Android. Están escritas en C++ y compiladas para la arquitectura específica del teléfono. El objetivo de dichas es proporcionar funcionalidad a las aplicaciones donde se realizan tareas que se repiten con frecuencia, aligerando de esta manera el código y reduciendo notablemente el tiempo de desarrollo de aplicaciones. Las librerías incluidas habitualmente son:

  - La tecnología de navegación de ***WebKit***, el mismo motor de código del navegador Safari de Mac. WebKit se ha convertido en el estándar de la mayoría de las plataformas móviles.
  - Soporte para acceso a base de datos ***SQLite***, una base de datos liviana y fácil de usar.
  - Soporte de gráficos avanzados, incluyendo 2D, 3D, motor gráfico SGL y ***OpenGL_ES***.
  - Soporte de audio y vídeo de ***OpenCORE*** de PacketVideo.
  - Protocolo ***SSL*** (Secure Sockets Layer) de Apache.

- **Entorno de ejecución**. Esta parte no es en sí una capa, como se observa en el diagrama. El entorno de ejecución contiene las librerías con funcionalidades java y alguna específica para Android.

  El principal componente de este apartado es la máquina virtual ***Dalvik***. Las aplicaciones se escriben bajo java y son compiladas en un formato específico para poder ser ejecutadas bajo esta máquina virtual. La ventaja de esto es que las aplicaciones se compilan únicamente una vez y ya pueden distribuirse para ejecutarse bajo cualquier dispositivo Android con los requerimientos mínimos.

- **Framework** de aplicaciones. Esta capa está formada por las clases y servicios que necesitan las aplicaciones para realizar sus funciones. La mayoría de los componentes de esta capa son librerías java que acceden a las capas inferiores del sistema. En el diagrama se observan distintas librerías con el correspondiente nombre que las describen.

- **Aplicaciones**. La última capa incluye todas las aplicaciones del dispositivo, tanto las que tienen interfaz gráfica como las que no, las nativas (C++) y las administradas (java), las que viene preinstaladas y las del usuario.

  En esta capa se encuentra la aplicación que se encarga de manejar las distintas pantallas y de lanzar las demás aplicaciones, el *Launcher*.

2.5. Conceptos esenciales

Una vez conocidas las partes del sistema, es hora de analizar los distintos componentes de una aplicación que ayudan a entender su funcionamiento. Se destacan cuatro:

- **Activity**: Representa una pantalla independiente con su propia interfaz gráfica. La aplicación se compondrá de una o múltiples pantallas y deberán tratarse como independientes cuando se programe. Si es necesario se podrá pasar información entre ellas como se explicará más adelante. Desde el momento que se crea una pantalla (actividad) hasta que se destruye pasa por los distintas etapas del ciclo de vida de una actividad.

  A la hora de programar se instancia como una clase que hereda del objeto Activity, uno de los principales en la plataforma Android.

- **Service**: Son componentes sin interfaz gráfica que permiten realizar tareas duraderas en segundo plano. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar notificaciones, etc. Pueden ser de dos tipos:
  - ***Started***: Son iniciados por algún componente y su ejecución es independiente a él. Puede que siga ejecutándose una vez que el componente ya no exista.
  - ***Bound***: Son iniciados cuando algún componente se vincula a este servicio, comportándose como una especie de cliente-servidor. Una vez que el componente que lo ha llamado desaparece este servicio termina.

- **Content Provider**: Mecanismo que define Android para acceder a datos estructurados. Además de encapsular los datos, mediante estos componentes es posible compartirlos entre distintas aplicaciones. Es necesario definir la seguridad y el nivel de acceso a ellos. Un claro ejemplo son los contactos en un teléfono móvil.

- **Broadcast Receiver**: Se trata del componente que permite que se reciban mensajes o eventos generados por el sistema o por otras aplicaciones. Ejemplos claros de ellos son los avisos de batería baja, las llamadas entrantes...

Uno de los puntos más destacables del diseño de Android es que una aplicación puede acceder a un componente desarrollado en otra aplicación. Un claro ejemplo de ello es el uso de la cámara, la cual puede utilizarse creando una nueva actividad para ello o

abrir el componente que viene instalado por defecto en el sistema operativo. En estos casos se utilizaría un **Intent**. En el siguiente esquema se representa su funcionalidad:

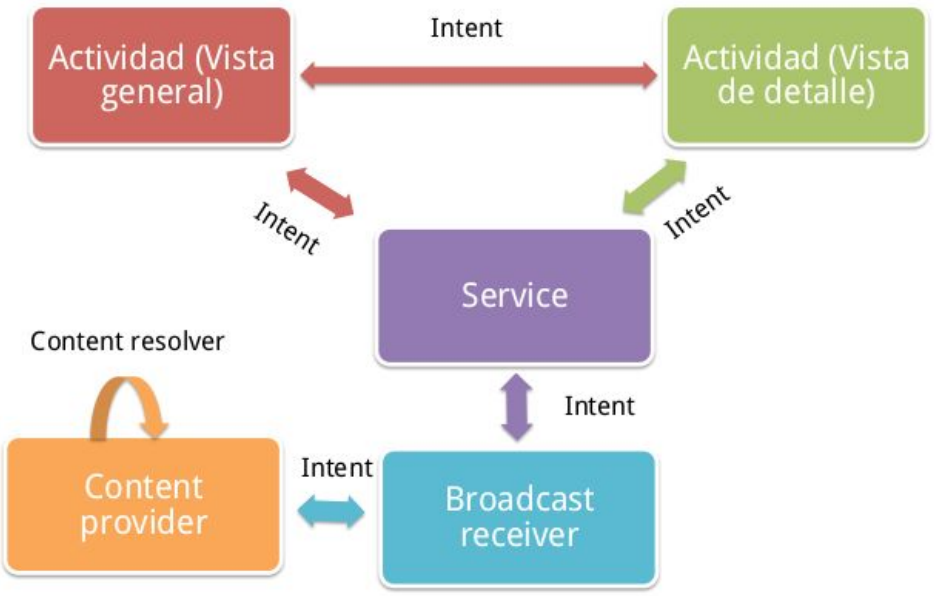

Imagen 4. Activando componentes.

La accesibilidad a estos componentes por parte de la aplicación se define en un archivo llamado ***AndroidManifest.xml*** que se encargará de comunicarle al sistema operativo los componentes disponibles, los permisos necesarios (cámara, gps, etc.), la versión de Android mínima, el hardware y software requerido, las librerías externas que utiliza (como Google Maps) y otras muchas características.

2.6. Ciclos de vida

Android se creó para dispositivos móviles y aunque actualmente se haya extendido a otros dispositivos con recursos limitados sus principales funcionalidades son las de comunicación. Por lo tanto el usuario puede necesitar atender a un mensaje de texto o a una llamada telefónica sin tener que esperar a cerrar la aplicación. El ciclo de vida de una actividad está orientada a poder realizar las acciones anteriores sin perder información o datos en la aplicación, y estará condicionado por la interacción del usuario y cualquier evento que pueda surgir del sistema.

Cuando se ejecuta una aplicación en un ordenador personal, es el usuario quién decide cuándo finalizarla. En el caso de Android es el propio sistema el que gestiona los

recursos y decide si finalizar la aplicación. Sin embargo, el usuario espera que ésta se encuentre en el mismo estado que estaba antes de abandonarla, por lo que surge la necesidad de crear un ciclo de vida donde realizar las distintas tareas de almacenamiento y control de la actividad.

Una actividad puede estar en cuatro estados:

- **Activa** (*Running*): La actividad se encuentra encima de la pila, lo que quiere decir que es visible y tiene el foco.

- **Visible** (*Paused*): La actividad se encuentra visible pero no tiene el foco. Esto sucede cuando otra actividad que ocupa parcialmente la pantalla está activa. Si la actividad activa ocupara completamente la pantalla la anterior estaría parada.

- **Parada** (*Stopped*): Cuando la actividad no es visible. El programador debe guardar el estado de la interfaz de usuario, preferencias, datos, etc. si fuera necesario.

- **Destruida** (*Destroyed*): Cuando se invoca el método *finish()* o el sistema requiera de recursos y necesite finalizarla.

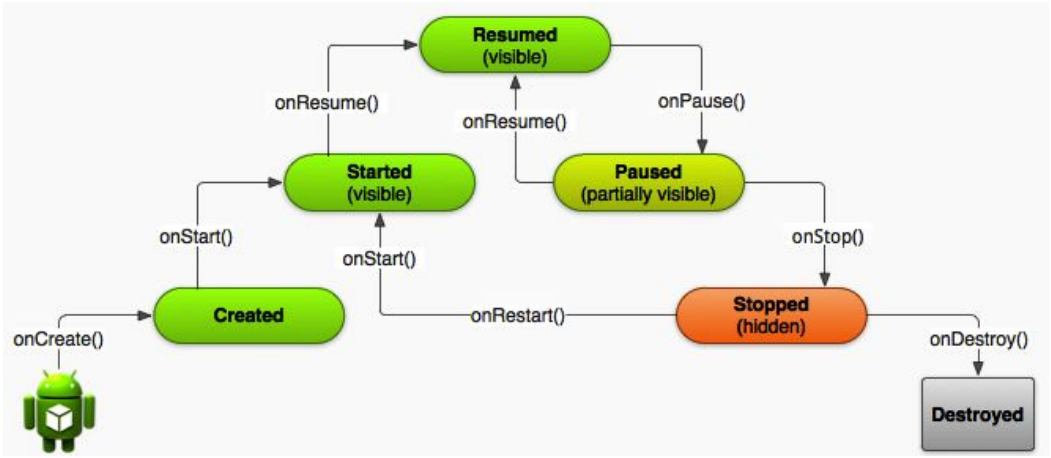

Imagen 5. Ilustración simple del ciclo de vida de una actividad.

Cada vez que una actividad cambia de estado se crean unos eventos que pueden ser capturados por ciertos métodos que heredan de la clase *Activity*. A continuación se muestra otro diagrama con más detalles respecto los métodos citados y se hace una breve descripción de cada uno.

- **onCreate()**: Se llama en la creación de la actividad y se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario y los respectivos elementos que contiene o la declaración del uso de base de datos. Puede recibir información de estado de la actividad (en una instancia de la clase *Bundle*).

- **onStart()**: Nos indica que la actividad está a punto de ser colocada en primer plano.

- **onResume()**: Se llama cuando la actividad va a interactuar con el usuario. Es un buen momento para lanzar música o animaciones.

- **onPause()**: Se llama justo antes de que la actividad vaya a pasar a un segundo plano. Este método es un buen lugar para almacenar datos que se están editando en la actividad principal.

- **onStop()**: La actividad ya no va a ser visible por el usuario. Cuando se programe hay que tener en cuenta que el sistema puede destruir la actividad antes de pasar por este método por la necesidad de recursos.

- **onRestart()**: La actividad se va a volver a representar después de haber pasado por *onStop()*.

- **onDestroy()**: Se llama antes de que la actividad sea totalmente destruida cuando, por ejemplo, se pulsa el botón de "Atrás" del sistema o se programe un *finish()*.

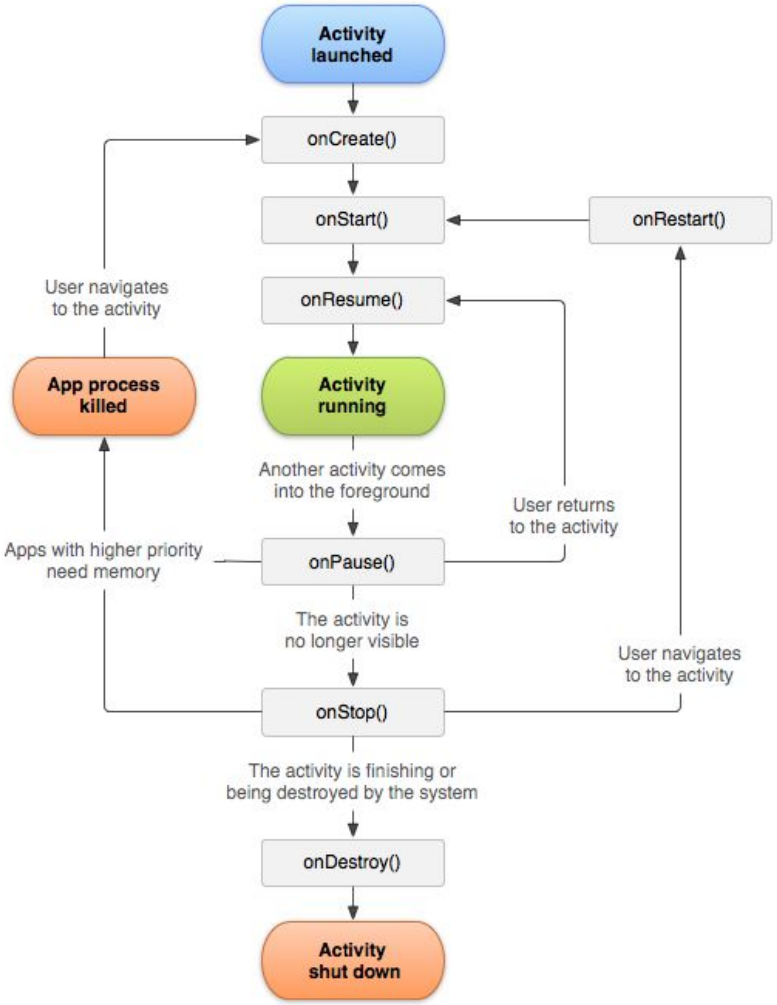

Imagen 6. Diagrama de ciclo de vida de una actividad con métodos.

2.7. Versiones

Una versión de Android se distingue por tres características:

- El **nombre**: El nombre comercial es por lo que normalmente se identifica una versión, y para ello se han elegido postres en orden alfabético (Cupcake, Donut, etc.).
- La **versión**: Son los números enteros separados por puntos que se colocan detrás del nombre comercial. También pueden ser precedidos por una "v" de la palabra versión. Una vez que la versión ha sido lanzada, pueden haber modificaciones o añadiduras. Se utilizan estos números para diferenciarlos entre sí.
- Nivel **API**: Es una librería que contiene un conjunto básico de paquetes y clases, de elementos y atributos XML para la declaración y acceso a recursos, de Intents, de permisos que las aplicaciones puedan solicitar, etc. Cada vez que surge una modificación de importancia se le debe cambiar el número por el cual está definida.

Antes de empezar a desarrollar un proyecto en Android es necesario tener claro y especificar sobre qué versión del sistema va a correr la aplicación. Es importante resaltar que hay clases y métodos disponibles a partir de una cierta versión y ésta debe ser declarada en el AndroidManifest.xml.

Cuando se lanza una nueva plataforma actualizada siempre mantiene la compatibilidad con las versiones antiguas. Añaden nuevas funcionalidades y en el caso de modificar alguna, únicamente se marca como obsoleta pero no se elimina. La ventaja de este procedimiento es que cuando una aplicación lleva mucho tiempo programada no será necesario modificar ningún aspecto para que siga funcionando en terminales y versiones nuevas.

A continuación se muestra una tabla con el historial de versiones de Android:

Versión	API	Nombre
Android 4.4	19	KitKat
Android 4.3	18	JellyBean MR2
Android 4.2, 4.2.2	17	Jelly Bean MR1
Android 4.1, 4.1.1	16	JellyBean
Android 4.0.3, 4.0.4	15	IceCreamSandwich MR1
Android 4.0, 4.0.1, 4.0.2	14	IceCreamSandwich
Android 3.2	13	HoneyComb MR2
Android 3.1.x	12	HoneyComb MR1
Android 3.0.x	11	HoneyComb
Android 2.3.4, 2.3.3	10	Gingerbread MR1
Android 2.3.2, 2.3.1, 2.3	9	GingerBread
Android 2.2.x	8	Froyo
Android 2.1.x	7	Eclair MR1

Android 2.0.1	6	Eclair 0.1
Android 2.0	5	Eclair
Android 1.6	4	Donut
Android 1.5	3	Cupcake
Android 1.1	2	Base 1.1
Android 1.0	1	Base

Tabla 3. Versiones Android.

3. Requisitos mínimos

3.1. Hardware

En cuanto a los requisitos físicos que deberá tener el terminal móvil para que la aplicación que se desarrolla en los siguientes apartados se ejecute con fluidez y sin problemas serán los siguientes:

- Terminal móvil inteligente. Smartphone, que soporte Android 4.0.
- RAM. Con memoria volátil de 512MB.
- CPU. Procesador de un núcleo de 1Ghz como mínimo.
- GPU. Procesador gráfico que soporte aceleración 2D y 3D.
- Capacidad de almacenamiento para la app de 5MB más caché.
- Mínima pantalla de 4”.
- Conectividad a la red de datos.
- Chip receptor GPS.

Esta aplicación se desarrollará y probará con un Motorola *Moto G*. La elección de realizarlo sobre un terminal real y no utilizar los recursos para emular el sistema se toma debido a la lentitud de estos últimos. Se necesitaría un ordenador potente para soportar la gran cantidad de RAM y CPU que consumen los emuladores de Android.

3.2. Software

Teniendo como versión mínima de Android la 4.0, los requerimientos mínimos de software serán los siguiente.

- **API** mínima versión 14, quiere decir que la aplicación únicamente funcionará con garantía desde IceCream hasta KitKat (a partir de Android 4.0). Si surgieran nuevas versiones, según lo expuesto en el apartado 2.7, la app seguiría siendo apta para éstas.
- **Google Play Service**. Los servicios de Google Play son esenciales para poder hacer uso de alguno de sus servicios.
- **Google Maps**. Servicio para utilizar los mapas y representar el usuario sobre ellos.

4. Diseño

4.1. Funcionalidad

La facilidad de conseguir un *smartphone*, de instalar multitud de aplicaciones completamente gratuitas y la comodidad del tamaño y peso de los aparatos citados, hacen de ellos unos perfectos compañeros. Por ello esta aplicación se ha desarrollado con la finalidad de hacer un registro de cualquier desplazamiento, pero en especial a las actividades deportivas como el ciclismo y el senderismo.

Como toda buena aplicación para hacer un seguimiento y registro de la actividad debe disponer las siguientes funcionalidades.

- ✓ Mostrar un mapa y la posición del usuario sobre él.
- ✓ Hacer un seguimiento de la posición y dibujar una linea por su recorrido.
- ✓ Mostrar los datos más relevantes (velocidad, tiempo de actividad y distancia).
- ✓ Almacenar el recorrido.
- ✓ Poder acceder a los distintos recorridos a través de una lista (historial) y visualizarlos.
- ✓ Insertar datos físicos del usuario.
- ✓ Disponer una página de inicio que haga un resumen de la totalidad de las actividades.

Siguiendo los puntos anteriores se crea un diagrama en el que se puede observar a grandes rasgos el comportamiento de la aplicación. En una primera instancia, al abrir la aplicación, se accede a una página inicial que alberga un resumen total de las actividades, el historial de las mismas y un acceso a las funciones de grabación y visualización. A través del acceso de visualización, dependiendo de la entrada del historial que se seleccione deberá mostrar la ruta ligada a ésta. Para el acceso a la grabación, únicamente se implementará un botón que cambie de pantalla. Una vez en la pantalla de “actividad en curso” existen dos salidas; la primera, volver a la pantalla inicial cancelando la grabación, y la segunda pulsar un botón para parar la grabación y acceder a la pantalla de “actividad detenida”. Desde esta pantalla también existen dos posibilidades; como anteriormente ir atrás, que en este caso volvería a la pantalla anterior siguiendo con la grabación, o pulsar un botón que almacene la ruta y nos dirija a la pantalla de inicio.

A continuación se muestra el diagrama:

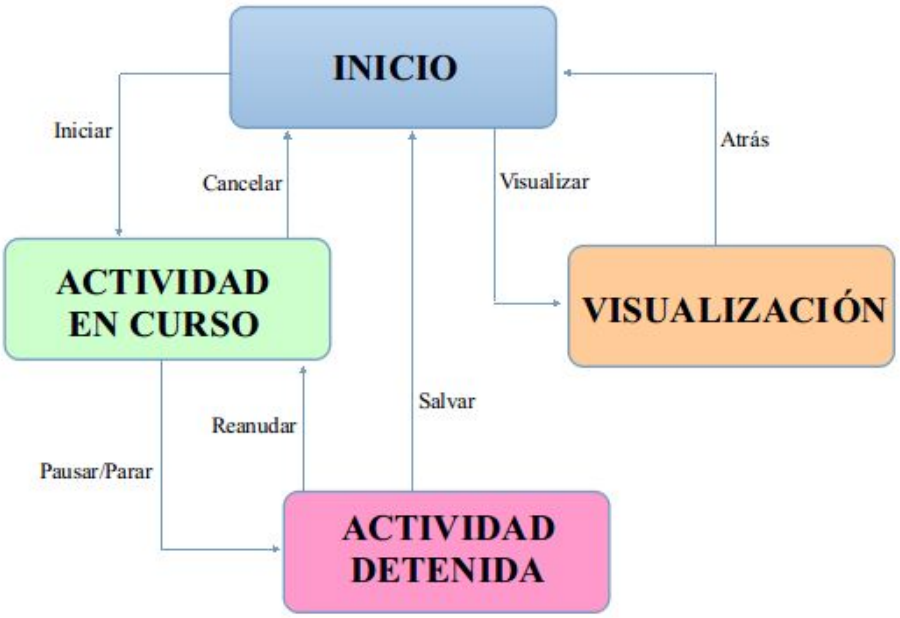

Imagen 7. Diagrama funcional.

4.2. Interfaz gráfica

Tomadas las funcionalidades del apartado anterior, se crea una lista de las pantallas necesarias para cumplir los requisitos del diseño funcional, y éstas son:

- **Inicio**: Recopilación de las estadísticas más relevantes de las distintas actividades.
- **Historial**: Una lista del registro de las actividades realizadas donde se pueda seleccionar cada una de ellas y visualizarlas o eliminarlas.
- **Visualizar recorrido mapa**: Pantalla en la que mostrarán los recorridos salvados en el mapa.
- **Visualizar recorrido datos**: Pantalla en la que mostrarán los datos de los recorridos salvados.
- **Datos del usuario**: Pantalla en la cual el usuario puede introducir algunos de sus datos personales (Nombre, fecha de nacimiento, altura, peso, etc.).
- **Explorar el mapa**: Pantalla no necesaria pero que permite al usuario antes de

empezar la actividad realizar un estudio de la zona y planificar un recorrido sobre el mapa.

- **Acerca de**: En ésta se muestra cierta información sobre la versión de la aplicación y el programador.

- **Actividad en curso**: Pantalla presente cuando se está realizando la actividad, se mostrará el mapa y la linea del recorrido además de algunos datos instantáneos de la misma.

- **Actividad detenida**: Mostrará un pequeño resumen de la actividad y permitirá reanudarla o salvarla, en este último caso volvería a la pantalla de inicio.

Una vez definidas las pantallas y con el apoyo de un diagrama se muestra el funcionamiento de éstas entre sí:

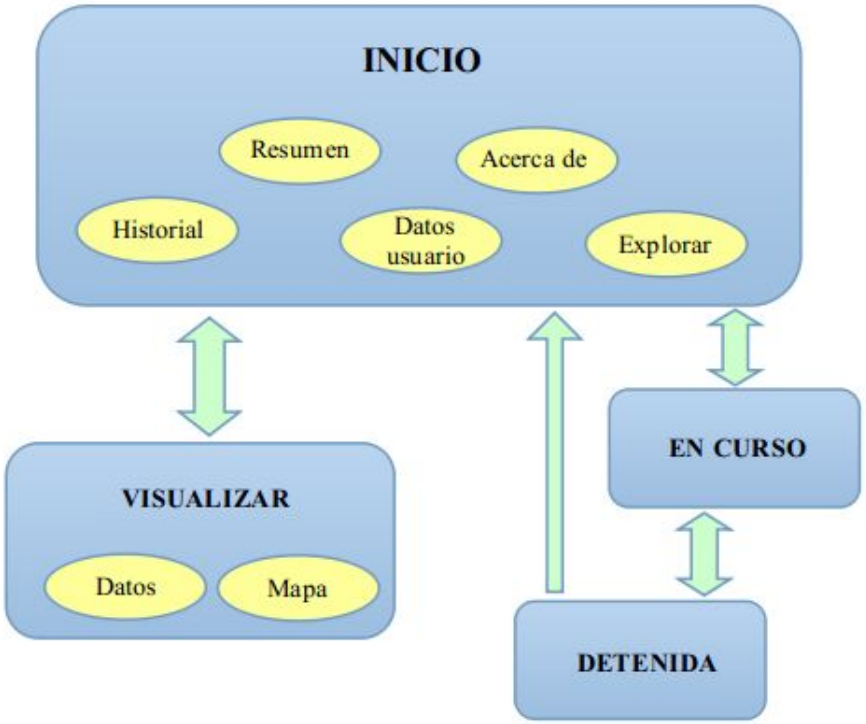

Imagen 8. Diagrama de comportamiento de la interfaz.

Siguiendo el diagrama funcional, y para facilitar el uso de la aplicación aprovechando los recursos gráficos de Android, se ha decidido implementar en la página de inicio un "*Navigation Drawer*" (Cajón de navegación) para encapsular los apartados de *Inicio*, *Historial*, *Datos del usuario*, *Explorar el mapa* y *Acerca de*. Se muestra a continuación un boceto de lo expuesto.

Imagen 9. Pantalla con cajón de navegación - diseño.

La pantalla de inicio mostrará un pequeño resumen global de las estadísticas de todas las actividades más un botón para iniciar el recorrido.

Imagen 10. Pantalla inicial - diseño.

La pantalla de historial muestra una lista con el registro de las actividades realizadas. Cuando se pulse sobre una de las entradas se mostrarán los datos y recorrido de la misma.

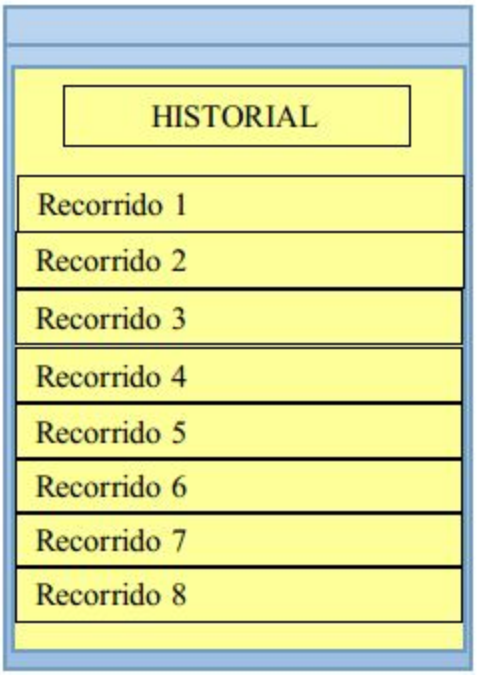

Imagen 11. Pantalla historial - diseño.

Imagen 12. Pantalla visualizar datos - diseño.

Imagen 13. Pantalla visualizar mapa - diseño.

Las pantallas restantes pertenecientes al módulo inicial tendrán las siguientes estructuras:

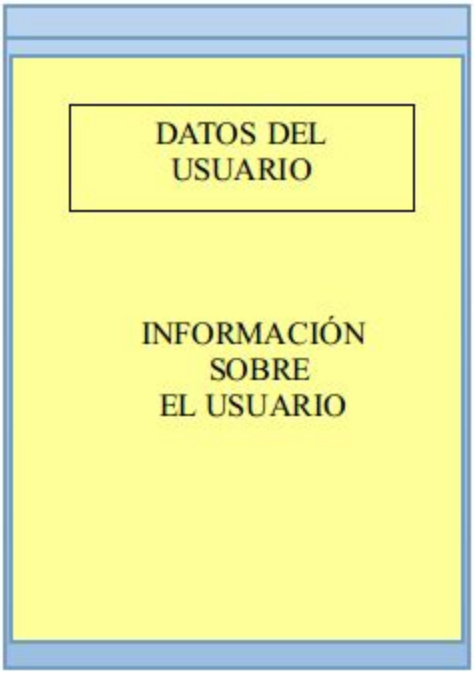

Imagen 14. Pantalla datos usuario - diseño.

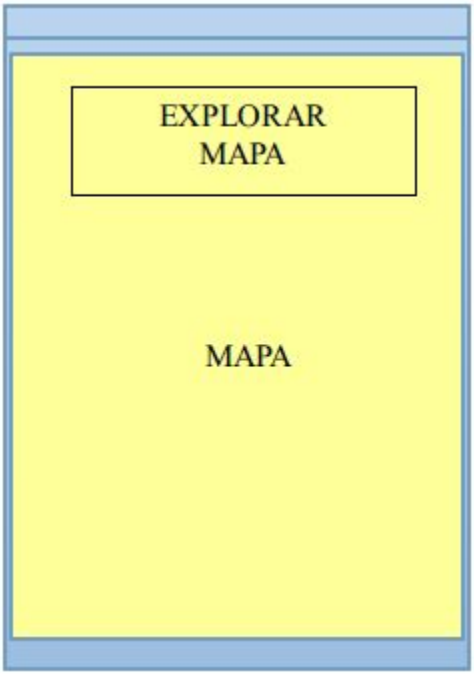

Imagen 15. Pantalla explorar mapa - diseño.

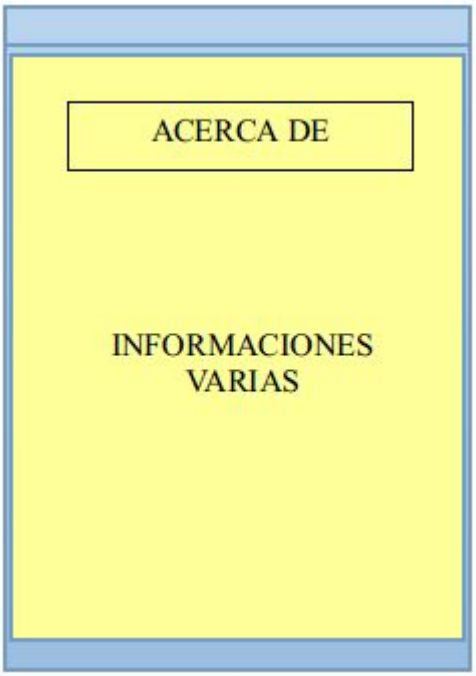

Imagen 16. Pantalla acerca de - diseño.

Para finalizar con los bocetos de las pantallas a continuación se muestran las correspondientes a la parte "actividad en curso" y "actividad detenida". Cada una tendrá los botones necesarios para cumplir con lo expuesto en la parte del diseño del funcionamiento.

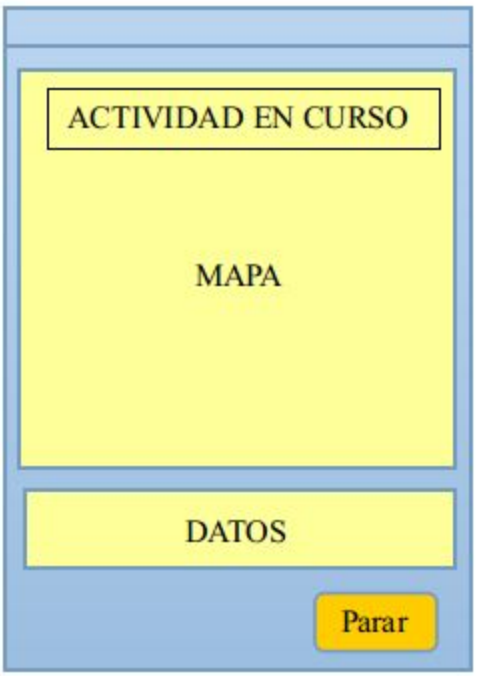

Imagen 17. Pantalla actividad en curso - diseño.

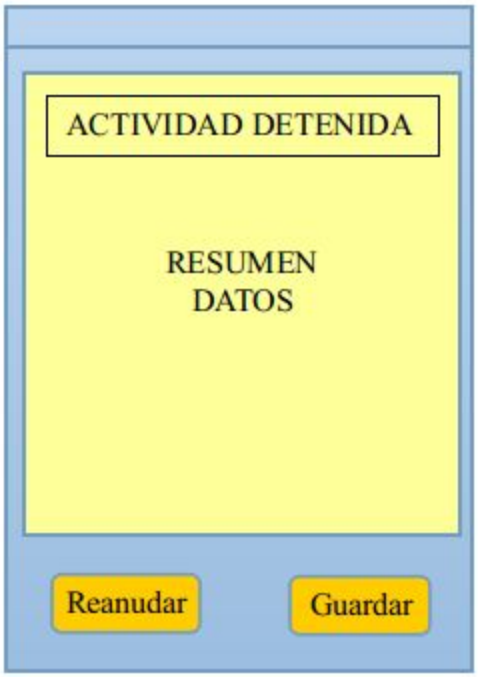

Imagen 18. Pantalla actividad detenida - diseño.

4.3. Base de datos

La base da datos estará formada por dos tablas dentro de un mismo archivo, una para guardar las posiciones de latitud y longitud y otra para guardar los datos finales de cada actividad. Los límites de datos en SQLite (http://www.sqlite.org/limits.html) son:

- Puede contener hasta 64 tablas.
- Cada tabla por defecto puede albergar hasta 2000 columnas, pero puede ampliarse a 32767 en caso de ser necesario.
- Las filas por tabla son como máximo de 2^{64} = 18.446.744.073.709.551.616.

Como se observa es muy difícil superar estos valores en una aplicación como la que se está desarrollando en este proyecto.

La primera de las tablas se llamará "*Routes*" y contendrá los siguientes campos:

Campo	Tipo	Observaciones
Identificador	Integer	Clave primaria. Número entero que contendrá el valor del registro.
Tipo de actividad	String	Podrá ser senderismo, correr, ciclismo, ciclismo de montaña, coche u otros.
Fecha inicial	DateTime	Fecha en formato dd/mm/yy HH:mm:ss en la se inicia la actividad.
Fecha final	DateTime	Fecha en formato dd/mm/yy HH:mm:ss en la se finaliza la actividad.
Tiempo total	Long	Tiempo en segundos que dura la actividad. Se modificará el formato para mostrarlo por pantalla.
Distancia	Long	Distancia que se ha recorrido.
Velocidad media	Long	Velocidad media del recorrido.

Tabla 4. Tabla "Routes" de la base de datos.

La segunda tabla se llamará "*Locations*" y contendrá los siguientes campos:

Campo	Tipo	Observaciones
Identificador	Integer	Clave primaria. Número entero que contendrá el valor del registro.
Identificador de ruta	Integer	Clave que relaciona este registro con el de la tabla "Routes".
Fecha	DateTime	Fecha en formato dd/mm/yy HH:mm:ss cuando se salva el dato.
Latitud	String	Latitud de la posición.
Longitud	String	Longitud de la posición.

Tabla 5. Tabla "Locations" de la base de datos.

5. Implementación

5.1. Entorno de desarrollo (IDE)

Para programar una aplicación Android es necesario un PC que tenga instalado un entorno de desarrollo integrado (IDE) para dicho sistema y añadir los distintos módulos y paquetes.

El ordenador personal sobre el que se trabajará es un portátil Sony Vaio FZ21S con las siguientes características:

- Pantalla de 15,6”
- CPU de doble núcleo (Core2Duo) a 2’0GHz
- Memoria RAM 2GB
- Sistema operativo linux Ubuntu 13.10

En cuanto al IDE (sigla en inglés de *Integrated Development Environment*), se trata de un programa informático compuesto por un conjunto de herramientas de programación que puede dedicarse a uno o varios lenguajes. Las herramientas principales que lo componen son un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).

5.1.1. Eclipse

Los IDE’s más utilizados son Eclipse y NetBeans, como ya se introdujo en el apartado de Android, y en este proyecto se empleará Eclipse por la simplicidad a la hora de configurar gracias a que Google a puesto a disposición de todos un paquete que incluye todo lo necesario para empezar a desarrollar, el llamado **Android SDK ADT Bundle**.

Anteriormente a la aparición de este paquete, los programadores tenían que perder como mínimo tres horas para poner a punto el entorno. Se debía descargar e instalar el JDK de Java, el SDK de Android, Eclipse, el plugin de Android (ADT) y los Platforms-Tools.

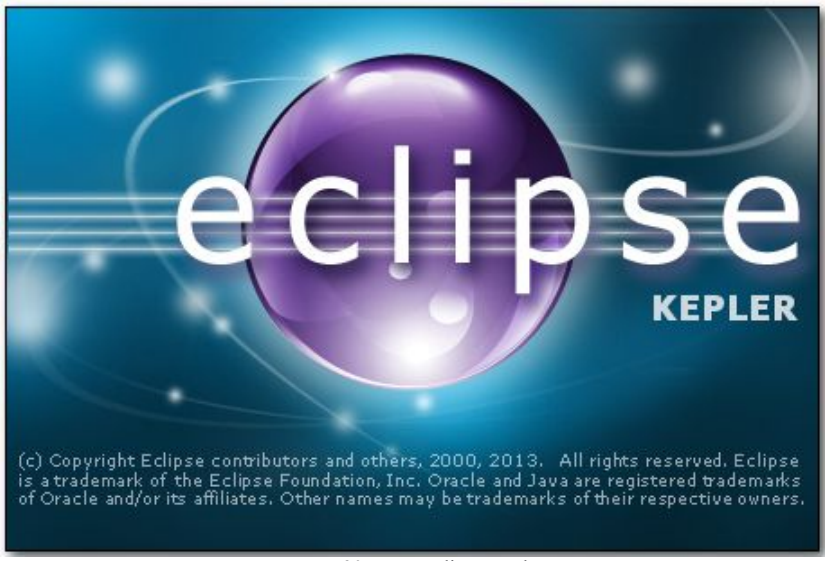

Imagen 20. Logo Eclipse Kepler.

El nuevo *Android SDK ADT Bundle* proporciona todas las bibliotecas API (hasta la 19) y las herramientas de desarrollo necesarias para crear, probar y depurar aplicaciones. Incluye los componentes esenciales de Android SDK y la última versión del IDE Eclipse (Kepler) con el ADT integrado, además viene preparado para simular dispositivos virtuales y realizar las pruebas sobre un terminal ficticio (emulador).

Imagen 19. Android Developer Tools.

5.1.2. Puesta a punto

A continuación se detallan los pasos a seguir para preparar el ordenador personal para la realización de la aplicación bajo Android.

1. Se necesita tener instalada una versión de **JDK** (*Java Development Kit*) y es aconsejable tenerla actualizada a la última versión, así que se procederá a comprobarlo en la página de Oracle (http://www.oracle.com/technetwork/es/java/javasebusiness/downloads/index.html). En este pc se trabajará con la versión 1.7.0_55 (versión 7 actualización 55).

2. Descargar e instalar Android SDK **ADT Bundle** desde la página de *Android Developers* http://developer.android.com/intl/es/sdk/index.html?hl=sk. Con ello se obtiene el SDK (*software Development Kit*) de Android, la versión Kepler de Eclipse y lo expuesto en el punto anterior. Se trabajará con la versión adt-bundle-linux-x86-20140321.

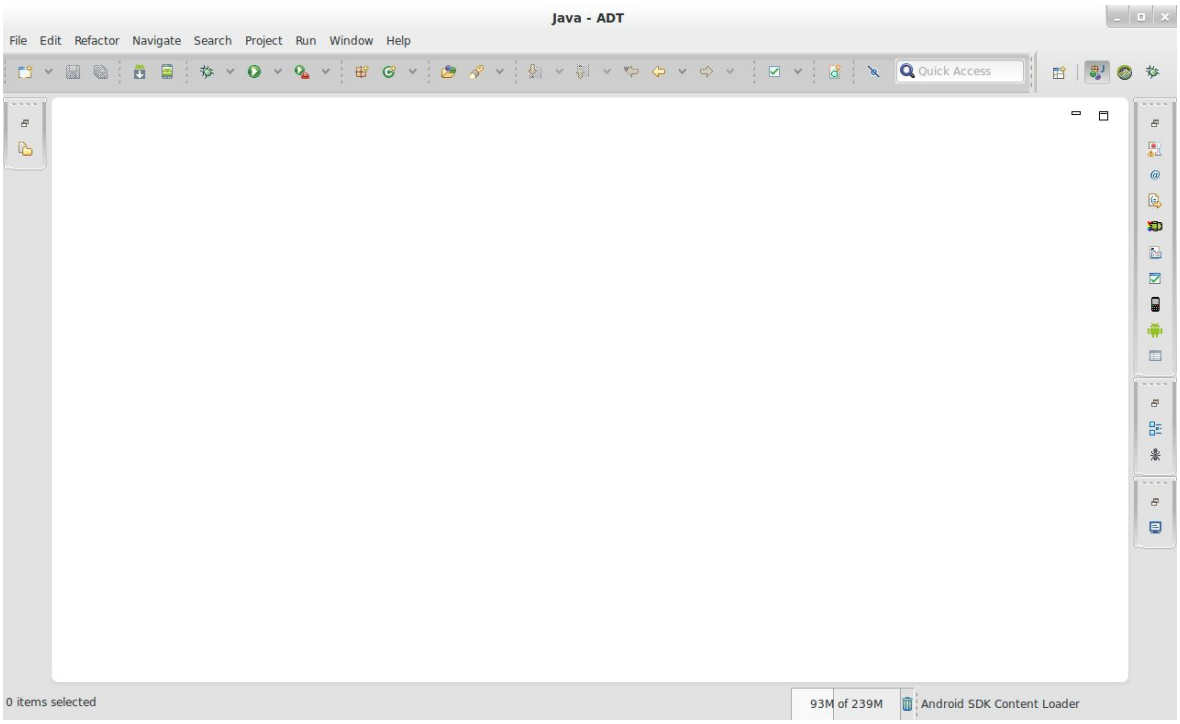

Imagen 20. IDE Eclipse.

3. Otra herramienta imprescindible para conectar el ordenador con los terminales es **ADB** (*Android Debug Bridge*) que permite multitud de acciones, como por ejemplo dar permisos (en caso de no tener el teléfono rooteado) a los archivos, copiar o mover carpetas, transferir del pc al móvil o al revés, etc. Se instala en el sistema Linux Ubuntu con el siguiente comando: *apt-get install android-tools-adb*. Para comprobar que se conectan ejecutar el siguiente código

en la terminal del sistema: *adb devices* en el caso de esta computadora y el terminal moto G se obtendría el siguiente resultado:

```
sauirqui@sauirqui-VGN-FZ21S:~$ adb devices
List of devices attached
TA8830E25N      device
```

Imagen 21. Conexión de móvil con pc.

4. En cuanto al móvil se debe habilitar la opción de poder instalar aplicaciones con orígenes desconocidos, para ello se selecciona "*Orígenes desconocidos*" situado en el apartado *Administración del dispositivo* en *Ajustes > Seguridad.*

5. También es necesario activar el modo desarrollador. Estas opciones están ocultas en las versiones nuevas de Android y para hacerlas visibles se debe acceder a *Ajustes>Acerca del teléfono* y hacer clic 7 veces en *Número de compilación*. Una vez hecho esto se vuelve a *Ajustes* y se accede a la nueva opción *Opciones de desarrollo* activando en el apartado de *Depuración* la opción *Depuración USB*.

Imagen 22. Opciones de desarrollo.

5.2. Estructura de un proyecto en Android

Cuando se crea un proyecto en Eclipse para Android, éste debe contener una serie de carpetas que clasifiquen los distintos elementos que puedan existir, como por ejemplo las librerías, el código java, el código XML de la interfaz gráfica, etc. A continuación se muestra una imagen de la estructura y se realiza una breve descripción de cada componente:

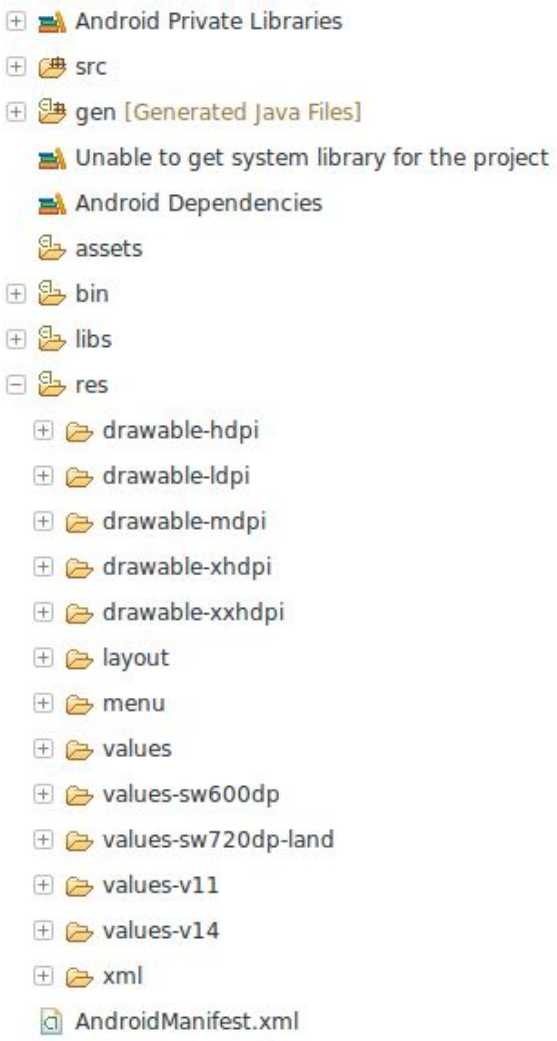

Imagen 23. Estructura de un proyecto Android

- Carpeta **/src/**

Esta carpeta contendrá el código fuente Java que se vaya escribiendo de nuestra aplicación, así como las clases que se creen. Se puede dividir en paquetes para tener una mejor estructura y encontrar rápidamente el código que se desee consultar.

- Carpeta **/res/**

Contiene todos los recursos necesarios para el proyecto: imágenes, vídeos, cadenas de texto, estilos, etc. Al mismo tiempo se dividen en las siguientes carpetas:

- ✓ **/res/drawable/** Contiene las imágenes, iconos y otros elementos gráficos que se utilizarán en la aplicación. Se subdivide según el tamaño y calidad de las imágenes para cargar unas u otras dependiendo de la resolución de la pantalla del terminal. Éstas son: /drawable-ldpi, /drawable-mdpi, /drawable-hdpi, /drawable-xhdpi y /drawable-xxhdpi.
- ✓ **/res/layout/** Contiene los ficheros XML que definen la interfaz gráfica de las distintas pantallas por las que esté compuesta la aplicación. Normalmente debe existir un fichero XML por cada pantalla. Se puede dividir en /layout y /layout-land para definir distintos layouts dependiendo de la orientación de la pantalla (horizontal o vertical):
- ✓ **/res/menu/** Esta carpeta contiene ficheros XML que definen los menús y sus opciones de la aplicación.
- ✓ **/res/values/** Esta carpeta es muy importante ya que contiene los recursos de cadenas de texto, estilos, colores, etc. En esta aplicación se utilizará únicamente el fichero de cadenas de texto *string.xml*, desde él se podrá modificar cualquier texto, título, comentario, etc. que aparezca en la aplicación. También es el archivo que se utiliza para cambiar de idioma una aplicación.
- ✓ **/res/xml/** Contiene otros ficheros de datos XML utilizados.
- ✓ **/res/raw/** Contiene recursos adicionales, normalmente en un formato distinto a XML.

- Carpeta **/gen/**

Contiene los ficheros que genera automáticamente Eclipse al ser compilado el proyecto. En ella podemos encontrar el fichero R.java que contiene la clase R donde son identificados todos los recursos ubicados en la carpeta /res entre otros.

- Capeta **/assets/**

Carpeta que contiene recursos auxiliares utilizados en la aplicación y que se incluyen en el propio paquete. La diferencia entre los recursos de la carpeta /res/ con éstos radica en que para los primeros se genera un ID en la clase R y se pueden acceder a ellos referenciando dicho ID. En cambio para los de la carpeta *assets* no se generará ningún ID, por lo que para acceder a ellos sería por su ruta como a cualquier fichero del sistema. Normalmente se utiliza para recursos que no requieran modificación.

- Carpeta **/bin/**

Contiene los ficheros compilados de la aplicación. Aquí es donde se genera el

ejecutable para poder instalar la aplicación con la extensión *.apk.

- Carpeta **/libs/**

  Contiene las librería auxiliares necesarias como por ejemplo *maps.jar*. Éstas suelen estar en el formato de compresión .jar.

- Fichero **AndroidManifest.xml**

  Es el fichero que describe los aspectos más importantes de la aplicación. En él se definen características como el nombre, la versión o el icono, también se definen permisos para que la aplicación pueda utilizar por ejemplo Internet o el GPS además de contener la definición de todas las actividades marcada una de ellas como la principal (*home launcher*).

5.3. Programación

Este apartado se centrará en mostrar y explicar las partes más relevantes del código fuente de la aplicación. Para tener una idea siempre de qué parte del proyecto sobre Eclipse se está trabajando se utiliza la *Imagen 8* para establecer una relación entre la nomenclatura de las actividades descritas y su nombre en el entorno ID.

INICIO	MainActivity
Resumen	StartFragment
Historial	HistoryFragment
VISUALIZAR	ViewRouteActivity
Datos	ViewRouteDataFragment
Mapa	ViewRouteMapFragment
Datos del usuario	DataFragment
Explorar el mapa	MapsFragment
Acerca de	AboutFragment
EN CURSO	RecordActivity
DETENIDA	SaveActivity

5.3.1. AndroidManifest

La configuración, los permisos de los servicios como el GPS y la declaración de actividades se realizan en este fichero.

Lo primero que se observa en este tipo de ficheros obviamente es el elemento raíz "*manifest*", que engloba todo los parámetros y que incluye como atributo más importante el *package*. Éste muestra el nombre del paquete que se define en el *wizzard* de la creación del proyecto.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="pfc.sauirqui.followtrack"
    android:versionCode="1"
    android:versionName="1.0">
    . . .
</manifest>
```

La versión más baja de Android bajo la que se puede ejecutar esta aplicación es la correspondiente a la API 14, es decir Android 4.0 "Ice Cream Sandwich".

```
<uses-sdk
    android:minSdkVersion="14"
    android:targetSdkVersion="19" />
```

Para declarar los permisos que necesita la aplicación y funcionar de manera apropiada se necesita *uses-permission*. Entre ellos se pueden encontrar el de acceso a GPS o a Internet por ejemplo. A continuación se muestra y detalla cada uno de los utilizados en este proyecto:

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Permite que la API compruebe el estado de la conexión con el fin de determinar si los datos se pueden descargar.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Permite descargar fragmentos de mapa desde los servidores de *Google Maps*.

```
<uses-permission
    android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
```

Permite el acceso a los servicios de *Google Play Service*.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Permite que se guarde parte de los mapas en el caché de la memoria externa del dispositivo.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Permite utilizar el Wi-Fi o la red de datos (o ambos) para determinar la posición del usuario.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Permite utilizar el sistema de posicionamiento global (GPS) para establecer la ubicación del usuario con la máxima precisión.

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

Permite que la aplicación no entre en suspensión mientras se está ejecutando.

Otro tipo de permiso para utilizar una característica ya sea de hardware o de software es el *uses-feature*. En esta aplicación se usa para poder utilizar aceleración gráfica, en concreto para usar Open GL 2.0 ES.

```
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true" />
```

Unos de los apartados imprescindibles en el fichero de AndroidManifest.xml es el atributo referido a la aplicación en sí (*application*). En él se describen el nombre, los iconos, la actividad principal, etc. Dentro debe albergar todas las actividades que se utilicen. A continuación se explican las líneas de código menos inteligibles.

```
<application
    android:allowBackup="true"
```

Habilita o deshabilita que puedan crearse copias de seguridad y restauraciones.

```
    android:icon="@drawable/ic_launcher_followtrack"
    android:label="@string/app_name"
    android:theme="@style/AppTheme"
    android:largeHeap="true">
```

Permite utilizar más memoria instantánea durante la ejecución de la aplicación.

```
    <meta-data
        android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />
```

Contiene la versión de los servicios de *Google Play* para los que se compiló la aplicación.

```
    <meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="AIzaSyDEzIx-nC51eRRi1x4obzDa2bOQi0Volfk" />
```

Establece el valor de la clave de *Google Maps* para poder visualizar el

fragmento de mapa en la aplicación. En el apartado 5.3.3.2 se explica con más detalle.

```
<activity
    android:name="pfc.sauirqui.followtrack.MainActivity"
    android:label="@string/app_name"
    android:screenOrientation="portrait">
```

Portrait indica que la actividad siempre estará en vertical con respecto a la pantalla. Si se deseara en horizontal se utilizaría "*landscape*".

```
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
```

Establece la actividad principal, es decir la actividad que se muestra cada vez que se inicie la aplicación.

```
</activity>

<activity
    android:name="pfc.sauirqui.followtrack.RecordActivity"
    android:label="@string/app_name"
    android:screenOrientation="portrait">
</activity>

<activity
    android:name="pfc.sauirqui.followtrack.SaveActivity"
    android:label="@string/app_name"
    android:screenOrientation="portrait">
</activity>

<activity
    android:name="pfc.sauirqui.followtrack.ViewRouteActivity"
    android:label="@string/app_name"
    android:screenOrientation="portrait">
</activity>

</application>
```

5.3.2. MainActivity

MainActivity es la actividad principal de la aplicación, esto quiere decir que se inicia en ella. Contiene los fragmentos que se detallan en los subapartados de este punto, un cajón de navegación (*navigation drawer*), algunas opciones del A*ctionBar* y la definición del comportamiento cuando se pulsa el botón *Atrás*.

✧ Opciones del ActionBar

Para empezar dotaremos de opciones al *ActionBar*. En esta actividad se definen dos: la primera de ellas será un icono visible en todo momento para poder iniciar la grabación de la ruta desde cualquier fragmento; y la segunda, estará en el *overflow* (parte no visible del *ActionBar*) y será **Salir**, que permitirá abandonar la aplicación totalmente.

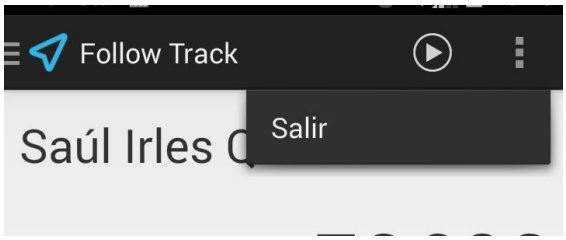

Imagen 24. ActionBar.

Para ello se debe definir en la carpeta "res" → "menu" un archivo xml, al que luego se hará referencia dentro de nuestro código java de la actividad, que contenga las referencias a los nombres.

Al crear el proyecto con una actividad, ya se proporciona un archivo (main.xml) el cual se utiliza en esta ocasión.

Primero se sobrescribe el método *onCreateOptions()* y se le dice que infle el archivo xml citado anteriormente, *main.xml*.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
```

Ahora se modifica para añadirle las nuevas opciones. Quedaría de la siguiente forma:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/action_start_record"
```

```
            android:icon="@drawable/ic_action_play"
            android:orderInCategory="100"
            android:showAsAction="ifRoom"
            android:title="@string/action_start_record"/>
        <item
            android:id="@+id/action_exit_app"
            android:orderInCategory="100"
            android:showAsAction="never"
           android:title="@string/action_exit_app"/>
</menu>
```

Como se observa cada *item* contiene varios argumentos que lo definen:

- *android:id* Identificación (el símbolo + indica que se crea una variable nueva para éste).
- *android:showAsAction* Indica si el ítem se muestra en el *ActionBar* o en el menu oculto.
- *android:title* Le proporciona un nombre, en estos casos se definen en el archivo *string*.
- android:icon Hace referencia al icono de la carpeta *drawable* que debe utilizar.

Y para finalizar se debe implementar, dentro de la función principal, la lista de opciones mediante el método:

```
public boolean onOptionsItemSelected(MenuItem item) {
    //Asigna a cada elemento del menú overflow del actionbar una acción.
    switch(item.getItemId())
        {
         case R.id.action_start_record:
                Toast.makeText(this, R.string.starting, Toast.LENGTH_SHORT).show();
                Intent intent = new Intent(MainActivity.this, RecordActivity.class);
                startActivity(intent);
                 break;
                    Pulsando sobre esta opción se cambia de actividad, en este caso
                cambia a "EN CURSO".
          case R.id.action_exit_app:
                DialogFragment dialogo = new DialogExit();
                FragmentManager fragmentManager = getFragmentManager();
                dialogo.show(fragmentManager, "tagExit");
                    Crea un diálogo para salir de la aplicación.
          default:
                return super.onOptionsItemSelected(item);
```

```
            }
        return super.onOptionsItemSelected(item);
    }
```

El diálogo que se utiliza para salir de la aplicación se muestra a continuación y extiende de la clase *DialogFragment*, se crea una ventana que flota en el centro de la actividad.

```
public class DialogExit extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder exitDialog = new AlertDialog.Builder(getActivity());
        exitDialog.setTitle(R.string.action_exit_app);
        exitDialog.setMessage(R.string.question_exit_app);
        exitDialog.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog,int which) {
                Intent intent = new Intent(Intent.ACTION_MAIN);
                intent.addCategory(Intent.CATEGORY_HOME);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(intent);
            }
        });
        Define qué pasa cuando se pulsa en botón de aceptar.
        exitDialog.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
            }
        });
        return exitDialog.create();
    }
}
```

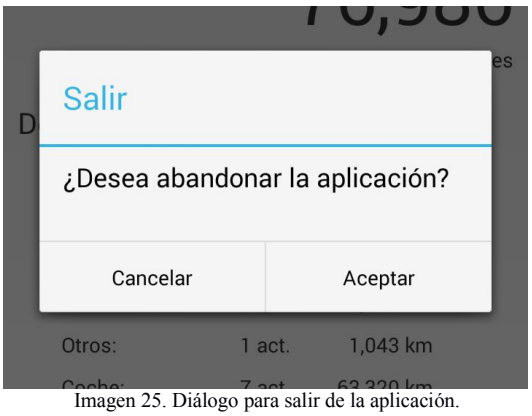

Imagen 25. Diálogo para salir de la aplicación.

✧ Botón Atrás

Cuando se está navegando entre los distintos fragmentos de MainActivity es posible que se tienda a pulsar el botón *Atrás* para volver al fragmento anterior, pero en realidad éste solo vuelve atrás entre actividades. Es decir, si se pulsa el botón *Atrás* en la primera actividad, dando igual el fragmento sobre el que esté, se saldrá de la aplicación. Para evitar esto, se sobrescribe el método *onBackPressed()* para otorgarle otro comportamiento, en este caso habrá que pulsar dos veces el botón *Atrás* para salir de la aplicación. Si se pulsa una vez saldrá un *Toast* (notificación temporal) que indique que hay que presionar otra vez para salir.

```
public void onBackPressed()
{
    if (back_pressed + 2000 > System.currentTimeMillis()){
        Intent intent = new Intent(Intent.ACTION_MAIN);
        intent.addCategory(Intent.CATEGORY_HOME);
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(intent);
    }else Toast.makeText(getBaseContext(), R.string.two_clicks_exit_app,
                                                    Toast.LENGTH_SHORT).show();
        back_pressed = System.currentTimeMillis();
}
```

Imagen 26. Al pulsar botón Atrás en MainActivity.

✧ Navigation Drawer

Navigation Drawer, es un menú vertical que aparece sobre la actividad en forma de fragmento y está disponible en las API's de Android desde el 2013.

Para implementarlo, se seguirán los siguientes pasos:

El primer paso será crear un *layout* de tipo *<android.support.v4.widget.DrawerLayout>* que defina el aspecto del mismo. Dentro de él se colocarán únicamente dos componentes, el *FrameLayout* que contendrá la información general y el *ListView* para la lista a mostrar. A continuación se expone el código para observar los distintos atributos con los que se puede jugar.

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <ListView android:id="@+id/left_drawer"
        android:layout_width="240dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="1dp"
      android:background="#424242"
      android:textColor="@android:color/white"/>
</android.support.v4.widget.DrawerLayout>
```

A continuación se deberá crear un *<string-array />* que contenga las entradas de la lista que debe contener el menú *NavigationDrawer* en el archivo *strings.xml*

```
<string-array name="nav_drawer_items">
        <item >Inicio</item>
        <item >Historial</item>
        <item >Datos Atleta</item>
        <item >Rutas</item>
```

```
        <item >Acerca de</item>
    </string-array>
```

Y para finalizar con el diseño del cajón se le añadirá otro archivo xml con un *TextView* que muestre los ítemes y defina algunas características. Se llamará drawer_list_items.xml y tendrá la siguiente forma:

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceListItemSmall"
    android:gravity="center_vertical"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:textColor="#fff"
    android:background="?android:attr/activatedBackgroundIndicator"
    android:minHeight="?android:attr/listPreferredItemHeightSmall" />
```

Una vez terminados los primeros pasos se pasará a agregar el código java necesario en nuestra actividad para hacer que funcione según lo estipulado en el diseño del comportamiento.

Para ello se abrirá el archivo java de la actividad principal, en este caso, MainActivity.java y se le añadirán las siguientes líneas:

Dentro del método *onCreate()*:

```
drawerListContent = getResources().getStringArray(R.array.nav_drawer_items);
drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
drawerList = (ListView) findViewById(R.id.left_drawer);
```

Se inicializan los *layouts* creados para la lista del menú y se les asigna a una variable que se utilizará a continuación.

```
drawerList.setAdapter(new ArrayAdapter<String>(this,
                                    R.layout.drawer_list_item, drawerListContent));
```

El método *setAdapter()* suministrará los datos y las vistas de cada ítem que contendrá la lista.

Con lo programado hasta ahora ya se podrá visualizar el *NavigationDrawer* con la lista de entradas del *Array*.

El siguiente paso será añadir un *listener* a través del método *setOnItemSelectedListener()* para saber cuándo se ha seleccionado una opción.

```
drawerList.setOnItemClickListener(new DrawerItemClickListener());
```

Según el diseño de la *app* se deberá mostrar desde un principio, antes de hacer clic en cualquier botón, el fragmento de inicio. Para ello añadiremos las siguientes líneas después del *setAdapter()*:

```
if (savedInstanceState == null) {
                displayView(0);
        }
```

Acto seguido se desarrollan las siguientes funciones dentro de la clase *MainActivity()* para que detecte cuando se pulsa un botón y guarde la posición del mismo.

```
private class DrawerItemClickListener implements ListView.OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
                        long id) {
                displayView(position);
        }
}
```

La función espera que se haga algún clic dentro del cajón para mostrar su contenido, luego coloca el título del fragmento en la cabecera de la aplicación excepto la opción inicio que siempre mostrará el nombre de la *app*, y finalmente se cierra el cajón. Se muestra en las siguientes líneas la resolución:

```
private void displayView(int position) {
        Fragment fragment = null;
        switch (position) {
        case 0:
                fragment = new StartFragment();
                break;
        case 1:
                fragment = new HistoryFragment();
                break;
        case 2:
                fragment = new DataFragment();
                break;
        case 3:
```

```
                fragment = new RoutesFragment();
                break;
            case 4:
                fragment = new AboutFragment();
                break;
            default:
                break;
            }

            if (fragment != null) {
                FragmentManager fragmentManager = getFragmentManager();
                fragmentManager.beginTransaction()
                        .replace(R.id.content_frame, fragment).commit();

                drawerList.setItemChecked(position, true);
                drawerList.setSelection(position);
                if (position == 0){
                    setTitle(R.string.app_name);
                }else{
                setTitle(drawerListContent[position]);
                }

                drawerLayout.closeDrawer(drawerList);
            } else {
                Log.e("MainActivity", "Error creando el fragmento");
            }
        }
```

Como se observa ya se tiene preparada la actividad principal para mostrar el *NavigationDrawer* y poder seleccionar las distintas opciones. Éstas vienen definidas gráficamente en el diseño inicial, pero ahora se deben crear los fragmentos y *layouts* necesarios para su representación real en la aplicación. Habrá dos archivos por cada opción, un xml *layout* y su implementación java que mostrarán su contenido en los subapartados de este punto.

Como último paso, para cumplir con los criterios de diseño de Google, se deberá poder acceder a este menú de navegación mediante un icono situado en el *ActionBar*.

```
public class MainActivity extends Activity {

    ...
    private DrawerLayout drawerLayout;
```

```
    private ActionBarDrawerToggle drawerToggle;
    ...
        drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        drawerToggle = new ActionBarDrawerToggle(
                        this,                    /* host Activity */
                        drawerLayout,            /* DrawerLayout object */
                        R.drawable.ic_drawer,    /* nav drawer icon to replace 'Up' caret */
                        R.string.drawer_open,    /* "open drawer" description */
                        R.string.drawer_close    /* "close drawer" description */
                        ) {

                /** Called when a drawer has settled in a completely closed state. */
                public void onDrawerClosed(View view) {
                        super.onDrawerClosed(view);
                }

                /** Called when a drawer has settled in a completely open state. */
                public void onDrawerOpened(View drawerView) {
                        super.onDrawerOpened(drawerView);
                }
        };

        // Set the drawer toggle as the DrawerListener
        drawerLayout.setDrawerListener(drawerToggle);

        getActionBar().setDisplayHomeAsUpEnabled(true);
        getActionBar().setHomeButtonEnabled(true);

}
  @Override
        protected void onPostCreate(Bundle savedInstanceState) {
                super.onPostCreate(savedInstanceState);
                // Sync the toggle state after onRestoreInstanceState has occurred.
                drawerToggle.syncState();
        }

        @Override
        public void onConfigurationChanged(Configuration newConfig) {
                super.onConfigurationChanged(newConfig);
                drawerToggle.onConfigurationChanged(newConfig);
```

```
...
}
```

Se le añade las siguiente lineas al método existente o*nOptionsItemSelected()*,

```
public boolean onOptionsItemSelected(MenuItem item) {

    //Activa la pulsación sobre el icono de actionbar para el nav drawer
    if (drawerToggle.onOptionsItemSelected(item)) {
        return true;
    }
```

Con esto ya tendremos completamente funcional el *NavigationDrawer* a la espera de añadir dentro de cada fragmento las funcionalidades pertinentes.

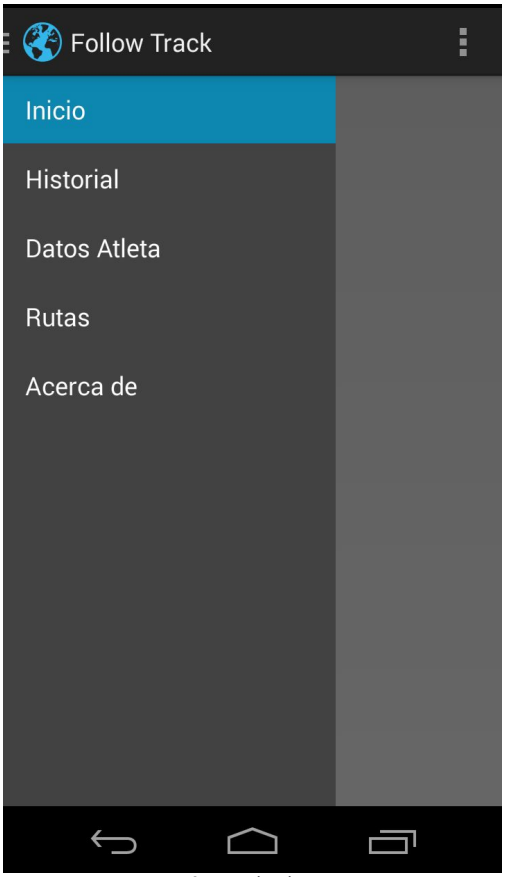

Imagen 27. Navigation Drawer.

5.3.2.1. Inicio

En la pantalla denominada *Inicio* correspondiente a la clase *StartFragment*, se muestra un resumen de las actividades realizadas y el usuario de la aplicación. En la imagen siguiente se aprecian los detalles:

Imagen 28. Pantalla Inicio.

Para realizar este fragmento se necesita una clase y un *layout*. En este último archivo, exceptuando la imagen del botón de *play*, únicamente estará compuesto de *TextView*'s, uno por cada texto. Dentro de éstos hay que diferenciar los que muestran el texto directamente desde el fichero de *strings.xml* y los que son variables y se cargan de cierta información de la aplicación.

Los *TextView*'s invariables, como lo son "km totales", "Detalle", y los nombres de las actividades se programarán como se indica en las siguientes líneas:

Definición en el *layout* (en este caso fragment_start.xml),

```
<TextView
        android:id="@+id/lbl_textTotalDistance"
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentRight="true"
android:layout_below="@+id/textTotalDistance"
android:layout_marginRight="10dp"
android:text="@string/lbl_textTotalDistance" />
```

Los parámetros que contiene se utilizan para identificarlo, ordenarlo dentro de la pantalla, definir el tamaño y su contenido. El último atributo indica el texto que contendrá, y éste va referido a un valor dentro del fichero *strings*:

```
<string name="lbl_textTotalDistance">km totales</string>
```

Con este simple paso ya se tiene en pantalla la etiqueta descrita en esta línea.

Para mostrar los datos variables también hay que definirlos en el *layout*, pero en este caso no se le especificará el atributo *android:text* ya que se le añadirá contenido en StartFragment.java. En este ejemplo se trabajará con el valor que se muestra bajo del usuario en la pantalla.

```
android:id="@+id/textTotalDistance"
```

Los pasos a seguir dentro de *StartFragment*:

✓ Declarar el objeto en la cabecera:

```
TextView textTotalDistance;
```

✓ Crearlo en onCreateView():

```
textTotalDistance = (TextView)rootView.findViewById(R.id.textTotalDistance);
```

✓ Instanciarlo en onResume():

```
textTotalDistance.setText(totalDistance2);
```

✓ Conseguir el valor de *totalDistance2*, para ello se llama a una función en *onCreate()* llamada getTotalDistance(); que consiga el valor a mostrar.

```
public Route getTotalDistance() {
    db = dbHelper.getReadableDatabase();
    route = new Route();
    Cursor cursor = db.query(DBHelper.ROUTE_TABLE, allColumns,
                                                    null, null, null, null, null);
   while(cursor.moveToNext()){
        route.setDistance(cursor.getString(5));
        totalDistance = Float.parseFloat(route.getDistance());
```

```
                totalDistance3 = totalDistance3+totalDistance;
                totalDistance2 = String.format("%.3f", totalDistance3);
            }
        cursor.close();
        if(totalDistance2 == null){
            totalDistance2 = String.valueOf(0);
        }
         return route;
    }
```

Esta función accede a la base de datos, que más adelante se explicará cómo se crea, y mediante la ayuda del objeto *route* se salvan y cargan los datos que el cursor lee de la tabla *ROUTE_TABLE*.

5.3.2.2. Historial

La finalidad de este fragmento es mostrar una lista de las rutas que se han almacenado en la base de datos. Una vez que se muestre la lista se debe poder interactuar con los ítemes para acceder a ellos o eliminarlos.

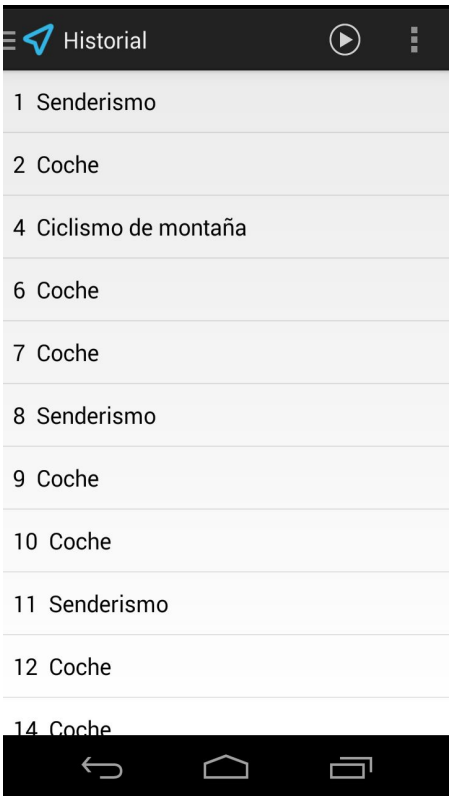

Imagen 29. Pantalla Historial.

Para su programación primero se debe crear una lista de todas las rutas accediendo a la base de datos. Estas serían los métodos y objetos a utilizar:

Dentro del método *onResume()* se hace una instancia del objeto *DBAdapter* y se accede a una de sus funciones para conseguir leer los datos salvados con la función *getAllRoutes()*. Una vez hecho esto se prepara para mostrar la lista con un *layout* predefinido de Android muy simple (*simple_list_item_1*).

```
dbAdapter = new DBAdapter(getActivity());
dbAdapter.open();
List<String> routes = dbAdapter.getAllRoutes();
adapter = new ArrayAdapter<String>(getActivity(), android.R.layout.simple_list_item_1,
                                                routes);
setListAdapter(adapter);
```

El método *getAllRoutes()* hace uso del cursor para extraer la información de la tabla de rutas que se define con la función *query*. En este caso se rescata con la ayuda del objeto *route* el tipo de actividad.

```
public List<String> getAllRoutes() {
        List<String> routes = new ArrayList<String>();
        Cursor cursor = db.query(DBHelper.ROUTE_TABLE,
            allColumns, null, null, null, null, null);
        while(cursor.moveToNext())
        {
            route= cursorToRoute(cursor);
            routes.add(route.getId() + "    " + route.getType());
        }
        cursor.close();
        return routes;
    }
```

A continuación se implementa el método para detectar qué entrada de la lista se ha pulsado y mostrar su contenido.

```
public void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    id = Long.parseLong(adapter.getItem(position).substring(0,3).trim());
    SharedPreferences preferences =
getActivity().getSharedPreferences("MyPreferences",Context.MODE_PRIVATE);

    SharedPreferences.Editor editor = preferences.edit();
    editor.putLong("idForViewRoute", id);
```

```
        editor.commit();
        Intent intent = new Intent(getActivity(), ViewRouteActivity.class);
        startActivity(intent);
    }
```

Para finalizar se crea el menú contextual para que al presionar tres segundos sobre una entrada muestre la siguiente ventana:

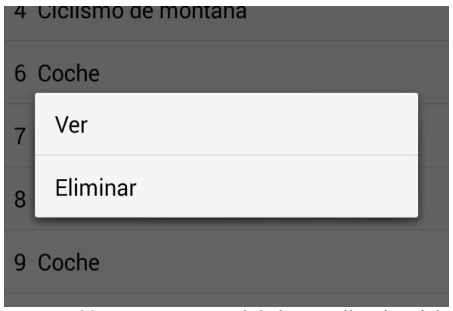

Imagen 30. Menú contextual de la pantalla Historial.

Al pulsar sobre "Ver" cambiará a la actividad de VISUALIZAR y al pulsar sobre eliminar se abrirá un diálogo que pida la confirmación para proceder. Se añadiría el código contiguo:

En *onResume()*: registerForContextMenu(this.getListView());

Y los nuevos métodos para definir la interfaz de la ventana y su comportamiento:

```
public void onCreateContextMenu(ContextMenu menu, View v,
            ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        MenuInflater inflater = getActivity().getMenuInflater();
        inflater.inflate(R.layout.contextual_menu, menu);
    }

public boolean onContextItemSelected(MenuItem item) {
        AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
        switch (item.getItemId()) {
            case R.id.CtxLblOpc1:
                Intent intent = new Intent(getActivity(), ViewRouteActivity.class);
                startActivity(intent);
                return true;
            case R.id.CtxLblOpc2:
                delete(Integer.parseInt(adapter.getItem(info.position).substring(0,3).trim()));
                return true;
```

```
                default:
                    return super.onContextItemSelected(item);
            }
        }
```

La función *delete()* crea un diálogo y cuando se pulsa el botón de confirmación se ejecuta lo siguiente para eliminar todo lo relacionado con el ítem pulsado de las tablas de la base de datos.

```
public void deleteRoute(long id) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    db.delete(dbHelper.ROUTE_TABLE, dbHelper.KEY_ID + " = " +id, null);
    db.delete(dbHelper.LOCATIONS_TABLE, dbHelper.KEY_ROUTEID + " = " +id, null);
}
```

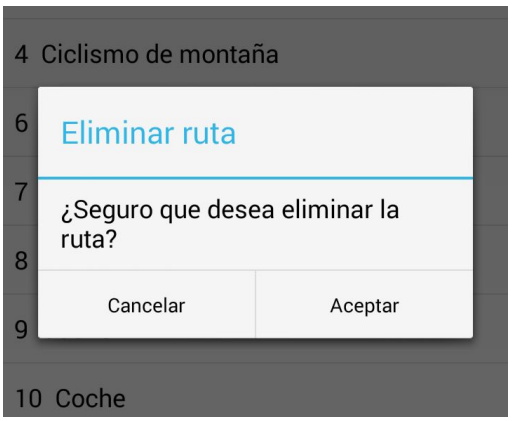

Imagen 31. Diálogo eliminar ruta desde Historial.

5.3.2.3. Explorar el mapa

En este fragmento se mostrará un mapa con la capa de satélite sobre la población de Elche y la posición del usuario. La finalidad de esta pantalla es la de explorar la zona antes de realizar el desplazamiento a grabar. En cuanto al código, se entrará en más detalle en el apartado 5.3.3.2.

La apariencia y el código se muestra a continuación:

Imagen 32. Pantalla Explorar.

```
private void setupMap() {
        if (googleMap == null){
            googleMap = getMap();
            if (googleMap != null){
                if (savedInstanceState == null){
                googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(ELCHE,
                                                                        10));
                }
                googleMap.setMyLocationEnabled(true);
                googleMap.getUiSettings().setZoomControlsEnabled(true);
                googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
            }
        }
    }
```

5.3.2.4. Acerca de

En este fragmento se muestra información relevante sobre la aplicación, como por ejemplo la versión, el programador, los derechos legales, la fecha de publicación y/o actualización, etc.

Se realizará mediante TextView's en el *layout* correspondiente cargando el texto desde el fichero *strings.xml*.

5.3.3. RecordActivity

RecordActivity.java corresponde con la clase de la actividad "EN CURSO", la más importante en toda la aplicación ya que gestiona los procesos de localización del usuario, muestra el mapa, genera la polilínea del trazado, guarda latitud y longitud en la base de datos y extrae las estadísticas del movimiento entre otras funciones.

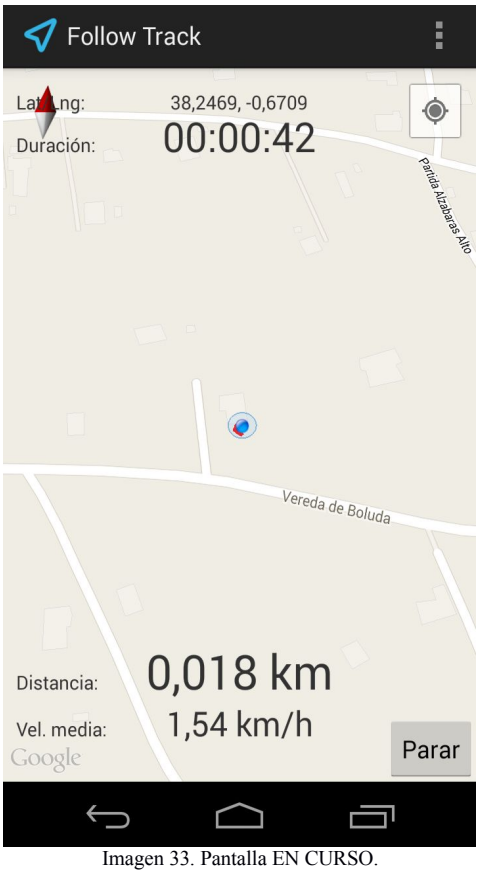

Imagen 33. Pantalla EN CURSO.

Como se observa en la imagen 33 esta actividad constará de un botón de parar que nos lleva a la siguiente pantalla, "ACTIVIDAD DETENIDA", de un mapa con funciones para dibujar el trazado utilizando el API v.2 de *Google Maps* y de cuatro objetos tipo *TexView* para mostrar datos informativos de posición, duración, distancia y velocidad.

Las partes más relevantes de esta actividad se separan en subapartados del punto, pero antes se comentarán ciertos aspectos y comportamientos.

✧ Diálogos

Consta de tres diálogos, el primero de ellos se inicia, si es necesario, junto con la actividad para avisar que el GPS está desactivado, permite entrar en el menú de configuración de conexiones inalámbricas y activarlo. A continuación se muestra el código necesario para implementarlo, pero como anteriormente ya se mostró cómo se creaba un diálogo, en los siguientes casos sólo se mostrará las funciones a las que habilita.

Imagen 34. Diálogo activar GPS.

Primero en *onCreate()* se verifica si está activo o no el GPS. En caso negativo se abre el diálogo.

```
if (!lm.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
        DialogFragment dialogo = new DialogSettingsGPS();
        FragmentManager fragmentManager = getFragmentManager();
        dialogo.show(fragmentManager, "tagGPS");
    }
```

Y la parte que se ejecuta cuando se pulsa el botón de *Configuración*:

```
Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
getActivity().startActivity(intent);
```

El próximo diálogo a comentar aparece cuando se presiona el botón *Atrás* de Android. Por el diseño funcional de la aplicación se quiere que únicamente se vaya hacia atrás desde esta actividad cuando se quiera descartar o cancelar el recorrido. Como puede haber almacenado alguna posición en la base de datos es necesario eliminar todas las entradas referidas a éste, por lo que el botón positivo del diálogo retrocederá a la actividad de "INICIO" eliminando los datos pertinentes.

Imagen 35. Diálogo descatar grabación.

Primero de sobrescribe el método *onBackPressed()* y desde él se llama al diálogo con las siguientes funciones que se realizan al pulsar el botón positivo.

```
lm.removeUpdates(locationListener);
```

Sobre esta función se entrará en detalle en otro apartado, pero su funcionalidad es cancelar la escucha de la recepción de la posición.

```
SharedPreferences preferences =
                        getSharedPreferences("MyPreferences",Context.MODE_PRIVATE);
int routeId = (preferences.getInt("lastRouteId", 0)+1);
```

SharedPreferences es una herramienta de Android para salvar información de cualquier tipo para posteriormente rescatarla desde la actividad que se necesite. En este caso se accede al valor que relaciona este recorrido en las tablas de la base de datos.

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
db.delete(DBHelper.LOCATIONS_TABLE, DBHelper.KEY_ROUTEID + " = " +routeId, null);
```

Elimina de la base de datos todos los elementos en que coincidan con el valor *routeId* rescatado en las líneas anteriores.

```
Intent intent = new Intent(RecordActivity.this, MainActivity.class);
startActivity(intent);
```

Cambia a la actividad MainActivity.

Por último se tiene el diálogo que se accede desde la parte oculta del *ActionBar* (*overflow*) y se utiliza para intercambiar las distintas capas de las que disponen los mapas. En el diálogo se muestra una lista de las capas para que se seleccione una de ellas.

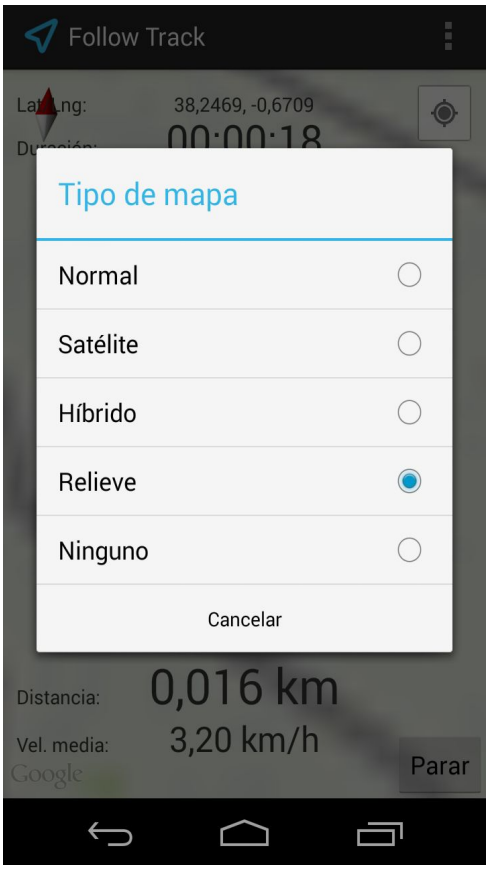

Imagen 36. Diálogo tipo de mapa.

Como este diálogo es distinto a los citados hasta el momento se mostrará a continuación el código más representativo.

Este diálogo, de la clase *setSingleChoiceItems()*, no contiene el botón positivo sino que realiza la función deseada cuando se pulsa sobre una de las entradas.

```
setSingleChoiceItems(R.array.type_map, id, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            String[] mapTypeArray = getResources().getStringArray(R.array.type_map);
```

Este código se utiliza para cargar la lista de las opciones desde el fichero *string.xml*.

```
            mapType = String.valueOf(mapTypeArray[id]);
            setMapType();
            SharedPreferences preferences =
```

```
                getSharedPreferences("MyPreferences",Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = preferences.edit();
        editor.putString("mapType", "" + mapTypeArray[id]);
        editor.putInt("mapTypeId",id);
        editor.commit();
        dialog.dismiss();
                }}).
```

Se le asigna a una variable *String* el valor pulsado de la lista para utilizarlo cuando se selecciona el tipo de mapa desde la función *setMapType()* y también para salvarlo y que la próxima vez que se ejecute esta actividad reanude ya con el tipo de mapa preferente.

```
private void setMapType(){
    if(mapType.equals(getString(R.string.type_map_normal))){
        googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
    }else if(mapType.equals(getString(R.string.type_map_satellite))){
        googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
    }else if(mapType.equals(getString(R.string.type_map_hybrid))){
        googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
    }else if(mapType.equals(getString(R.string.type_map_terrain))){
        googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
    }else if(mapType.equals(getString(R.string.type_map_none))){;
        googleMap.setMapType(GoogleMap.MAP_TYPE_NONE);
    }else{
    }
}
```

Función que compara el valor pulsado con un *String*, y en el caso de que sea igual ejecuta la línea adyacente que utiliza un objeto de *Google Maps* para cambiar el tipo del mapa.

✧ Clase PowerManager

Una vez que el usuario inicia el seguimiento de una ruta, se desea que la aplicación registre los datos de ubicación incluso si la pantalla está apagada. Se usa *PowerManager* para adquirir un *WakeLock* para evitar que el dispositivo entre en "sleep" y pueda continuar recibiendo datos GPS.

Dentro del método *onStart()* implementaremos las siguientes líneas.

```
PowerManager powerManager =    (PowerManager)
getSystemService(Context.POWER_SERVICE);
wakeLock = powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "No
                                                    sleep");
```

```
wakeLock.acquire();
```

Y para cancelar se liberará *WakeLock* en *onStop()*.

```
wakeLock.release(); //
```

El resto de código de esta actividad forma parte de los puntos siguientes.

5.3.3.1. LocationData

Android.location es el paquete que contiene las clases e interfaces necesarias para el uso y adquisición de datos de localización. En esta aplicación se utilizará:

✧ LocationManager

Proporciona acceso a los servicios de ubicación del dispositivo y son dependientes del hardware del dispositivo. Dependiendo del aparato móvil son varios los proveedores de posición que podrían usarse y es esta clase la que se encarga de elegir la mejor mediante el objeto *Criteria*.

✧ Criteria

Objeto que se encarga de escoger el mejor servicio proveedor según los requisitos de la aplicación en desarrollo. Los parámetros que se pueden definir son la precisión, uso de la batería, la velocidad, la altitud y el coste monetario entre otros. Una vez escogido el proveedor de ubicación se hará uso del objeto *LocationListener*.

✧ LocationListener

Se utiliza para escuchar la actualización de posición del usuario y si el dispositivo lo permite y lo requiere la aplicación aportará datos como latitud y longitud, altitud, velocidad...

✧ GpsStatus.Listener

Se emplea esta interfaz para escuchar cuando el terminal ya ha fijado su posición con varios satélites (suficientes datos para rastrear) y la exactitud de la localización es aceptable. Se podría no implementar esta función pero al iniciar la actividad el GPS podría ser muy inexacto y dar algunos valores pico que modificarían seriamente el resultado de los valores medios finales.

Ahora se analizará el código de los puntos expuestos arriba.

```
lm = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
locationListener = new MyLocationListener();
```

Primero se instancian los objetos de *LocationManager* y *LocationListener* en *onCreate()*.

El código dentro de *onStart()*:

```
List<String> locationProviders = lm.getAllProviders();
for (String provider : locationProviders) { Log.d("LocationProviders", provider); }
```

Consigue la lista de los proveedores existentes en el dispositivo.

```
Criteria c = new Criteria();
c.setAccuracy(Criteria.ACCURACY_FINE);//alta precisión
c.setAltitudeRequired(false);//que devuelva altura
c.setBearingRequired(false);//Rumbo
c.setCostAllowed(true); //coste monetario
c.setPowerRequirement(Criteria.POWER_HIGH);//consumo batería
```

Se establecen los criterios de selección.

```
String bestProvider = lm.getBestProvider(c, true);
Toast.makeText(getApplicationContext(), "Mejor proveedor: " +bestProvider,
                                                    Toast.LENGTH_LONG).show();
```

Se muestra el proveedor escogido, el mejor según los requisitos.

```
lm.requestLocationUpdates(
        bestProvider,
        0,     //mínimo tiempo de actualización(en milisegundos)
        0,     //mínima distancia de act.(en metros)
        locationListener);
```

Se indican los parámetros para actualizar la posición del GPS.

```
lm.addGpsStatusListener(gpsStatusListener);
```

Registro que indica si la posición ya es fija.

Como función independiente que implementa *GpsStatus.Listener* y que devuelve un valor booleano se tiene:

```
GpsStatus.Listener gpsStatusListener = new GpsStatus.Listener()
{
    public void onGpsStatusChanged(int event)
    {
```

```
            if (event == GpsStatus.GPS_EVENT_FIRST_FIX)
            {
                gpsFix = true;
                initTimeA = System.currentTimeMillis();
                    Se aprovecha esta función para establecer el tiempo inicial.
                Toast.makeText(getApplicationContext(),getResources()
                                                .getString(R.string.toast_signal_acquired),
                                                Toast.LENGTH_LONG).show();
            }
        }
};
```

Para escuchar los cambios de posición del *LocationManager* se necesita crear una clase e implementar la interfaz de *LocationListener* como se muestra a continuación.

```
private class MyLocationListener implements LocationListener
  {
    @Override
      public void onLocationChanged(Location loc) {
          if (loc != null&& gpsFix) {
```

Si *Location* no está vacío y la posición del usuario ya es fija se realizan las funciones del interior de *if*.

```
              double latitude = loc.getLatitude();
              double longitude = loc.getLongitude();
```

Se crean y asignan a la vez los valores de latitud y longitud a sus variables.

```
              DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
              Calendar cal = Calendar.getInstance();
              String time = dateFormat.format(cal.getTime());

              //conseguir tiempo actual
              currentTime = (System.currentTimeMillis() - initTimeA)/1000;

              //pasar el tiempo a formato HH:mm:ss
              HH_totalTime = currentTime/3600;
              mm_totalTime = (currentTime - 3600*HH_totalTime)/60;
              ss_totalTime = (currentTime - 3600*HH_totalTime - 60*mm_totalTime);

              //Pasarlo a cadena y dotarle de dos dígitos
              HH_totalTime2 = String.format("%02d", HH_totalTime);
              mm_totalTime2 = String.format("%02d", mm_totalTime);
```

```
                ss_totalTime2 = String.format("%02d", ss_totalTime);

                //cálculo de la velocidad en km/h
                averageSpeed = (distanceTraveled*3600)/(currentTime*1000);

                //coloca las variables obtenidas en textview asociado
                mLatLng.setText(getString(R.string.latitude_longitude, latitude, longitude));
                distance.setText(String.format("%.3f", distanceTraveled/1000) + " km");
                times.setText(HH_totalTime2 + ":" + mm_totalTime2 + ":" + ss_totalTime2);
                speed.setText(String.format("%.2f", averageSpeed) + " km/h");
            . . .
                if (previousLocation != null)
                 {
                     // add to the total distanceTraveled
                     distanceTraveled += loc.distanceTo(previousLocation);
                 }
               }
               previousLocation = loc;
        }
```

Las líneas superiores se utilizan para extraer los datos y mostrarlos por pantalla. Aprovecha la actualización de éste método para refrescarlos cada vez que cambian.

```
        @Override
        public void onProviderDisabled(String provider) {
        }

        @Override
        public void onProviderEnabled(String provider) {
        }

        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {
        }
}
```

5.3.3.2. Google Maps API v.2

Antes de empezar a mostrar el código es necesario hacer una pequeña introducción de la nueva versión de API de *Google Maps* citando algunos de sus cambios más destacados. Además se explicarán los pasos para conseguir una clave que permita descargar los mapas desde los servidores de Google.

Google Maps permite hacer uso de su sistema de cartografía online libremente en los dispositivos móviles. Aunque éste se pueda utilizar de manera libre no significa que lo sea, en este caso se trata de un servicio privativo dentro del paquete de *Google Play Services* y para utilizarlo habrá que aceptar antes unos términos y condiciones. También cabe destacar que para poder representar mapas del API v.2 el dispositivo móvil debe tener instalado los servicios citados de Google.

Los cambios respecto a la versión anterior son los siguientes:

- Integración con los *servicios de Google Play* y la *consola de API's*.
- Utilización a través de un tipo específico de *fragment, MapFragment*.
- Mayor velocidad de carga y mayor eficiencia en cuanto al uso de ancho de banda gracia al uso de mapas vectoriales.
- Mejoras en el sistema de caché que reduce las áreas en blanco.
- Permite la utilización de 3D, así se podrá cambiar el punto de vista de representación.

A continuación se muestra cómo conseguir la clave:

1. En primer lugar se debe prepara el IDE para poder cargar en el proyecto de nuestra aplicación las librerías necesarias, para ello se accede desde la barra de menús de *Eclipse* a *Window » Android SDK Manager* donde se abrirá un gestor que permite instalar, desinstalar y actualizar distintas características y paquetes.
   En él hay que seleccionar en *Extras* la opción de *Google Play Services y* descargarlo.

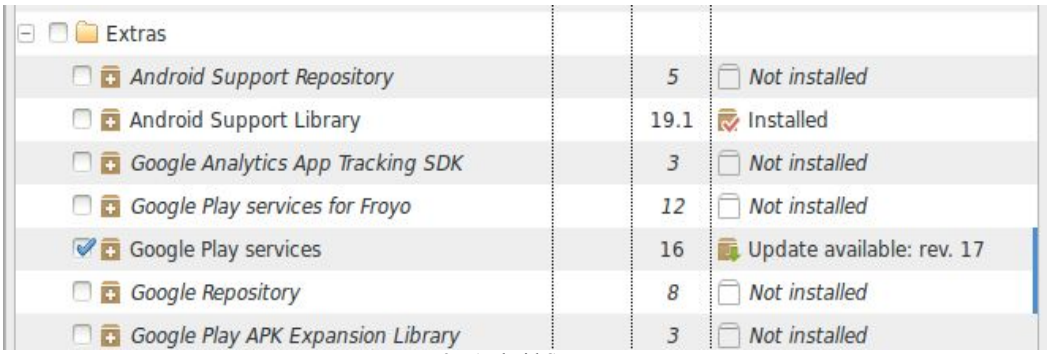

Imagen 37. Android SDK Manager.

2. Una vez descargados el paquete de servicios hay que importarlo dentro del entorno de desarrollo. Para ello se seleccionará *File » Import » Android » Existing Android Code Into Workspace* se escribirá la ruta donde se encuentra y asegurarse que esté activo el botón de *Copy projects into workspace*.

3. Importado a *Eclipse* ahora se debe referenciar a la aplicación que lo necesite. Se pulsará el botón derecho sobre el directorio raíz del proyecto *Properties » Android* y en el apartado inferior añadir las librerías.

4. Para acceder a los servidores de *Google Maps* con la nueva API se debe añadir una API_KEY a nuestra aplicación que estará asociada únicamente a ella utilizando la huella SHA-1.

5. Una vez obtenida la huella se accede a la consola de Google (https://console.developers.google.com/) donde si se tiene ya una cuenta de desarrollador, en el apartado de API's y autorizaciones se tendrá que activar *Google Maps Android API v2*. Seguidamente se le dará al botón de crear nueva llave para Android y en la ventana emergente se colocará el SH-1 y el nombre del paquete de la aplicación separados por ";". Devolverá la *API_KEY* cuando se termine este paso.

6. La llave se colocará en el archivo *AndroidManifest.xml* como se mostró en el apartado referido a él.

Con estos pasos ya está todo preparado para insertar los mapas en la aplicación y utilizar las distintas herramientas que ofrece la nueva API.

El código que interviene en el uso del objeto *GoogleMaps* es el siguiente:

```
GoogleMap googleMap;
```

Declaración de la variable de tipo *GoogleMap*.

```
setUpMapIfNeeded();
```

En *onStart()* se llamará a esta función con el código adyacente.

```
private void setUpMapIfNeeded() {
    if (googleMap == null) {
        googleMap =

((MapFragment)getFragmentManager().findFragmentById(R.id.map)).getMap();
```

Si el objeto está vacío, en este caso se refiere al mapa, se cargará mediante esta línea. Y en caso de tener ya un valor realiza la función *setUpMap()* para definir alguno de sus comportamientos.

```
            if (googleMap != null) {
                setUpMap();
            }
        }
    }

    private void setUpMap() {
        googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(ELCHE, 10));
```

Cuando se inicia el mapa hace un zoom sobre unos parámetros establecidos en la ciudad de Elche.

```
        googleMap.setMyLocationEnabled(true);
```

Se habilita detectar y mostrar donde se encuentra el dispositivo mediante un punto azul.

```
        googleMap.getUiSettings().setZoomControlsEnabled(false);
```

Deshabilita los controles mostrados en pantalla de zoom.

```
        SharedPreferences preferences =
                getSharedPreferences("MyPreferences",Context.MODE_PRIVATE);
        mapType = (preferences.getString("mapType", "Normal"));
        setMapType();
```

Estas líneas se utilizan para cargar el tipo de capa que se debe mostrar en el mapa, por defecto será normal, pero si el usuario lo ha cambiado durante un anterior uso mostrará el último seleccionado.

```
    }
```

También se considera importante cómo está definido el mapa en el *layout* de la actividad:

```
    <fragment
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.MapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

Para finalizar se mostrará cómo se dibuja la línea del trazado del recorrido y cómo se desplaza el mapa para conseguir tener centrada la ubicación del usuario. Todo ello se realizará dentro del método *onLocationChanged()*.

```
    List<LatLng> routePoints = new ArrayList<LatLng>();
```

Primero se declara una lista con el formato *LatLng* para ir guardando más

adelante los valores de latitud y longitud obtenidos.

```
LatLng newPoint = new LatLng(latitude, longitude);
routePoints.add(newPoint);
```

Se crean y se almacenan los puntos que definen la posición.

```
googleMap.addPolyline(new PolylineOptions()
        .addAll(routePoints)
        .width(5)
        .color(Color.RED)
        .geodesic(true));
```

Con esta función se dibuja la línea que define el recorrido cargando los datos de la lista y asignándole el aspecto de la misma.

```
CameraPosition cameraPosition = new CameraPosition.Builder()
        .target(newPoint)      // Sets the center of the map to Mountain View
        .zoom(17)              // Sets the zoom
        .bearing(0)            // Sets the orientation of the camera to east
        .tilt(30)              // Sets the tilt of the camera to 30 degrees
        .build();              // Creates a CameraPosition from the builder
```

Se describen algunos parámetros para que la posición de la cámara sea como el programador la defina.

```
googleMap.animateCamera(CameraUpdateFactory.
                newCameraPosition(cameraPosition),500,null);
```

Y para finalizar se describe cómo será el movimiento de la cámara sobre el mapa.

5.3.3.3. Almacenamiento SQLite

Se trata de un sistema de gestión de base de datos relacional, es decir, que permite establecer relaciones entre los distintos datos almacenados. Una de sus principales ventajas, por lo que Android la utiliza, es el poco espacio que ocupa, los pocos recursos que consume y que está escrita en C.

Lo que la hace tan liviana es que a diferencia de los sistemas de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica, sino que se integra con él. Ésto quiere decir que no hay comunicación entre procesos (que requiere muchos recursos) y que utiliza llamadas simples a subrutinas y funciones (mucho más eficiente).

Como ya se ha comentado es el que Android utiliza por defecto, por lo tanto no será necesario instalar servicios extras.

Para crear la base de datos y sus tablas según lo estipulado en el apartado de diseño, se creará una nueva clase que extienda *SQLiteOpenHelper* llamada *DBHelper()*. A continuación el código y algunos comentarios del mismo.

```
public class DBHelper extends SQLiteOpenHelper {

    public static final String KEY_LOCID = "Id";
    public static final String KEY_ROUTEID = "routeId";
    public final static String KEY_LAT = "lat";
    public final static String KEY_LNG = "lng";
    public final static String KEY_TIME = "time";
```

Declaración de las variables de la tabla *Locations*.

```
    public final static String LOCATIONS_TABLE = "Locations";
    public final static String DATABASE_CREATE = "CREATE TABLE " +
        LOCATIONS_TABLE + "("    +
        KEY_LOCID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
        KEY_ROUTEID + " INTEGER, " +
        KEY_TIME + " TEXT, " +
        KEY_LAT + " TEXT, " +
        KEY_LNG + " TEXT) ";
```

El lenguaje para manejar SQLite es en forma de cadenas de texto, por lo tanto aquí se define el texto que contiene las características de la tabla.

```
    public static final String KEY_ID = "Id";
    public static final String KEY_TYPE = "type";
    public static final String KEY_INITTIME = "initTimeB";
    public static final String KEY_FINALTIME = "finalTimeB";
    public static final String KEY_TOTALTIME = "totalTime";
    public static final String KEY_DISTANCE = "distance";
    public static final String KEY_SPEED = "speed";
```

Declaración de las variables de la tabla *Routes*.

```
    public final static String ROUTE_TABLE = "Routes";
    public final static String DATABASE_CREATE2 = "CREATE TABLE " +
        ROUTE_TABLE +    "( " +
        KEY_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
        KEY_TYPE + " TEXT, " +
```

```
        KEY_INITTIME + " TEXT, " +
        KEY_FINALTIME + " TEXT, " +
        KEY_TOTALTIME + " TEXT, " +
        KEY_DISTANCE + " TEXT, " +
        KEY_SPEED + " TEXT) ";
```

Igual que en la otra tabla se definen las características que más adelante, durante la creación de la misma, se utilizarán para darle forma.

```
public DBHelper(Context context, String name, CursorFactory factory,
        int version) {
    super(context, name, factory, version);
    // TODO Auto-generated constructor stub
}
public DBHelper(Context context) {
    super(context, null, null, 1);
    // TODO Auto-generated constructor stub
}
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(DATABASE_CREATE);
    db.execSQL(DATABASE_CREATE2);
}
```

En esta función se llaman a las cadenas de texto creadas anteriormente para crear las tablas.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + LOCATIONS_TABLE);
    db.execSQL("DROP TABLE IF EXISTS " + ROUTE_TABLE);
    onCreate(db);
}
```

Como su nombre indica actualiza las tablas si lo necesitan.

```
}
```

Las funciones involucradas directamente en la actividad actual son las siguientes:

```
private DBAdapter dbAdapter;
private DBHelper dbHelper;
```

Se declaran los objetos.

```
dbHelper = new DBHelper(getApplicationContext(), "locations.db", null, 1);
```

```
dbAdapter = new DBAdapter(getApplicationContext());
```

Se crean los objetos en *onCreate()*.

```
dbAdapter.insertLocations(routeId+1, time, String.valueOf(latitude), String.valueOf(longitude));
```

Se instanciarán dentro de *onLocationChaged()*, y esto hará que cada vez que se actualice la posición se ejecute la línea que, como se observa, pasa al objeto los valores de identidad de la ruta, el tiempo, la latitud y la longitud.

A continuación se muestran las funciones que definen *insertLocations()* dentro de DBAdapter.java.

```
public void insertLocations(int routeId, String time, String lat, String lng){
        ContentValues values = buildContentValuesFromLocations(routeId, time, lat, lng);
```

Función que inserta los valores a almacenar en el objeto *values* como apoyo.

```
        SQLiteDatabase db = dbHelper.getWritableDatabase();
```

Da permiso para poder escribir en la base de datos.

```
        db.insert(DBHelper.LOCATIONS_TABLE, null, values);
```

Inserta los datos en la tabla correspondiente.

```
        db.close();
 }

    public ContentValues buildContentValuesFromLocations (int routeId, String time, String
lat, String lng) {
        ContentValues values = new ContentValues();
        values.put(DBHelper.KEY_ROUTEID, routeId);
        values.put(DBHelper.KEY_TIME, time);
        values.put(DBHelper.KEY_LAT, lat);
        values.put(DBHelper.KEY_LNG, lng);
        return values;
    }
```

5.3.4. SaveActivity

Como se comentó en el apartado de diseño, esta clase denominada "ACTIVIDAD PARADA" contiene un resumen de los datos de la actividad anterior pudiendo regresar a ésta, guardar e ir al inicio de la aplicación o descartar e ir igualmente a inicio.

Imagen 38. Pantalla "PARADA"

Esta pantalla, como se observa, estará compuesta por cuatro datos que facilitan información recogida del recorrido realizado más un *spinner* con una lista de distintos tipos de actividades para guardarlos en la base de datos.

A continuación se entra en detalle en cuanto al código:

Se empezará por saber cómo se comparten los datos entre la actividad *RecordActivity* y la actual. Para ello se utiliza la herramienta disponible en Android denominada *Intent* que es un mecanismo para invocar componentes como por ejemplo las alarmas. En este caso se utiliza para llamar a un atributo adicional llamado *Extras* que sirve para salvar cierta información o estado con la forma clave/valor y recuperarla cuando se necesite volviendo a llamar a este servicio mediante *Intent*.

Se ha decidido guardar los datos en *RecordActivity* en el momento que se pulse el botón de parar:

```
Intent intent = new Intent(RecordActivity.this, SaveActivity.class);
intent.putExtra("initial_time", initTimeB);
intent.putExtra("final_time", finalTimeB);
intent.putExtra("total_time", currentTime);
intent.putExtra("distance_traveled", distanceTraveled);
startActivity(intent);
```

Para rescatar los valores de las variables almacenadas mediante *Extras* en la actividad actual se escribe el siguiente código:

```
final String resc_initTime = getIntent().getStringExtra("initial_time");
final Long resc_totalTime = getIntent().getLongExtra("total_time", 0);
final Double resc_distanceTraveled = getIntent().getDoubleExtra("distance_traveled", 0);
final String resc_finalTime = getIntent().getStringExtra("final_time");
```

Una vez compartidos los datos entre actividades y ser mostrados se pueden almacenar o eliminar, para ello se han implementado los botones de *Descartar* y *Guardar*. Cuando se pulsa el primero de ellos aparece un diálogo cuyo funcionamiento es idéntico al expuesto en el apartado 5.3.3. de la actividad "EN CURSO" en la sección de *Diálogos*. El segundo de los citados se expone a continuación.

```
saveButtonDB.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            final String resc_type = spinner.getSelectedItem().toString();
```

Rescata el valor seleccionado del *spinner*.

```
            dbAdapter.insertDataRoutes(resc_type, resc_initTime, resc_finalTime,
                    String.valueOf(resc_totalTime), totalDistance, averageSpeed2);
```

Como se explica en el apartado de la base de datos se almacenan las variables citadas entre paréntesis.

```
            lastRouteId = dbAdapter.saveLastRouteId();
            SharedPreferences preferences =
getSharedPreferences("MyPreferences",Context.MODE_PRIVATE);
            SharedPreferences.Editor editor = preferences.edit();
            editor.putInt("lastRouteId", lastRouteId);
            editor.commit();
```

Estas líneas se utilizan para rescatar de la base de datos el último valor del identificador de rutas para salvarlo en *SharedPreferences* y poder utilizarlo de manera más simple en otras actividades.

```
            Intent intent = new Intent(SaveActivity.this, MainActivity.class);
```

```
                startActivity(intent);
            }
        });
```

Las tablas en la base de datos SQLite una vez que en la aplicación se haya pasado por las actividades "EN CURSO" y "PARADA" y en esta última haber pulsado el botón de guardar quedarían de la siguiente manera, suponiendo que se han realizado ya varios recorridos:

Id	type	initTimeB	finalTimeB	totalTime	distance	speed
9	Bicicleta	2014/04/28 16:53:49	2014/04/28 17:00:23	393	2,453	22,47
15	Senderismo	2014/04/28 17:32:59	2014/04/28 17:38:48	349	0,005	0,05
16	Coche	2014/04/28 18:12:20	2014/04/28 18:28:21	961	12,640	47,35
17	Coche	2014/04/28 22:11:08	2014/04/28 22:18:02	414	4,930	42,87
18	Patines	2014/04/29 10:39:20	2014/04/29 10:39:42	21	0,000	0,00
20	Senderismo	2014/04/29 11:05:30	2014/04/29 11:08:21	171	0,173	3,64
21	Senderismo	2014/04/29 16:29:13	2014/04/29 16:29:43	29	0,002	0,25
24	Coche	2014/04/29 19:46:17	2014/04/29 19:53:03	406	6,167	54,68
25	Ciclismo de montaña	2014/04/30 10:50:05	2014/04/30 10:50:18	13	0,004	1,11
26	Ciclismo	2014/04/30 10:51:07	2014/04/30 10:51:15	7	0,004	1,96

Tabla 6. Ejemplo tabla de rutas.

En la siguiente tabla se han eliminado entradas por la inmensa cantidad de filas que existen. Para su comprensión se únicamente se muestran las siguientes:

routeId	time	lat	lng
9	2014/04/28 16:51:16	38.24703841	−0.67084435
9	2014/04/28 16:51:21	38.24694907	−0.6708964
9	2014/04/28 16:51:30	38.24693406	−0.67091839
9	2014/04/28 16:51:35	38.24678317	−0.67096116
9	2014/04/28 16:51:40	38.2466352	−0.67098695
15	2014/04/28 17:30:46	38.2467685	−0.67082418
15	2014/04/28 17:31:30	38.24703817	−0.67106294
15	2014/04/28 17:31:37	38.24695209	−0.6708518
15	2014/04/28 17:32:19	38.24698059	−0.67089001
15	2014/04/28 17:32:27	38.24695639	−0.67089956
15	2014/04/28 17:32:32	38.24694008	−0.67090354
15	2014/04/28 17:32:37	38.24693285	−0.67088873
15	2014/04/28 17:32:42	38.24693411	−0.6708921
15	2014/04/28 17:38:39	38.24690761	−0.67089314
16	2014/04/28 18:12:25	38.2466152	−0.67097505
16	2014/04/28 18:12:32	38.24662457	−0.67104428
16	2014/04/28 18:12:37	38.24632459	−0.67108654
16	2014/04/28 18:12:42	38.24653081	−0.6716488
16	2014/04/28 18:12:51	38.2465835	−0.67311088
16	2014/04/28 18:12:56	38.24661628	−0.67405203
17	2014/04/28 22:11:16	38.2739727	−0.68008463
17	2014/04/28 22:11:24	38.27364818	−0.67991491
17	2014/04/28 22:11:29	38.27318185	−0.67975541
17	2014/04/28 22:11:38	38.27229529	−0.67939268
17	2014/04/28 22:11:43	38.27182923	−0.67916679
18	2014/04/29 10:39:38	38.24691626	−0.67085734
20	2014/04/29 11:05:36	38.24691842	−0.67085653
20	2014/04/29 11:05:42	38.24686441	−0.67087724
20	2014/04/29 11:05:51	38.24680778	−0.67089745
20	2014/04/29 11:06:00	38.24692992	−0.67089115
20	2014/04/29 11:06:09	38.24695966	−0.67083166
21	2014/04/29 12:30:27	38.24686337	−0.67126166
21	2014/04/29 12:47:36	38.24690555	−0.67092641

Tabla 7. Parte del ejemplo tabla localizaciones.

5.3.5. ViewRouteActivity

A esta actividad se accede desde el fragmento de "Historial" y se encarga de visualizar los recorridos salvados en la base de datos. Constará a su vez de dos fragmentos para mostrar en uno los datos del recorrido y en el otro su trazada sobre el mapa. Para ello se crean los fragmentos correspondientes a los subapartados de este punto, la denominación de ellos es *ViewRouteDataFragment* (Datos) y *ViewRouteMapFragment* (Mapa). Cabe destacar que para la visualización de éstos se hace uso del elemento de interfaz gráfica *ViewPager*. A continuación se explica cómo implementarlo:

Para empezar se añade en el *layout* de la actividad los elementos necesarios para definir el *ViewPager*. El segundo bloque, elemento hijo, define las características de la barra superior que se muestra en la imagen de su derecha.

```
<android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="0px"
        android:layout_weight="1">

        <android.support.v4.view.PagerTitleStrip
                android:id="@+id/pager_title_strip"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="top"
                android:background="#33b5e5"
                android:paddingBottom="4dp"
                android:paddingTop="4dp"
                android:textColor="#fff" />
</android.support.v4.view.ViewPager>
```

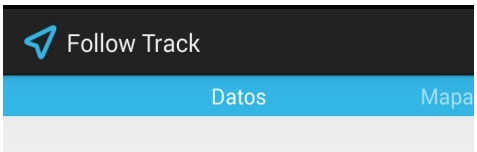

Imagen 39. ViewPager - PagerTitleStrip

Ya en el fichero de la actividad se debe hacer lo siguiente:

En *onCreate()* se crea, se asocia y se carga el elemento:

```
PagerAdapter adapter = new PagerAdapter(getSupportFragmentManager());
ViewPager mViewPager = (ViewPager) findViewById(R.id.pager);
mViewPager.setAdapter(adapter);
```

Y luego se crea una clase para definir su comportamiento que extienda de *PagerFragmentAdapter* y contenga algunos métodos esenciales para su funcionamiento.

```
    public class PagerAdapter extends FragmentPagerAdapter {

        public PagerAdapter(FragmentManager fm) {
            super(fm);
        }

        public Fragment getItem(int arg0) {
            switch (arg0) {
                case 0:
                    return new ViewRouteDataFragment();
                case 1:
                    return new ViewRouteMapFragment();
                default:
                    return null;
            }
        }
        public int getCount() {
            return 2;
        }
    }
}
```

5.3.5.1. **Datos**

Muestra los datos de la ruta seleccionada en el historial a partir de la base de datos. Se destacará el código referido a conseguir los datos:

```
    public Route getItem() {
            db = dbHelper.getReadableDatabase();
            route = new Route();
            Cursor cursor = db.query(DBHelper.ROUTE_TABLE, allColumns,
                                     DBHelper.KEY_ID + " = " +id_route,
                                     null, null, null, null);

          while(cursor.moveToNext()){
                route.setId(cursor.getInt(0));
                route.setType(cursor.getString(1));
```

```
                route.setInitTimeB(cursor.getString(2));
                route.setFinalTimeB(cursor.getString(3));
                route.setTotalTime(cursor.getString(4));
                route.setDistance(cursor.getString(5));
                route.setSpeed(cursor.getString(6));
                cursor.close();
            }
            return route;
        }
```

Como se observa se crea un cursor que recorre todas las filas de la tabla de rutas y extrae los datos cuando se cumple una condición, que la llave identificadora sea igual a la referida en la entrada del historial pulsada (*id_route*).

También se puede ver en este fragmento de código que se utiliza un objeto llamado *Route* que ayuda a compartir las variables, únicamente está formado por *setters* y *getters*.

Imagen 40. Datos.

5.3.5.2. Mapa

En este fragmento se muestra el mapa y el trazado del recorrido añadiendo unos *Markers* en el punto inicial y final.

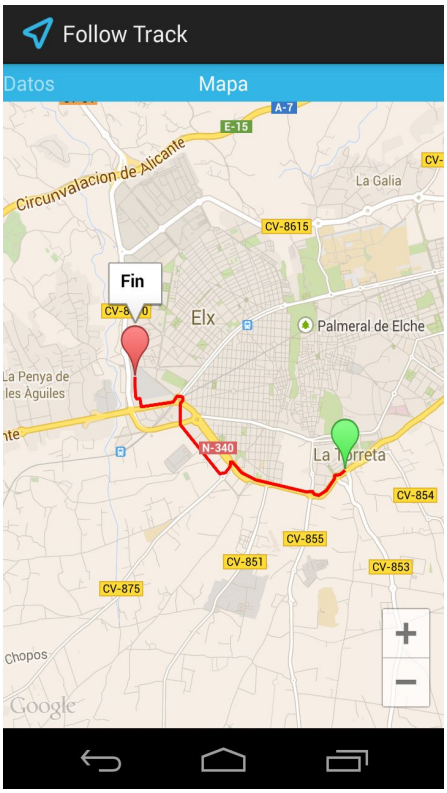

Imagen 41. Mapa.

El código más relevante de esta pantalla:

getItemToDraw();

Función que se llama desde *onStart()* para conseguir la lista de puntos y trazar línea.

```
googleMap.addPolyline(new PolylineOptions()
    .addAll(routePoints)
     width(5)
     .color(Color.RED)
     .geodesic(true));
```

Como se comentó en otro apartado gracias a estas líneas se carga desde una lista de pares de puntos (longitud, latitud) la polilínea.

```
    if(firstPoint==null){
```

```
            googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(ELCHE, 13));
            Toast.makeText(getActivity(), "La ruta no contiene información. ¡Eliminar del
                                historial!", Toast.LENGTH_LONG).show();
```

Este condicional trata de evitar un error producido al intentar abrir una ruta cuando la lista de pares de puntos está vacía, por ello se indica que si es así se sitúe el mapa sobre Elche y aparezca una notificación informativa de la situación del contenido de la ruta.

```
        }else{
            googleMap.addMarker(new MarkerOptions()
             .position(firstPoint)
             .title("Inicio")
             .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREE
                                                                                    N)));
            googleMap.addMarker(new MarkerOptions()
             .position(lastPoint)
             .title("Fin")
            .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED)));
            googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(firstPoint,13));
        }
}
```

En caso de que sí contenga información se colocará una señalización verde en el punto de partida y otra roja en el punto final como se indica arriba.

```
public Locations getItemToDraw(){

        db = dbHelper.getReadableDatabase();
        location = new Locations();
        Cursor cursor = db.query(DBHelper.LOCATIONS_TABLE,
                                        allColumns,
                                        DBHelper.KEY_ROUTEID + " = " +id_route,
                                        null, null, null, null);
        while(cursor.moveToNext())
            {
             location.setLatitude(cursor.getDouble(3));
             location.setLongitude(cursor.getDouble(4));
             if(newPoint==null){
                    newPoint = new LatLng(location.getLatitude(),location.getLongitude());
                    firstPoint = newPoint;
                    routePoints.add(newPoint);
```

```
                }
                newPoint = new LatLng(location.getLatitude(),location.getLongitude());
                routePoints.add(newPoint);
            }
        lastPoint = newPoint;
        cursor.close();
        return location;
    }
```

Como en el apartado de "Datos" se crea un cursor para buscar la información referente a la actividad seleccionada en la base de datos. Esta vez la información extraída se irá colocando en la lista llamada *routePoints*.

6. Conclusión

La finalidad de este tipo de proyecto desarrollado bajo Android era conseguir realizar una aplicación que hiciera un seguimiento de la posición instantánea del usuario. Además, según los objetivos, era necesario cumplir con unos requisitos mínimos como diseñar una interfaz gráfica amigable, extracción y representación de los datos más significativos en cuanto al recorrido se refiriese y poder almacenar las rutas para una posterior consulta. Todos estos puntos son conseguidos en la aplicación finalmente nombrada "**FollowTrack**".

Aunque el objetivo final de la aplicación era la realización de la *app*, se hizo a priori un estudio de la tecnología utilizada, la telefonía móvil y el GPS, dotando a este proyecto de un enfoque histórico que va desde la primera llamada con un teléfono móvil realizada por el CEO de Motorola *Martin Cooper* en 1973 hasta la actualidad donde los móviles son computadoras diminutas, llamados *smartphones* (véase Imagen 1).

Durante el desarrollo de *FollowTrack* se han tenido que utilizar muchos de los recursos de la plataforma Android. El tipo de aplicación realizada ha necesitado un profundo análisis teórico y práctico de los servicios de *Google Play Services* para la utilización de los mapas, de la librería de *Locations* para la utilización del GPS con el fin de localizar la posición, de la base de datos *SQLite* para almacenar los recorridos, de la herramienta *SharedPreferences* para poder compartir ciertos datos entre las actividades y guardar preferencias de configuración, de los ficheros XML del *layout* para la interfaz gráfica por citar algunos de los más importantes de los múltiples utilizados. Cabe destacar la gran cantidad de información extraída de Internet para la documentación sobre estos recursos, donde se ha podido encontrar y solucionar cualquier problema surgido durante la programación.

La intención al realizar esta aplicación no es crear algo nuevo, ya que existen multitud de aplicaciones similares para registrar las rutas más completas y complejas, sino es realizar una *app* sencilla, con gran funcionalidad y personal. De las ventajas a extraer cabe mencionar que se trata de una aplicación abierta a crecer y ampliar sus características. Algunas de las posibles mejoras que se podrían realizar son:

- ✓ Extraer la altitud.
- ✓ Registrar velocidad máxima.
- ✓ Cambiar la vista de la actividad Historial para una mayor identificación de las rutas.

Las citadas estarán disponibles en la próxima actualización de la aplicación, junto con posibles cambios en ciertos aspectos debido a la constante evolución de la plataforma Android.

En cuanto al nombre y al icono utilizados, se decidieron por ellos por su simplicidad, por su internacionalidad y por ser descriptivos en cuanto a la temática de la aplicación.

Follow Track

7. Manual de usuario

En este apartado se explicará cómo navegar e interactuar entre las distintas pantallas. También se darán algunos comentarios como consejo antes de empezar a utilizar la aplicación.

Durante la instalación de la aplicación **FollowTrack** se pedirá que se confirme el acceso a ciertas características del teléfono.

Una vez es instalada la primera pantalla que aparece es la que se muestra en la imagen 42. Se trata de un resumen global del kilometraje efectuado hasta el momento y el número de actividades por tipo. También se muestra el nombre del usuario que se deberá insertar en los pasos siguientes. Los iconos de "Play" situados en la parte inferior de la imagen y en el *ActionBar* son para iniciar la grabación. Este último permanecerá constante entre los distintos apartados de la actividad principal para empezar a grabar desde cualquiera.

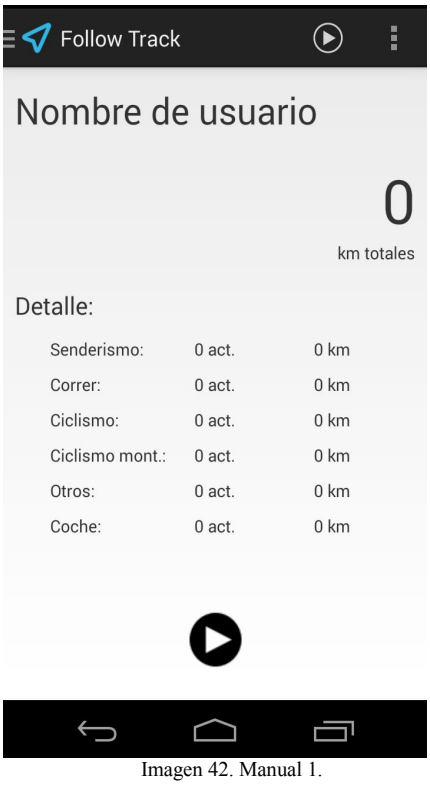

Imagen 42. Manual 1.

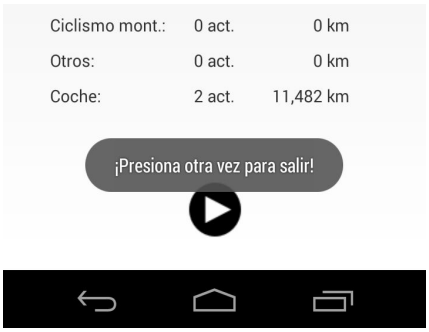

Imagen 43. Manual 2.

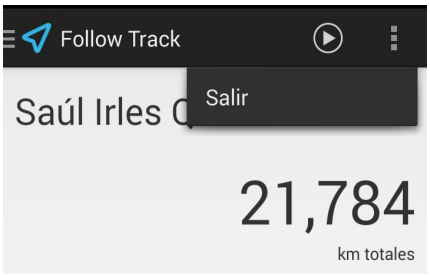

Imagen 44. Manual 3.

Para salir de la aplicación se tendrán dos opciones, las correspondientes a las imágenes 43 y 44. En la primera se pulsará dos veces sobre el botón de retroceso, y la segunda mediante el *overflow* del *ActionBar*, donde se encuentra la opción "Salir" y que abre un diálogo que habrá que aceptar.

Para examinar las distintas posibilidades dentro de la actividad de inicio se ha dotado a la aplicación de un "*Navigation Drawer*" (Cajón de navegación) con distintas pantallas de información y configuración. Para acceder a él se pulsará sobre el lado izquierdo del icono de la aplicación o se deslizará el dedo de izquierda a derecha de la pantalla iniciando desde el margen de la pantalla.

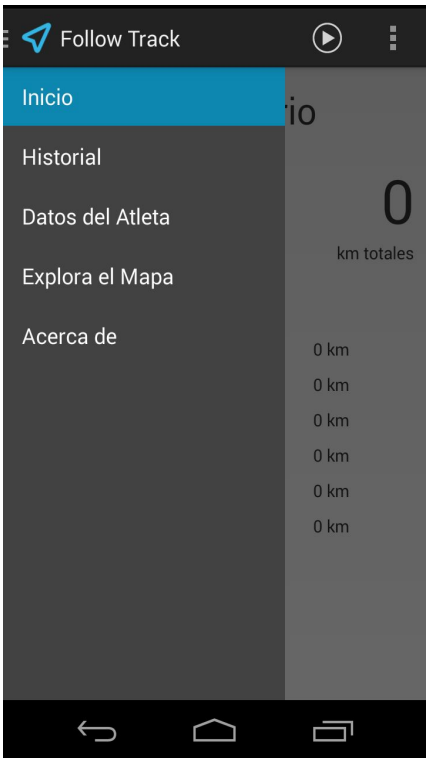

Imagen 45. Manual 4.

La pantalla inicio es la comentada en la imagen 42, el resto se hará a continuación.

HISTORIAL: Muestra una lista de las distintas actividades realizadas, conteniendo el identificador y el tipo como se aprecia en la imagen 46. Desde cada entrada se puede acceder a su contenido pulsando sobre ella y mostrando en primera instancia un resumen de las estadísticas del recorrido (imagen 47) y si se desliza el dedo de derecha a izquierda se accede al mapa con el recorrido (imagen 48). Los marcadores de color verde y rojo son para indicar el punto de salida y final respectivamente.

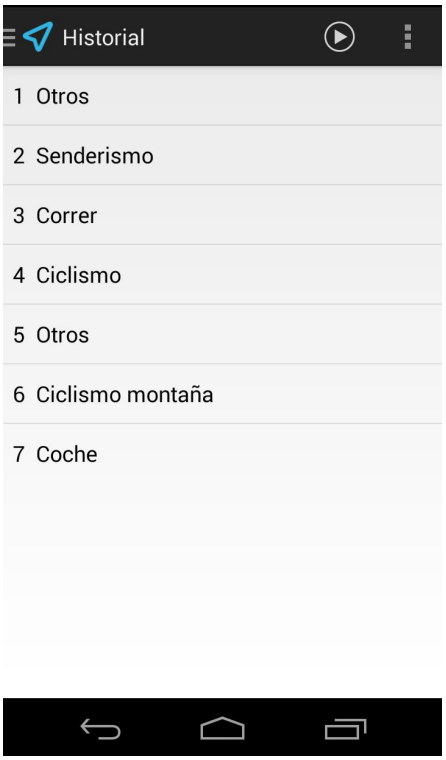

Imagen 46. Manual 5.

Imagen 47. Manual 6.

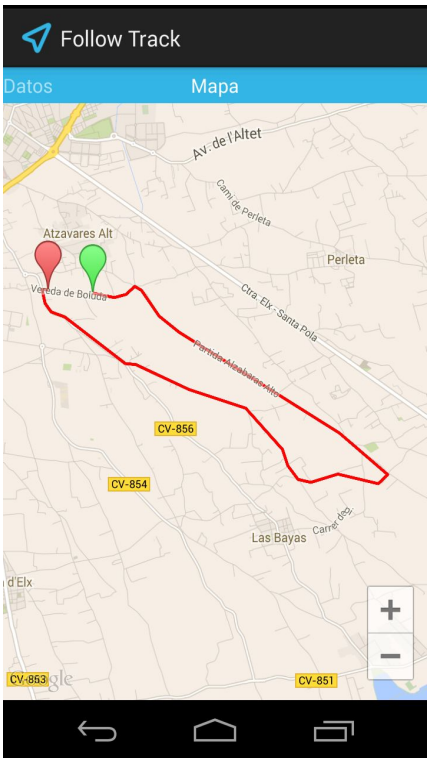

Imagen 48. Manual 7.

Desde la pantalla historial también es posible eliminar las rutas que no se deseen, para ello hay que presionar sobre la correspondiente durante 3 segundos y aparecerá

un menú contextual con dos opciones, una para visualizar y la otra para eliminar. Si se pulsa sobre esta última se creará un diálogo de confirmación.

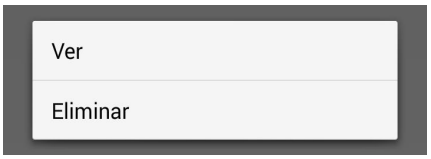

Imagen 49. Manual 8.

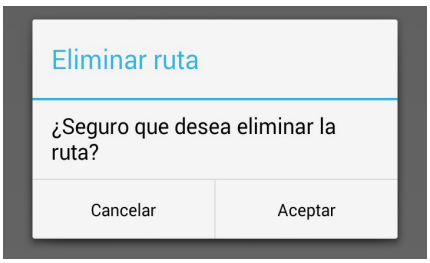

Imagen 50. Manual 9.

DATOS DEL ATLETA: Se trata de una pantalla para insertar en la aplicación distintos datos personales y físicos del usuario. A continuación se muestra la pantalla antes de insertar algún dato y una vez hecho. Cuando se pulsa sobre cada una de las entradas aparece un diálogo con un teclado específico para cada tipo de dato.

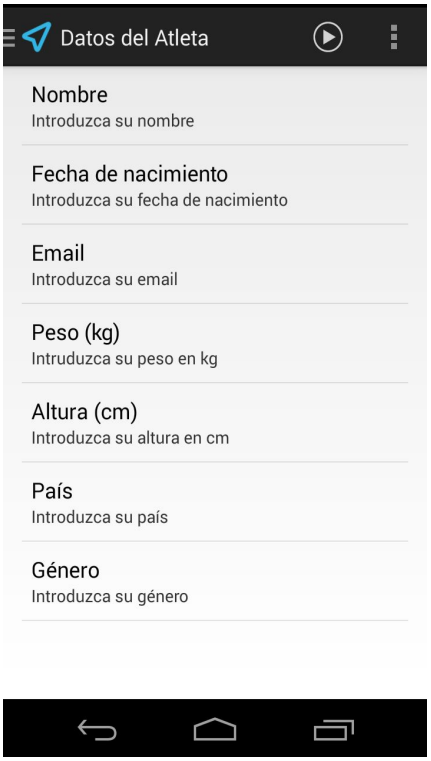

Imagen 51. Manual 10.

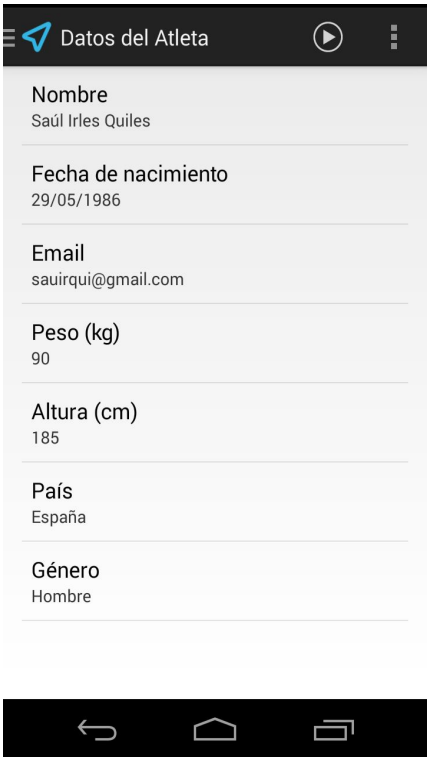

Imagen 52. Manual 11.

EXPLORA EL MAPA: Esta pantalla tiene la finalidad de analizar la zona por la que se va a pasar, antes de iniciar la actividad, y así poder planificar la ruta y evitar posibles obstáculos como por ejemplo los geográficos. Como añadidura contiene un botón para encontrar la posición del usuario y otro para hacer zoom in o zoom out sobre cualquier punto.

Imagen 53. Manual 12.

ACERCA DE: Contiene la información relevante sobre la versión de la aplicación, cómo fue desarrollada así como su programador, la última actualización producida y la fuente de los distintos iconos que se muestran en las pantallas.

Imagen 54. Manual 13.

Como ya se comentó anteriormente, para iniciar la actividad de grabación es necesario pulsar el botón de "*Play*" colocado en diversos sitios de la pantalla principal. Una vez es pulsado, si el terminal no tiene activado el receptor GPS saldrá un diálogo de advertencia desde donde se puede acceder al menú de configuración de dicho. En las imágenes siguientes se representa el caso.

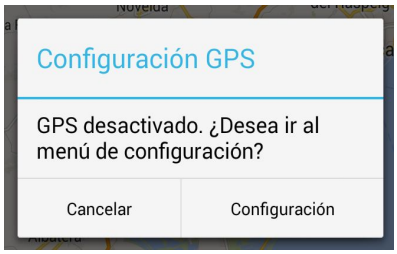

Imagen 55. Manual 14.

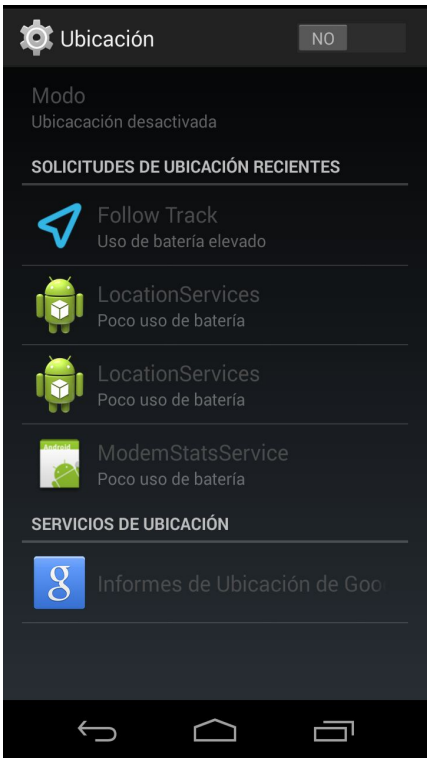

Imagen 56. Manual 15.

Una vez activado el GPS, irán mostrándose unos avisos instantáneos para confirmar los distintos estados por los que pasa el receptor.

Con todo listo y la actividad en curso, la pantalla representará el recorrido mediante una línea roja, la posición estática del usuario mediante un punto azul y la posición en movimiento mediante una punta de flecha también del mismo color que indica la dirección del desplazamiento. Los datos numéricos quedan bastante claros observando la imagen 57.

Para pausar la grabación de la actividad se seleccionará el botón de "*Pause/Stop*" situado en la esquina inferior derecha de la pantalla. Para volver se pulsará el botón de retroceso.

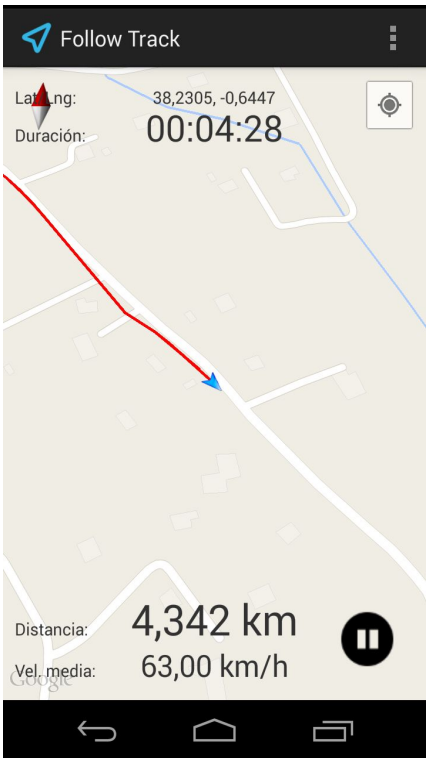

Imagen 57. Manual 16.

Es posible cambiar la capa del mapa accediendo desde el *overflow* del *ActionBar* a la opción "Tipo de mapa" que abre una ventana con las distintas posibilidades.

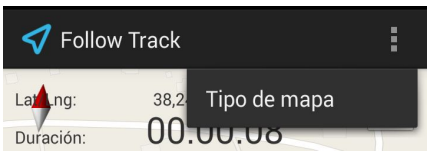

Imagen 58. Manual 17.

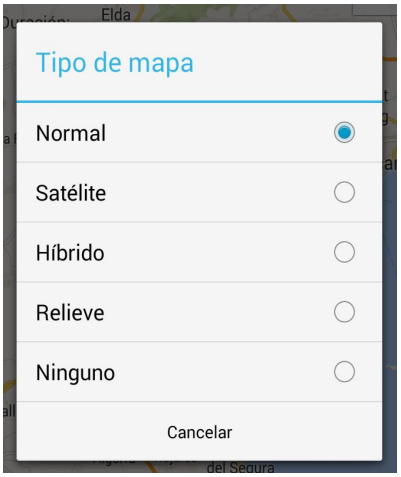

Imagen 59. Manual 18.

En el caso de que no se desee seguir ni guardar el recorrido, se pulsará el botón de retroceso y aparecerá un diálogo para confirmar el descarte. Cuando se pulse sobre "Aceptar" se volverá a la pantalla de inicio de la *app*.

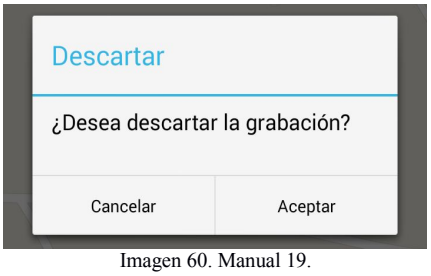

Imagen 60. Manual 19.

Una vez terminado el recorrido y pulsado el botón de "*Pause*", se abrirá una pantalla con un resumen de las estadísticas obtenidas durante la actividad. A través de la imagen siguiente se observa cuáles son:

Imagen 61. Manual 20.

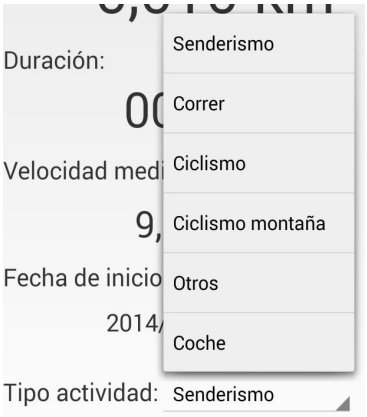

Imagen 62. Manual 21.

Mediante el *spinner* (lista desplegable) situado a continuación de la etiqueta "Tipo actividad" se seleccionará el tipo de actividad que se ha realizado. Obsérvese la imagen 62.

Los iconos en sendas esquinas inferiores son utilizados para salvar la actividad o para descartarla. Si se pulsa el botón con una "X" aparecerá un diálogo como el de la imagen 60 para no grabar la actividad, y si se pulsa sobre el botón con una "V" en su interior sí se almacenará el recorrido en la base de datos. En ambos casos se volverá a la pantalla principal de la aplicación.

A continuación se muestra la pantalla principal con algunas actividades realizadas y con el nombre del usuario establecido.

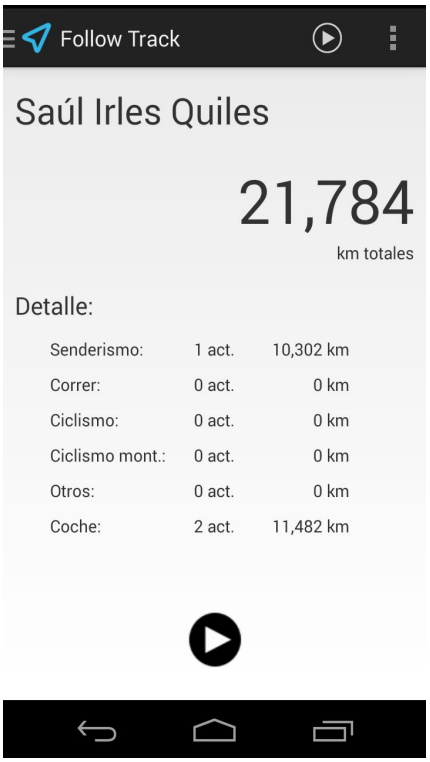

Imagen 63. Manual 22.

8. Bibliografía

[1] http://developer.android.com/intl/es/index.html, Android Developers.

[2] http://stackoverflow.com/, Stack Overflow.

[3] http://www.elandroidelibre.com/, El Androide Libre.

[4] http://www.xataka.com/, Xataka.

[5] http://www.sgoliver.net/, Sgoliver.

[6] http://www.androidhive.info/, Android Hive.

[7] http://www.androidcurso.com/, Curso libre Universidad Politécnica Valencia.

[8] http://telescopio.galileo.edu/, Curso libre Universidad Galileo.

[9] http://www.wikipedia.org/, Enciclopedia libre Wikipedia.

[10] http://www.vogella.com/, Tutoriales Android Vogella.

[11] http://notasprogramacion.sodenet.es/, Blog Android con ejemplos SQLite.

[12] http://geekytheory.com/, Tutoriales Android Geek Theory.

[13] http://www.tutorialspoint.com/sqlite/index.htm, información SQLite.

[14] https://github.com/, Repositorio GitHub de proyectos Android.

[15] "*Android for programmers and app-driven approach*", Paul Deitel, Harvey D., Abbey D., Michael Morgano, Deitel Developers Series, 2012.

[16] "*Proffesional Android 4 - Application Development*", Reto Meier, John Willey & Sons, Inc., 2012.

[17] "*Android Application Development CooKbook: 93 Recipes for Building Winning Apps*", Wei-Meng Lee, John Willey & Sons, Inc., 2013.

[18] "*Sams Teach Yourself Android Application Development in 24 Hours*", Lauren Darcey, Shane Conder, SAMS, 2010.