UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Aplicación móvil en Android para la gestión de entrenos de deportistas

Proyecto Final de Carrera

Ingeniería en Informática

Autor: Angel Patiño Diaz-Alejo

Director: José Vicente Busquets Mataix

Septiembre 2013

A mis padres Emilio y Rosa Mª por el esfuerzo y paciencia que han tenido conmigo.

A mi tía Nuria por preocuparse y estar siempre encima de mí para que finalizase con este proyecto los estudios iniciados.

A mis amigos por su ayuda prestada y su insistencia en finalizar esta etapa de la vida que parecía no tener fin.

En especial a mi madrina que allá donde esté pueda ver que su ahijado es un ingeniero.

Resumen

La aparición de los teléfonos inteligentes, comúnmente llamados smartphones, ha revolucionado la vida. Los teléfonos que antes solo se utilizaban para realizar llamadas y recibir mensajes se han convertido en una herramienta de trabajo y de ocio indispensable.

Un gran porcentaje de estos dispositivos móviles, hoy en día, llevan como sistema operativo Android.

Tras una breve introducción de la plataforma Android, con sus características y conceptos, se analizará, diseñará e implementará una aplicación para entrenos deportivos.

La aplicación está dirigida principalmente para actividades deportivas, como atletismo y/o ciclismo, aunque también podría ser utilizada para realizar senderismo o un simple paseo.

La función principal de la aplicación será mostrar el tiempo, la distancia recorrida y el consumo de calorías invertido en el desarrollo de la actividad deportiva en cuestión, así como su visualización en un mapa.

Palabras clave

Android, GPS, Geolocalización, aplicación móvil, deporte

Índice

Índice de ilustraciones

Índice de Tablas

1. Introducción

1.1 Una vista atrás

Afirmar, a estas alturas, que el mundo de las telecomunicaciones está en continuo cambio y evolución, no es decir nada nuevo. En efecto, basta con echar una rápida mirada atrás en el tiempo para darse cuenta que, desde que comenzaran a comercializarse a nivel global los primeros ordenadores personales (a finales de la década de los ochenta del pasado siglo), hasta la adquisición masiva de terminales móviles (también, a finales de la década de los noventa), la tecnología implícita en estos dispositivos no ha hecho sino avanzar a pasos agigantados.

Si bien dicha evolución ha quedado recogida de forma diferente según el producto al que se aplique (electrodomésticos, equipos audiovisuales, etc.), no es menos cierto que, a nivel informático, los avances que ha experimentado el sector de las telecomunicaciones ha sido descomunal. Cierto es que los primeros terminales móviles que se comercializaron resultaban una combinación tan curiosa como acertada de un teléfono tradicional con una agenda electrónica pues, a las prestaciones del establecimiento de llamadas y de envío de mensajes SMS se les unían otras funciones tales como registro de eventos en un calendario, activación de alarmas, o utilización de calculadora.

Sin embargo, con el paso de los años y con el fin de siglo cada vez más cercano, no resultaba raro comprobar que cada vez era mayor el número de terminales que comenzaron a incluir otra serie de extras como, por ejemplo, sofisticadas cámaras fotográficas (en muchas ocasiones, con mayor calidad de resolución que las cámaras propiamente dichas), con capacidad para tomar tanto instantáneas como vídeos, mp3 por el que poder reproducir música (que quedaba interrumpida cuando el usuario recibía alguna llamada) o, incluso, en algunos casos, conexión a Internet.

Ahora bien, toda esta evolución que, en a penas, una década había ido desarrollándose con gran celeridad, dio un gran paso a finales del año 2007, cuando tuvo lugar la presentación del primer terminal móvil con Android, y otro paso aún más grande cuando, un año después, se empezaron a comercializar dichos aparatos para alcanzar, poco más de año y medio después, un nivel masivo de ventas a lo largo y ancho de todo el mundo.

1.2 Aparición de Android

La aparición del sistema operativo Android supuso un antes y un después en lo que a la utilización de terminales móviles se refiere. Incluso, tampoco sería exagerado afirmar que, junto con su competidor más directo de Apple, ha revolucionado la forma de entender la telefonía móvil. Donde antes el usuario sólo veía un instrumento a través del que poder realizar llamadas telefónicas (con el avance que supuso la incorporación de algunos elementos mencionados en el apartado anterior), ahora ese mismo usuario ha pasado a disponer de una herramienta potentísima que le permite seguir manteniendo las mismas propiedades básicas de cualquier teléfono móvil –recordemos que la principal finalidad sigue siendo la de poder establecer llamadas telefónicas-, a la vez que disponer de todo un mundo de aplicaciones a su alcance de forma fácil y rápidamente instalable.

Evidentemente, como cualquier sistema operativo, Android ha ido sufriendo variaciones a lo largo de los últimos años, de forma que, partiendo de propiedades más básicas como la aparición de los widgets en la pantalla principal de los terminales, soporte para Wifi o Bluetooth o de funciones sencillas como la organización en carpetas, se ha pasado a otras más avanzadas como el desbloqueo de los terminales por reconocimiento facial o la conexión de dispositivos USB.

Sin embargo, estas características que se han ido presentando paulatinamente con cada versión actualizada del sistema Android también han supuesto una revolución en lo que a la interacción del propio usuario con su entorno se refiere. A través de la creación y desarrollo de aplicaciones para esta plataforma, al individuo de a pie se le ofrece la posibilidad de tener presencia virtual en una cantidad gigantesca de servicios y productos y, a su vez, a numerosísimas empresas facilitarle sus servicios e información a dicho usuario. Y es, precisamente, en este ámbito, donde se focaliza la atención del presente proyecto.

1.3 Una nueva forma de conexión con el mundo

En efecto, tal y como comentábamos en el apartado precedente, el mundo de las aplicaciones creadas para el sistema Android (no necesariamente en exclusiva), permiten al usuario tener una presencia casi absoluta en multitud de servicios y productos y, a las empresas que los producen, acceder a estos usuarios a través de una ventana asombrosamente rápida, y más ventajosa de lo que jamás antes se podrían haber imaginado.

En primer lugar, las aplicaciones se pueden clasificar por tema o utilidad. Esto le permite al usuario filtrar de antemano el servicio al que desea acceder, pudiendo orientarse más hacia el objetivo de su búsqueda: finanzas, juegos, herramientas para el propio terminal móvil, noticias, salud, meteorología, etc. De esta forma, se produce una primera aproximación del individuo con ese

"mundo virtual" al que accede, y del que comienza a beneficiarse al ritmo que éste elija y con las preferencias que cada uno tenga.

En segundo lugar, es frecuente que la variedad de aplicaciones que se refieran a un mismo aspecto o tema sea elevada, hecho que puede resultar muy ventajoso para el usuario. De esta forma, se le ofrece la posibilidad de elegir entre los diferentes criterios, gustos o preferencias que cada aplicación contiene, de manera que se le dé a cada aplicación el uso que se considere oportuno.

En tercer lugar, cada vez es mayor el número de usuarios que disponen de conexión a la red de forma permanente (hecho del que se encargan con notable insistencia las compañías telefónicas con continuas promociones y campañas publicitarias), lo que les hace interesarse de forma implícita en las aplicaciones que se encuentren disponibles, y que les pueden servir con independencia de que se trate de servicios de ocio o útiles para su día a día. La ventaja de este último punto reside en que no es la empresa o marca la que debe esforzarse por llegar hasta el individuo, sino que es él quien accede a ellos de forma totalmente voluntaria.

Por supuesto, dejando de lado el caso de aquellas empresas únicas en su sector (no nos referimos tanto al producto, sino al poder diferenciador de su marca: casos como McDolands, Zara, Tous, Bankia…), encontramos un lado diferente de las aplicaciones basado en la utilidad de las mismas, con independencia del producto al que puedan estar vinculadas. Así pues, el usuario que, por ejemplo, desea conocer la predicción meteorológica, dispone de múltiples aplicaciones que le facilitan dicha información ("El tiempo 14 días", "Tiempo AEMET"…), al igual que si se interesa por el horario de transportes de su ciudad ("EMTValencia", "Metro Valencia", etc.).

1.4 Planteamiento de los fundamentos del proyecto

Estos últimos ejemplos de empresas e instituciones que se sirven de las aplicaciones para llegar al usuario final son los que nos han llamado la atención a la hora de plantear el presente estudio. Desde el inicio, hemos querido centrarnos en una aplicación que pudiera estar disponible para cualquier terminal con sistema Android, al contar con una elevadísima expansión y aceptación por parte de los consumidores españoles. No obstante, a la hora de seleccionar el nicho de aplicaciones en el que centrar la atención, nos hemos decantado por escoger un sector que, desde hace unos pocos años, ha experimentado un notable crecimiento y aumento del interés por parte de los ciudadanos: la práctica del deporte.

Es verdad que, planteado de esta forma, la elección del sujeto de estudio pueda parecer un tanto banal y, hasta cierto punto, aleatorio. Sin embargo, nada más lejos de la realidad. Basta tener en cuenta el creciente interés por deportes que, hasta hace unos años, no contaban con el soporte popular que tienen hoy en día (la Fórmula 1, pádel, por poner dos ejemplos), y de la "modernización" de los

usuarios a la hora de practicar su deporte favorito, o de utilizar nuevas herramientas para comenzar a dedicarse a actividades nuevas. De tal forma, aquellos ciudadanos que corren o, simplemente, se montan en la bici y realizan sus propias rutas, suponían un público objetivo extraordinario en el que poder basar el desarrollo de nuestro proyecto.

Por consiguiente, lo que se describe en este documento es la creación de una aplicación orientada en especial hacia aquellos usuarios de terminales móviles con Android, que deseen servirse de su teléfono como complemento a la hora de realizar una variada actividad deportiva, a la que hemos bautizado como **SPORT TRACKING**.

Ilustración 1 - Icono de la aplicación

1.5 Objetivos y motivación

El mercado de los dispositivos móviles está dominando a nivel de plataforma por dos grandes compañías, Google con su Android y Apple con su IOS. Fuera de este proyecto queda la comparación de ambos sistemas, con sus ventajas e inconvenientes.

Dada la expansión y la gran cantidad de dispositivos con Android, el ser una plataforma de código abierto, la facilidad y disposición de herramientas para desarrollar aplicaciones en su entorno nos ha decantado en su elección. No por ello queremos indicar que sea mejor ni peor.

Además parece ser que el mercado de móviles tiende a ser dominado principalmente por Android. De su uso principalmente en dispositivos móviles de usuarios domésticos se ha pasado a usarse en terminales industriales.

Es por todo esto y por las oportunidades futuras que nos ofrece que uno de los principales objetivos es conocer la plataforma Android y sus principales características.

Habiendo programado hasta ahora principalmente en .Net y en dispositivos móviles, actualmente en desuso, con Windows Mobile, programar en Android sobre Java abre una nueva puerta y enriquece nuestros conocimientos y formación.

También se pretende poder aplicar los conocimientos adquiridos en la carrera de ingeniería informática y culminarnos en un proyecto real, en todas las fases

que componen el desarrollo de una aplicación: el análisis, el diseño, la implementación y las pruebas.

De las muchas características que tienen hoy en día los dispositivos móviles, y otras que nos puede llegar a ofrecer la plataforma Android la aplicación a desarrollar deberá hacer uso principalmente de la geoposición mediante GPS, el uso del API de Google Maps y el almacenamiento en una base de datos.

Una vez finalizado el proyecto deberemos ser capaces de tener los conocimientos suficientes para poder desarrollar aplicaciones futuras estables y eficientes en Android que puedan aprovechar todo su potencial y características.

1.6 Metodología

La metodología en la que nos hemos basado para realización del desarrollo de software ha sido el modelo en espiral ya que reduce riesgos y permite introducir mejoras y nuevos requerimientos durante el proyecto.

Normalmente el proyecto se divide en módulos más pequeños y a cada unos de ellos se le aplica el siguiente proceso:

1. Análisis.

   Durante esta etapa de estudia detalladamente los requerimientos que cada objetivo conlleva, se identifican los riesgos y se determinarán las posibles alternativas para solucionarlos. En esta etapa se establecen todos los detalles funcionales deseados.

2. Diseño.

   En este paso, con los datos de la etapa anterior, se diseña el sistema. Se realiza un diseño, por ejemplo, de la base de datos, interface, entorno,

3. Implementación.

   Este tercer paso comprende la programación y test unitarios después de haber elegido el modelo de desarrollo oportuno.

4. Testing

   En este último paso es donde el proyecto se revisa, se realiza un test del módulo completo así como su evaluación frente al estudio de requerimientos y se toma la decisión si se debe continuar con un ciclo posterior al de la espiral. Si se decide continuar, se desarrollan los planes para la siguiente fase del proyecto.

Con cada iteración alrededor de la espiral, se crean sucesivas versiones del software, cada vez más completas y, al final, el sistema de software ya queda totalmente funcional

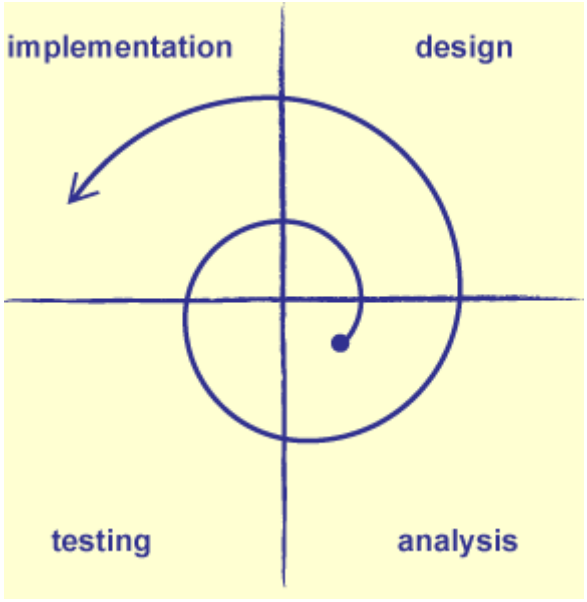

Ilustración 2 - Metodología: Modelo en espiral

1.6 Estructura del proyecto

En el proyecto podemos encontrar principalmente 6 bloques que se detallan a continuación:

1. Introducción: En esta primera parte se hará una breve introducción de Android, se comentarán sus principales características y arquitectura. A la hora de desarrollar una aplicación se tiene que tener en cuenta los principales componentes que forman parte de una aplicación Android, así como el ciclo de vida de una "pantalla" y las distintas versiones que nos podemos encontrar.

2. Análisis: Después de la introducción, se procederá al análisis de la aplicación que se va a desarrollar. En este parte nos encontraremos los requerimientos tanto funcionales como no funcionales y los casos de uso. Junto a estos últimos se incluyen las plantillas para su mayor compresión.

3. Diseño: Una vez analizada la aplicación el siguiente bloque será el diseño. En esta parte se detallará los diagramas de clases que componen la aplicación, los diagramas de secuencia, la estructura de las tablas que forman la base de datos que se utilizará, el diseño de la interfaz gráfica de las pantallas y la arquitectura que presentará la aplicación.

4. Implementación: En esta parte se mostrará el entorno hardware y software utilizado, explicando la configuración de este último. Se

detallará la estructura que tiene un proyecto Android en el entorno de programación. Aquí se incluirá y comentará las partes de código de las funcionalidades más relevantes.

5. Pruebas y Conclusiones: Una vez finalizada la implementación, se realizan las pruebas que se detallarán en esta parte para comprobar el correcto funcionamiento de la aplicación. Como finalización se detallan las conclusiones obtenidas y si se han cumplido los objetivos planteados.

Como documentación final en el proyecto se incluirá un manual de la aplicación, donde se detallará la instalación y funcionamiento de la aplicación desarrollada con todos los aspectos que deben ser tenidos en cuenta.

2. Android

2.1 ¿Qué es Android?

En pocas palabras, Android es un sistema operativo orientado a hardware móvil, como teléfonos y otros dispositivos informáticos limitados, tales como netbooks y tabletas.

El concepto y la plataforma fue una idea original de Android Inc., una pequeña empresa de Palo Alto, California, que fue adquirida por Google en 2005. Su objetivo era crear un sistema operativo pequeño, estable, flexible y fácil de actualizar para dispositivos que sería muy atractivo para los fabricantes de dispositivos y para las operadoras de telefonía.

La plataforma Android fue presentada en noviembre de 2007. La inauguración coincidió con el anuncio de la formación de la Open Handset Alliance, un grupo de empresas cuyo objetivo principal es promover estándares abiertos para plataformas de dispositivos móviles como es Android.

En octubre de 2008, Android fue lanzado bajo la licencia de código abierto Apache 2.0. Esto y su diseño flexible de los componentes en que está basado ofrece oportunidades innovadoras y rentables para los fabricantes y desarrolladores de software por igual.

2.2 Plataforma Android

Android es un entorno de software creado para dispositivos móviles. Incluye un núcleo basado en el sistema operativo Linux, una completa interfaz de usuario, aplicaciones finales de usuario, bibliotecas de código, aplicaciones frameworks, soporte multimedia, y mucho más. Mientras que los componentes del sistema operativo están escritos en C ó C++, las aplicaciones de usuario para Android se escriben en Java. Incluso las aplicaciones propias incorporadas están desarrolladas en Java.

Una de las características de la plataforma Android es que no hay diferencia entre las aplicaciones que incorpora y las aplicaciones que se pueden desarrollar con el SDK. Esto significa que es posible crear aplicaciones que aprovechan todo el potencial de los recursos disponibles en el dispositivo. La característica más notable de Android podría ser que es de código abierto, y los elementos que le falten pueden o serán desarrollados por la comunidad global de programadores. El núcleo del sistema operativo basado en Linux no incluye un sofisticado intérprete de comandos o shell, pero, en parte, porque la plataforma es de código abierto y tú puedes desarrollar o instalar un shell en el dispositivo. Del mismo modo, los codecs multimedia, por ejemplo, pueden ser suministrados por desarrolladores de terceras partes y no depender de Google para proporcionar una nueva funcionalidad. Esto es el poder de la introducción de una plataforma de código abierto en el mercado móvil.

2.3 Arquitectura Android

La pila de Android incluye una impresionante variedad de características para las aplicaciones móviles. De hecho, mirando a la arquitectura solo, sin el contexto de que Android es una plataforma diseñada para entornos móviles, sería fácil confundir Android con un entorno informático en general. Todos los componentes principales de una plataforma de computación están ahí.

Por ejemplo, los principales componentes de la arquitectura de Android serían:

- Un núcleo Linux proporciona una capa fundamental de abstracción del hardware, así como servicios básicos, tales como gestión de procesos, memoria y sistema de archivos. En el núcleo es donde están implementados los drivers específicos del hardware, para Wi-Fi o Bluetooh por ejemplo. La pila de Android está diseñado para ser flexible con muchos otros componentes opcionales que dependerán en gran medida de la disponibilidad de hardware específico en un dispositivo determinado. Estos componentes incluyen características tales como el tacto en las pantallas, cámaras, receptores GPS y acelerómetros.

- Importantes librerías que incluyen:

  - La tecnología de navegación de WebKit, el mismo motor de código abierto del navegador Safari de Mac e iPhone. WebKit se ha convertido, de hecho, en el estándar para la mayoría de las plataformas móviles.
  - Soporte para acceso de base de datos SQLite, una base de datos liviana y de fácil uso.
  - Soporte de gráficos avanzados, incluyendo 2D, 3D, motor gráfico SGL y OpenGL ES.
  - Soporte de audio y vídeo de OpenCORE de PacketVideo y un framework multimedia propio de Google, Stagefright.
  - Protocolo Secure Sockets Layer (SSL) de Apache.

- Entorno de ejecución. Lo constituyen las Core Libraries, que son librerías con multitud de clases de Java, y la máquina virtual Dalvik. Las aplicaciones básicas y aplicaciones de terceros (que puede crear cualquier desarrollador) se ejecuta en la máquina virtual Dalvik sobre los componentes descritos.

- Framework de aplicaciones. Representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo framework, representado por este nivel.

- El último nivel del diseño arquitectónico de Android son las aplicaciones. Éste nivel incluye tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o que vaya desarrollando él. Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores.

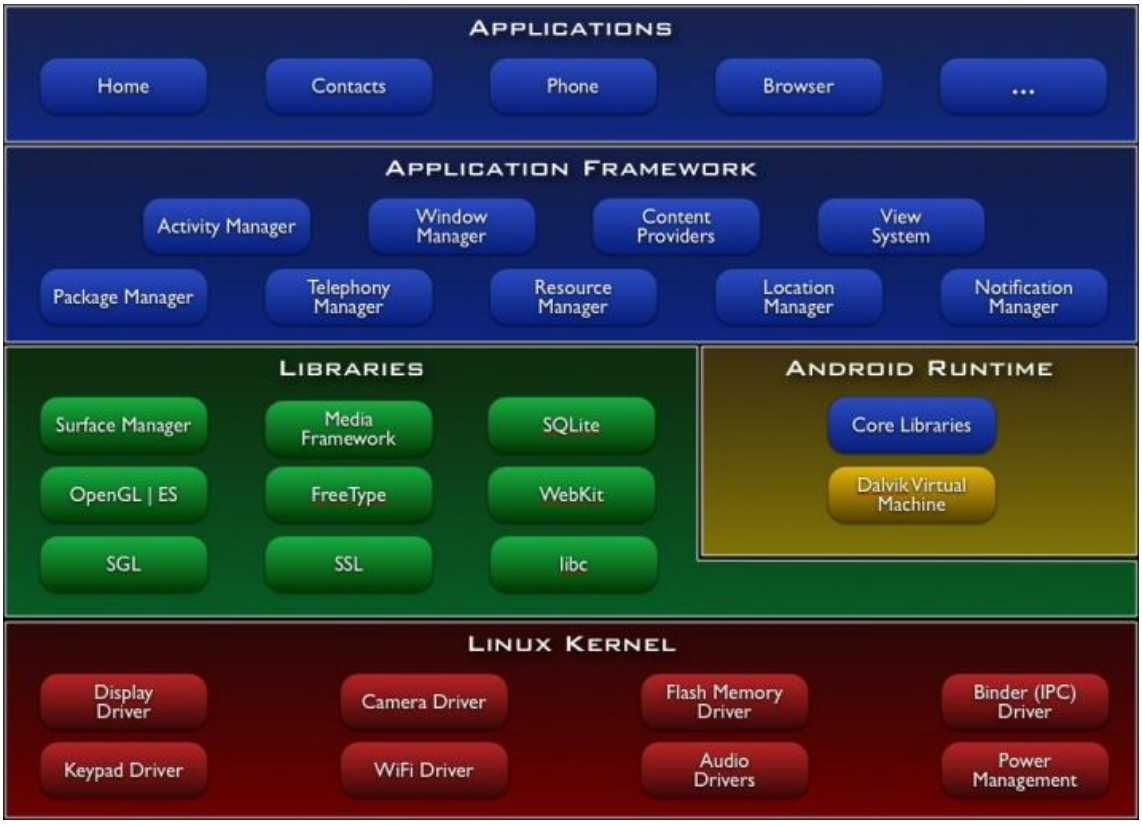

Ilustración 3 - Arquitectura Android

2.4 Conceptos básicos

A continuación se enumerarán los conceptos básicos que hay que tener en cuenta en el desarrollo de toda aplicación Android y que se harán referencia de ahora en adelante:

Activity

Las actividades (activities) representan el componente principal de la interfaz gráfica de una aplicación Android, contiene la interface de usuario de nuestra aplicación. Correspondería a las ventanas de cualquier otro lenguaje de programación. Desde el momento que una activity se muestra en pantalla hasta el momento que se destruye, pasa por varias etapas, conocido como el ciclo de vida de una activity.

View

Los objetos views son los componentes básicos con lo que se crea la interfaz gráfica de la aplicación (activity). Por paralelisimo viene a ser los controles de java o .NET. Android pone a disposición del usuario una gran cantidad de controles básicos: botones, imágenes, listas desplegable, etc. Aunque también se tiene la posibilidad de crear nuevos controles personalizados, modificar o ampliar la funcionalidad de los ya existentes.

Service

Son componentes sin interfaz gráfica que nos permiten realizar tareas duraderas en un segundo plano. Vienen a ser como los servicios en cualquier sistema operativo. Los service pueden ser de dos tipos:

- Started: Son iniciados por algún componente y su ejecución es independiente del componente que lo ha iniciado. Pueden continuar ejecutándose aunque el componente que lo inicio ya no exista.
- Bound: Son iniciados cuando algún componente se vincula a este service, comportándose como una especie de cliente-servidor permitiendo a los componentes interaccionar con el service. Es posible vincular varios componentes a un mismo service, pero estos services se destruirán cuando el componente al que estaba vinculado se destruya.

Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales (*activities*) si se necesitase una interacción con el usuario en un momento determinado.

Content Provider

Un *content provider* es el mecanismo que se ha definido en Android para tener acceso a datos estructurados. Además de encapsular los datos, mediante estos componentes es posible compartir determinados datos, definiendo la seguridad, de nuestra aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, nuestra aplicación podrá acceder a los datos de otra a través de los *content provider* que se hayan definido.

Broadcast Receiver

Un broadcast receiver es un componente permite que se reciban mensajes o eventos generados por el sistema o por otras aplicaciones. El componente que envía la notificación lo hace a través de un sendbroadcast. Los componentes que la reciben se asocian o suscriben a un Receiver. No están dirigidos a una aplicación en concreto sino a cualquiera que desee hacer uso de los eventos o mensajes enviados. Por ejemplo: batería baja, llamada entrante, etc.

Widget

Los *widgets* son componentes visuales, normalmente interactivos, que normalmente se muestran la pantalla principal (*home screen*) del dispositivo Android. Suelen mostrar cierta información, pudiendo actualizarse periódicamente y permitiendo al usuario realizar ciertas acciones.

Intent

Un *intent* es el elemento básico de comunicación entre los distintos componentes Android que han sido descritos. Se pueden entender como los *mensajes* o *peticiones* que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un intent se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etc.

2.5 Versión Android

La plataforma Android proporciona un framework API que las aplicaciones pueden utilizar para interactuar con el sistema Android. El framework API consta de:

• Un conjunto básico de paquetes y clases

• Un conjunto de elementos y atributos XML declarados en un archivo manifest

• Un conjunto de elementos y atributos XML para la declaración y el acceso a los recursos

• Un conjunto de Intents

• Un conjunto de permisos que las aplicaciones pueden solicitar, así como permisos de aplicación incluidos en el sistema.

Cada versión sucesiva de la plataforma Android puede incluir actualizaciones en el framework API de aplicación que proporciona.

Las actualizaciones del framework API se han diseñado para que el nuevo API siga siendo compatible con las versiones anteriores del API. Es decir, la mayoría de los cambios en la API son añadidos e introducen nuevas funcionalidades o reemplazo de las existentes. Como parte del API se actualiza, las partes más antiguas reemplazados están en desuso, pero no se eliminan, por lo que las aplicaciones existentes todavía pueden hacer uso de ella. En un número muy pequeño de casos, partes de la API pueden ser modificados o eliminados, aunque por lo general estos cambios sólo son necesarios para asegurar la solidez del API y la seguridad del sistema. Todas las demás partes del API de versiones anteriores se mantienen sin modificaciones en las versiones posteriores.

El nivel de API declarado en la aplicación es muy importante por razones de compatibilidad del dispositivo y el tiempo de vida de desarrollo y mantenimiento del software.

En la tabla siguiente se especifica el nivel de API con el apoyo de cada versión de la plataforma Android.

Versión	API	Nombre
Android 4.3	18	Jelly Bean
Android 4.2, 4.2.2	17	Jelly Bean
Android 4.1, 4.1.1	16	Jelly Bean
Android 4.0.3, 4.0.4	15	Ice Cream Sandwich
Android 4.0, 4.0.1, 4.0.2	14	Ice Cream Sandwich
Android 3.2	13	Honeycomb
Android 3.1.x	12	Honeycomb
Android 3.0.x	11	Honeycomb
Android 2.3.4 Android 2.3.3	10	Gingerbread
Android 2.3.2 Android 2.3.1 Android 2.3	9	Gingerbread
Android 2.2.x	8	Froyo
Android 2.1.x	7	Eclair
Android 2.0.1	6	Eclair
Android 2.0	5	Eclair
Android 1.6	4	Donut
Android 1.5	3	Cupcake
Android 1.1	2	Banana Bread
Android 1.0	1	Apple Pie

Tabla 1 - Versiones de Android

2.6 Ciclo de vida

Android es una plataforma diseñada para dispositivos móviles. La pantalla, la batería y los recursos disponibles, entre otros aspectos, en los dispositivos nos condicionarán a la hora de desarrollar las aplicaciones.

La mayoría de los dispositivos Android son equipos de comunicación, teléfonos en los que es posible realizar y recibir llamadas. Un usuario no espera cerrar una aplicación para contestar una llamada, espera que de forma automática se presente el interfaz del teléfono que le permita atenderla. Es por esto que el ciclo

de vida de las activities estará condicionado por la interacción del usuario, así como otros posibles eventos que puedan ocurrir.

A diferencia de lo sucede en los ordenadores personales, el usuario únicamente lanza aplicaciones, no tiene la opción de finalizarlas. Es el propio sistema el que se encargará de finalizarlas cuando necesite recursos, pero cuando el usuario vuelve a la aplicación espera que esta se encuentre en el mismo estado en que la dejo. Así que las aplicaciones tienen un ciclo de vida que está controlado por el usuario y el sistema.

Las interfaces de usuario donde las ventanas se solapan han tenido una gran aceptación en los ordenadores personales, pero no son adecuadas para los dispositivos móviles donde la pantalla es reducida y debe primar la sencillez en la gestión gráfica. Por lo general Android utiliza una interfaz que ocupa toda la pantalla, aunque es posible mostrar notificaciones o diálogos que permiten ver parcialmente la pantalla sobre la que se han ejecutado.

La interfaz de una aplicación estará formada por un conjunto de pantallas que permitirán interacción con el usuario.

Por motivos de seguridad, cada aplicación de android se ejecuta en su propio proceso que además corresponde a un usuario diferente de Linux. Es posible compartir pantallas entre aplicaciones, el sistema permite que desde una aplicación mostrar una pantalla de otra aplicación. Por ejemplo, si se tuviese un mapa que muestre determinada información no es necesario que se implemente en cada aplicación que haga uso de dicho mapa.

En todo momento el usuario puede pulsar el botón del menú hacia atrás que le permite volver a la pantalla previa. Desde el punto de vista del usuario, una aplicación está formada por una pila de pantallas abiertas.

Android puede en cualquier momento pausar, parar, destruir nuestra aplicación según las necesidades de recursos del momento y el desarrollador debe controlar todos estos eventos para hacer una aplicación estable, eficiente y transparente al usuario.

De este modo, una activity puede pasar por los siguientes estados:

- onCreate(): Este método se llama cuando la activity es creada. Es donde se suelen realizar las inicializaciones, configurar la interfaz de usuario o crear adaptadores. Puede recibir como parámetro el estado anterior de la actividad para que podamos restaurarla. Después de esta llamada siempre se llama a onStart().

- onRestart(): Llamada cuando tu actividad ha sido parada, antes de volver a ser reanudada. Siempre viene después un onStart().

- onStart(): Llamada justo antes de que la actividad vaya a ser visible por el usuario. Después se pasará a onResume() si la actividad va a ser visible para interactuar con el usuario o a onStop() si no es mostrada al usuario.

- onResume(): Se ejecuta en el momento en que la actividad se encuentra en la parte superior de la pila y la activity interactúa con el usuario. Después se pasará a onPause().

- onPause(): Se llama cuando el sistema va a empezar una nueva activity, es decir, cuando se ha llamado al onRestart() de otra. En este método se debe aprovechar para liberar recursos y guardar datos de forma persistente. No deberá contener tareas lentas ya que hasta que no se ejecuta este método no empieza el siguiente de la nueva actvivity(onResume). El siguiente método es onResume() si la activity vuelve a primer plano o onStop() si se hace invisible al usuario.

- onStop(): Se ejecuta cuando la activity deja de ser visible al usuario. Puede ser porque otra activity pase a primer plano, un lugar más prioritario en la pila. El siguiente método será onRestart() si la activity vuelve para interaccionar con el usuario o onDestroy() si es destruida completamente.

- onDestroy() : Es la última llamada antes de destruir la Activity. Durante la destrucción de la actividad se perderán todos los datos asociados a la misma, por lo que se puede controlar la persistencia de los mismos. Puede ocurrir porque se realiza una llamada finish() en la activity o porque el sistema la elimina ya que necesita recursos. Si hay poca memoria es posible destruir la activity sin pasar por este método.

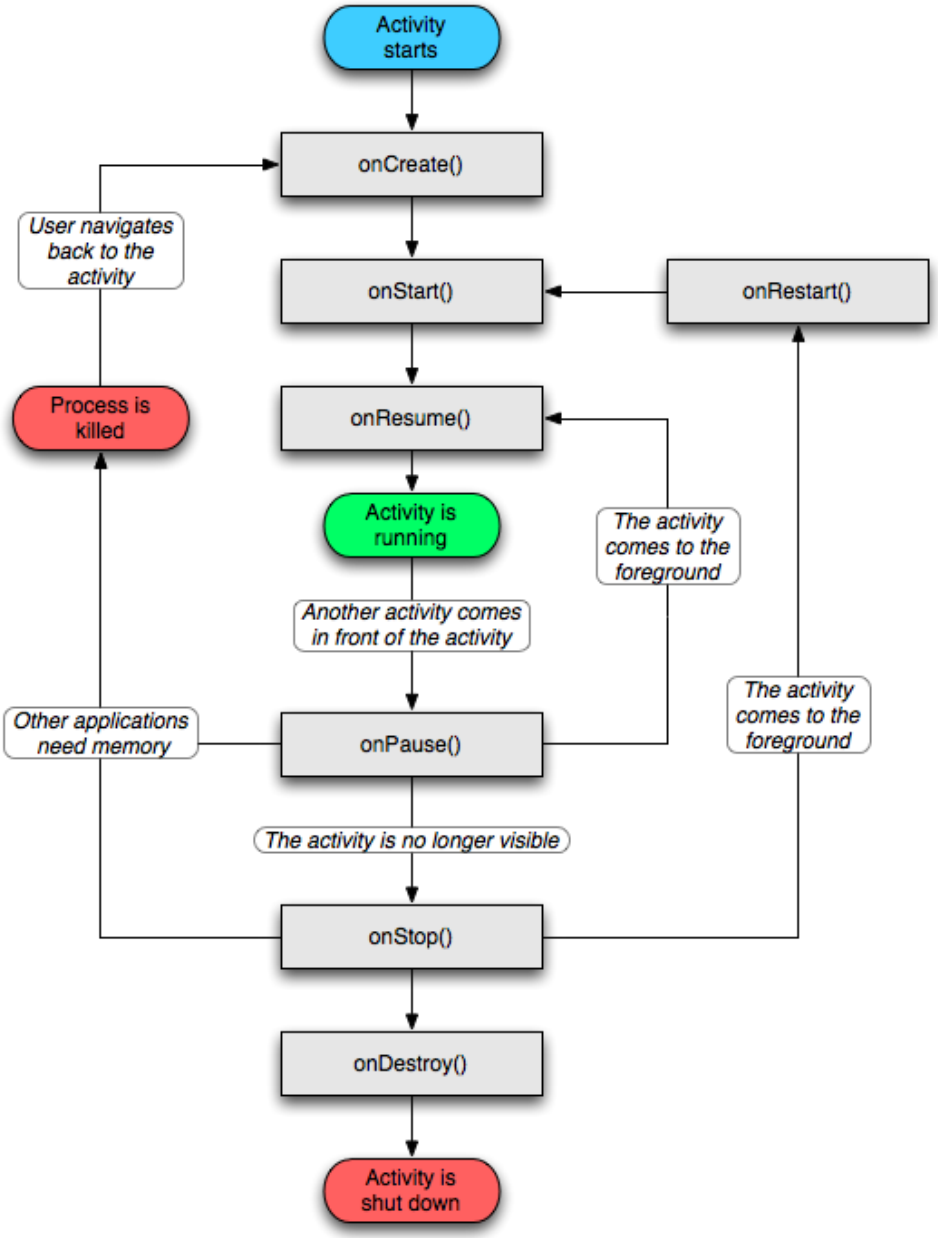

Ilustración 4 - Ciclo de vida de una Activity

3. Análisis

3.1 Requisitos funcionales

A continuación se detallarán los requisitos funcionales.

Requisito Funcional 1: Al entrar en la aplicación se mostrarán las opciones disponibles “Recorrido”, “Histórico”, “Configuración” y “Salir”

Requisito Funcional 2: Al seleccionar “Configuración” se debe poder configurar la aplicación con su peso (necesario para calcular posteriormente el consumo de calorías) y si desea utilizar como conexión a internet solo la conexión Wi-Fi y no el plan de datos del dispositivo.

Requisito Funcional 3: Al entrar en “Recorrido” debe poder seleccionar el tipo de recorrido que va a realizar, es decir, si va a correr, caminar o ir en bicicleta.

Requisito Funcional 4: Si se inicia un recorrido y el GPS no está activo se debe tener la posibilidad de encenderlo.

Requisito Funcional 5: Una vez iniciado un nuevo recorrido se mostrar el tiempo que lleva activo, la velocidad a la que va, la distancia recorrida y el consumo de calorías. También se mostrará si el GPS está activo, y si fuese así la precisión del mismo. Se debe poder pausar el tiempo del recorrido.

Requisito Funcional 6: Se debe poder activar un mapa y comprobar el punto en el que se encuentra y el recorrido que se ha realizado hasta el momento.

Requisito Funcional 7: Si se ha configurado “Solo conexión Wi-Fi” y no existe tal conexión no se mostrarán mapas de Google Maps debido al consumo de datos que tiene sobre la tarifa de datos.

Requisito Funcional 9: Al parar o apretar el botón “Atrás” del dispositivo se debe poder guardar o descartar el recorrido realizado.

Requisito Funcional 10: Se debe poder listar todos los recorridos realizados y guardados. Del mismo modo se debe poder eliminar uno, varios o todos los recorridos realizados.

Requisito Funcional 11: Se debe poder consultar un recorrido viendo todos los datos almacenados: tipo, distancia, velocidad, duración y su visualización en un mapa.

Requisito Funcional 12: Se debe poder comparar dos recorridos realizados, viendo todos los datos almacenados de ambos recorridos y su visualización superpuesta en un mapa.

Requisito Funcional 13: Las velocidades para recorridos realizados a pie o caminando se mostrarán en metros por minuto. En cambio, en bicicleta se mostrarán en kilómetros por hora.

3.1 Requisitos no funcionales

A continuación se detallarán los requisitos no funcionales.

Requisito no Funcional 1: El dispositivo debe tener GPS, tenerlo activo y con una localización válida para que se muestre la distancia, velocidad y consumo, así como el posicionamiento y recorrido que se está realizando en un mapa.

Requisito no Funcional 2: El dispositivo debe tener conexión a internet para visualizar el recorrido realizado o que se está realizando sobre un mapa de Google.

Requisito no Funcional 3: El dispositivo debe tener una versión Android 3.0 o superior, una pantalla de 4" o mayor, un procesador 800MHz o superior y 512 MB o más de memoria RAM.

3.2 Casos de uso

Mediante el diagrama de casos de uso mostraremos de una forma gráfica la funcionalidad de todo el sistema y como el usuario de la aplicación (Actor) interacciona con el mismo.

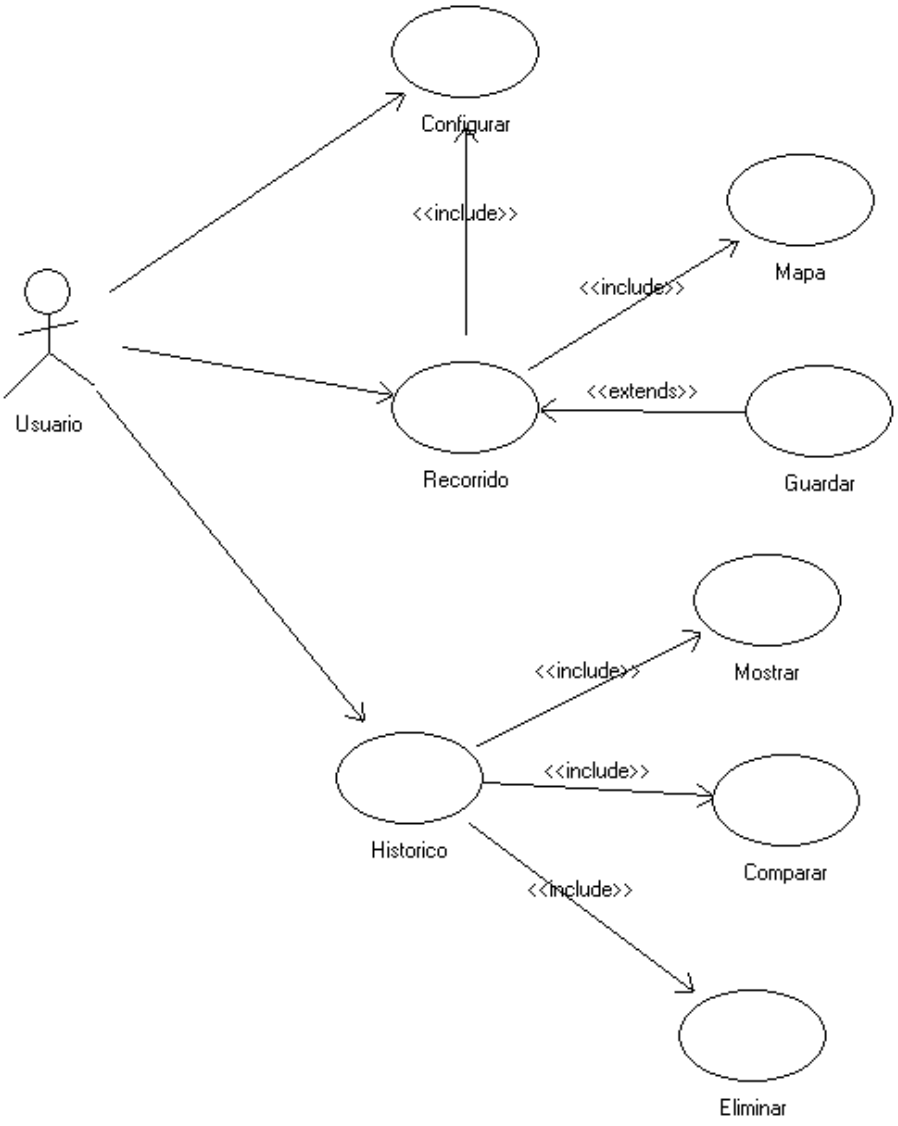

Ilustración 5 - Diagrama de casos de us

Para una mayor compresión del diagrama de casos de uso a continuación se detallan las plantillas de los diferentes casos de uso.

Nombre: Configurar
Descripción: Permite al usuario configurar los parámetros de la aplicación
Actores: Usuario
Precondición: No aplicable
Postcondición: El usuario ha configurado la aplicación
Flujo Principal: 1) El usuario solicita configurar la aplicación 2) El sistema muestra los parámetros de configuración 3) El usuario introduce los parámetros de configuración 4) El sistema valida los campos introducidos 5) El caso de uso finaliza
Flujo Alternativo: 4a) El sistema muestra un mensaje de error si faltan campos o los valores no son correctos.

Nombre: Recorrido
Descripción: Permite al usuario iniciar un nuevo recorrido
Actores: Usuario
Precondición: El usuario ha configurado previamente el sistema
Postcondición: El usuario realiza un recorrido
Flujo Principal: 1) El usuario solicita recorrido 2) El sistema muestra el tipo de recorrido a realizar (correr, bicicleta, caminar) 3) El usuario selecciona el tipo de recorrido. 4) El usuario inicia el recorrido. 5) El usuario finaliza el recorrido 6) El caso de uso finaliza
Flujo Alternativo: 3.a)El sistema muestra el mensaje “GPS desactivado” El usuario puede activar la señal GPS o iniciar el recorrido sin señal GPS 4.a) El usuario pausa el recorrido El usuario puede volver a iniciar el recorrido o salir del recorrido 4.b)El sistema muestra el mensaje “Sin Señal GPS”

Nombre: Mapa
Descripción: El usuario desea visualizar su posición y el recorrido que está realizando hasta el momento en un mapa
Actores: Usuario
Precondición: El usuario ha iniciado un recorrido
Postcondición: Se muestra en un mapa la posición actual y recorrido realizado
Flujo Principal: 1) El usuario solicita mostrar el mapa. 2) El sistema muestra en un mapa la posición actual y el recorrido que se está realizando. 3) El caso de uso finaliza
Flujo Alternativo: 2a) El sistema muestra el mensaje "No existe conexión de internet disponible". No se mostrará el recorrido y la posición actual en un mapa. 2b) El GPS no está activado. Se mostrará un mapa centrado en España. 2c) El mapa ya está activo mostrando la posición actual y el recorrido realizado. Se ocultará el mapa

Nombre: Guardar
Descripción: El usuario guarda en la base de datos un recorrido realizado
Actores: Usuario
Precondición: El usuario ha realizado un recorrido
Postcondición: El recorrido es almacenado en la base de datos del sistema
Flujo Principal: 1) El sistema solicita confirmación para almacenar el recorrido. 2) Se almacena el recorrido en la base de datos. 3) El caso de uso finaliza
Flujo Alternativo: 1a) El usuario descarta almacenar el recorrido. El caso de uso finaliza.

Nombre: Histórico
Descripción: Permite al usuario listar los recorridos almacenados en la base de datos
Actores: Usuario
Precondición:
Postcondición: Se listan los recorridos almacenados en la base de datos
Flujo Principal: 1) El usuario solicita listar los recorridos almacenados 2) El sistema muestra los recorridos almacenados en el sistema 3) El caso de uso finaliza
Flujo Alternativo:

Nombre: Mostrar
Descripción: Permite al usuario consultar un recorrido realizado mostrando toda la información almacenada
Actores: Usuario
Precondición: El usuario ha seleccionado un recorrido
Postcondición: Se muestra el recorrido almacenado en la base de datos
Flujo Principal: 1) El usuario solicita mostrar un recorrido 2) Se muestra el recorrido seleccionado 3) El caso de uso finaliza
Flujo Alternativo: 2a) El sistema muestra el mensaje “No existe conexión de internet disponible”. Se mostrará el recorrido exceptuando el recorrido en Google Maps. 2b) No se ha almacenado en la base de datos puntos GPS. Se mostrará los datos disponibles almacenados que no necesiten posicionamiento GPS para ser calculados.

Nombre: Comparar
Descripción: Permite al usuario realizar un comparativo de dos recorridos almacenados en la base de datos
Actores: Usuario
Precondición: El usuario ha seleccionado dos recorridos
Postcondición: Se muestran comparativamente dos recorridos almacenados en la base de datos previamente filtrados
Flujo Principal: 1) El usuario solicita comparar dos recorridos 2) Se muestra un comparativo de los recorridos seleccionados 3) El caso de uso finaliza
Flujo Alternativo: 2a) El sistema muestra el mensaje "No existe conexión de internet disponible". Se mostrará el comparativo de recorridos exceptuando el comparativo en Google Maps. 2b) No se ha almacenado en la base de datos puntos GPS. Se mostrará los datos disponibles almacenados que no necesiten posicionamiento GPS para ser calculados.

Nombre: Eliminar
Descripción: Permite al usuario eliminar uno o varios recorridos almacenados en la base de datos
Actores: Usuario
Precondición: El usuario ha seleccionado uno o varios recorridos almacenados en la base de datos previamente filtrados
Postcondición: Se eliminan los recorridos seleccionados
Flujo Principal: 1) El usuario solicita eliminar recorridos 2) El sistema solicita confirmación para eliminar los recorridos seleccionados 3) Los recorridos son eliminados de la base de datos 4) El caso de uso finaliza
Flujo Alternativo: 2a) El usuario cancela la confirmación de eliminar los recorridos. Finaliza el caso de uso

4. Diseño

4.1 Diagrama de clases

A continuación se muestra el diagrama de clases, que representar los objetos fundamentales del sistema, es decir los que acaba percibiendo el usuario y con los que espera interaccionar para tratar obtener el resultado deseado. Reflejan la estructura estática del sistema.

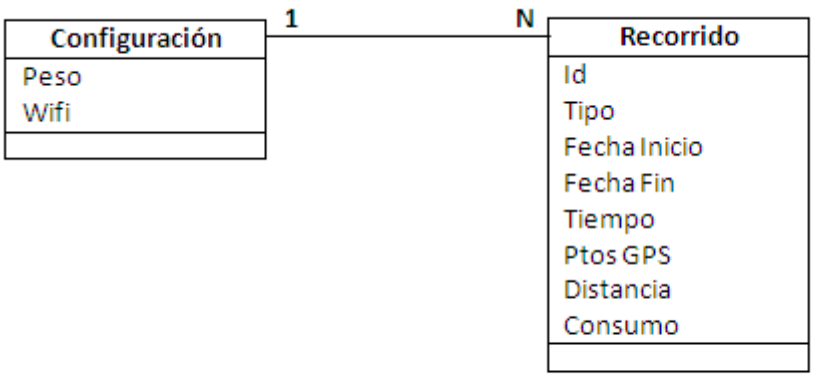

Ilustración 6 - Diagrama de clases

4.2 Diagrama de secuencia

Los objetos interactúan para realizar colectivamente los servicios ofrecidos por las aplicaciones. Los diagramas de secuencia muestran cómo se comunican los objetos en una interacción, es decir, muestran la secuencia de mensajes entre objetos durante un escenario concreto.

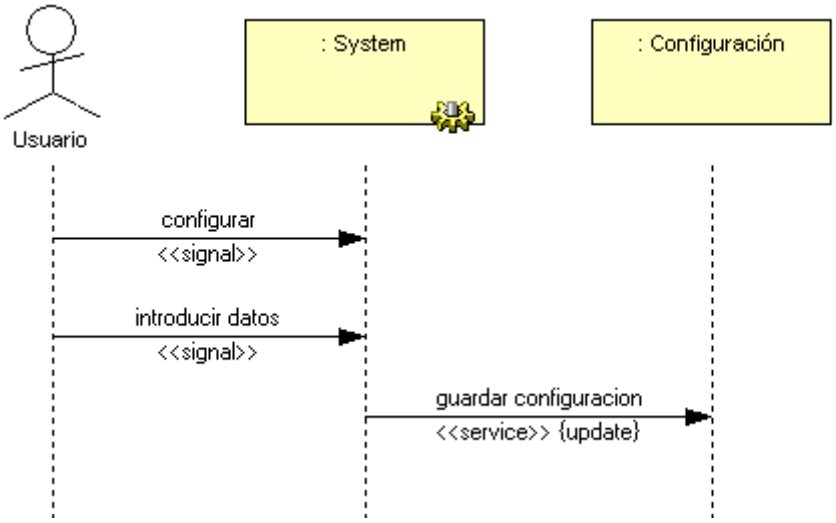

Ilustración 7 - Diagrama de secuencia Configurar

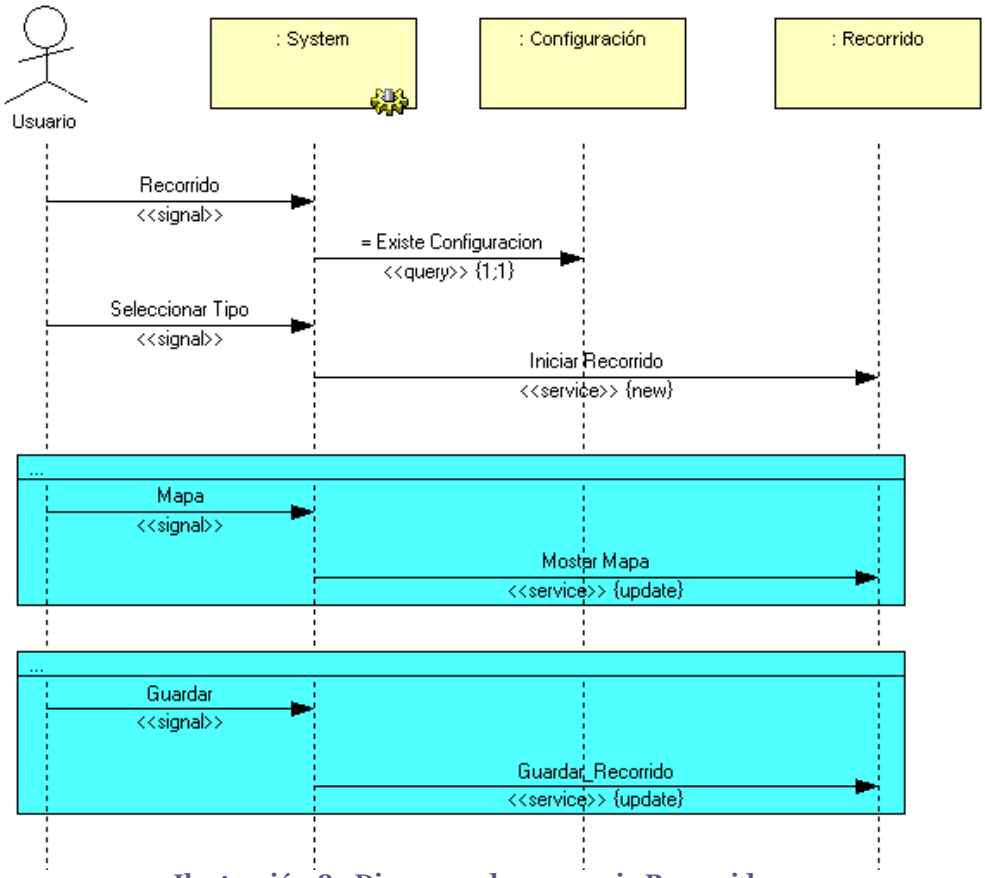

Ilustración 8 - Diagrama de secuencia Recorrido

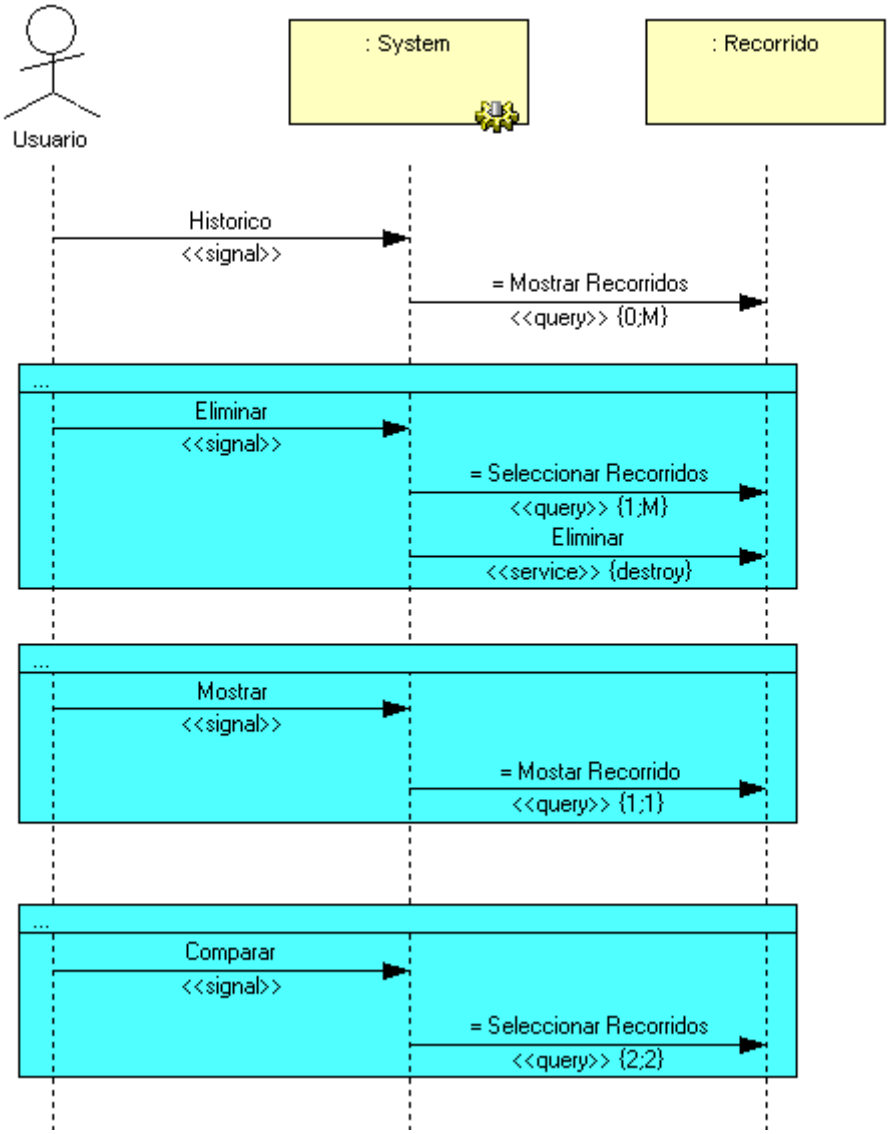

Ilustración 9 - Diagrama de secuencia Histórico

4.3 Arquitectura del sistema

La arquitectura del sistema que se va a utilizar será la de 3 capas. Con esta arquitectura el software se divide en 3 niveles diferentes: capa de presentación, capa de negocio, y la capa de datos. Cada nivel se desarrolla y se mantiene como una capa independiente.

- Capa de presentación: es el nivel superior de la aplicación. La capa de presentación proporciona la interfaz de usuario de la aplicación. La forman todos los componentes que comunicación y captan la información del usuario. Se comunica con la capa de negocio.
- Capa de negocio: Es la capa que contiene los procesos a realizar con la información recibida desde la capa de presentación, trata las peticiones que el usuario ha realizado, y con la responsabilidad de que se envíen las respuestas adecuadas a la capa de presentación.

Es una una capa intermedia, a medio camino entre la capa de presentación y la capa de datos, se relaciona con ambas, además de que también procesa la información que proviene de la capa de datos.

- La capa de datos: Es la última capa, donde se almacenan los datos. La capa de negocio se encarga, mediante el gestor de base de datos, del acceso de los datos para su consulta, actualización, almacenaje o eliminación. Se comunica con la capa de negocio facilitando los datos o recibiéndolos.

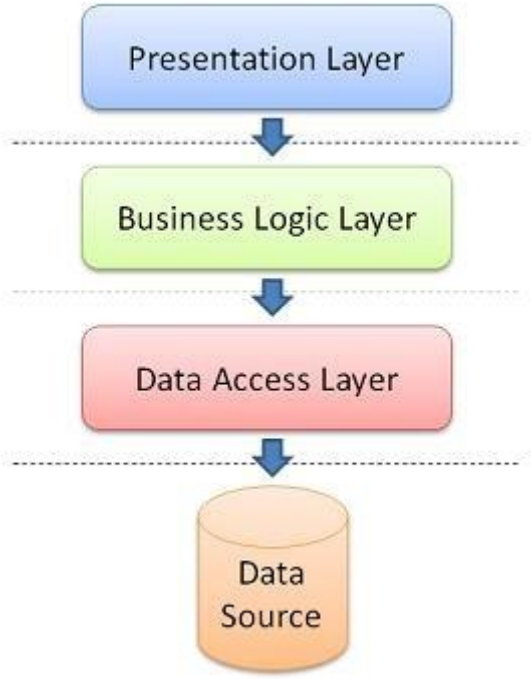

Ilustración 10 - Arquitectura del sistema

4.4 Interfaz gráfica

A continuación mostraremos el diseño de las pantallas que componen la aplicación y una breve explicación de las mismas.

La pantalla principal de la aplicación tendrá el siguiente diseño:

Ilustración 11 - Pantalla Principal (diseño)

Compuesta de 4 botones se podrá acceder a las diferentes opciones de la aplicación.

Al seleccionar "Configuración" se mostrará la siguiente pantalla que nos permitirá introducir los parámetros de configuración de la aplicación:

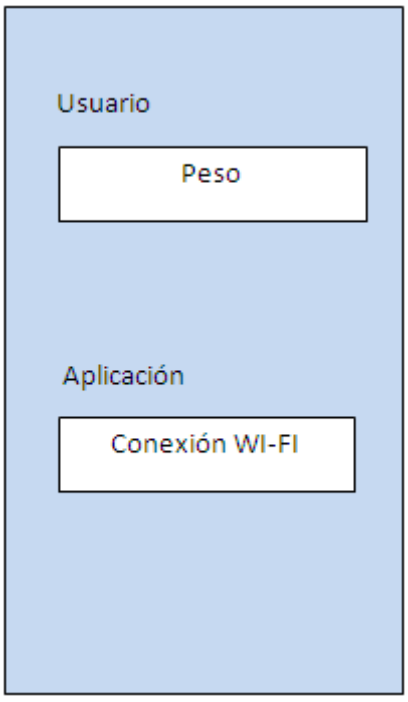

Ilustración 12 - Pantalla de Configuración (diseño)

Al seleccionar "Recorrido" se mostrará la siguiente pantalla:

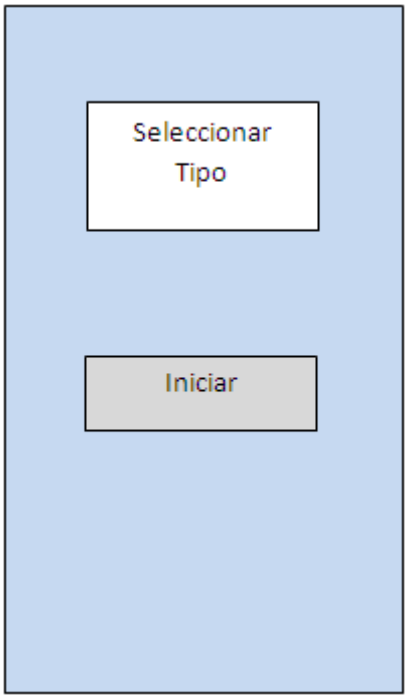

Ilustración 13 - Selección Recorrido (diseño)

Donde podremos seleccionar el recorrido que se va a realizar. Una vez iniciado se mostrará la siguiente pantalla formada por dos pestañas:

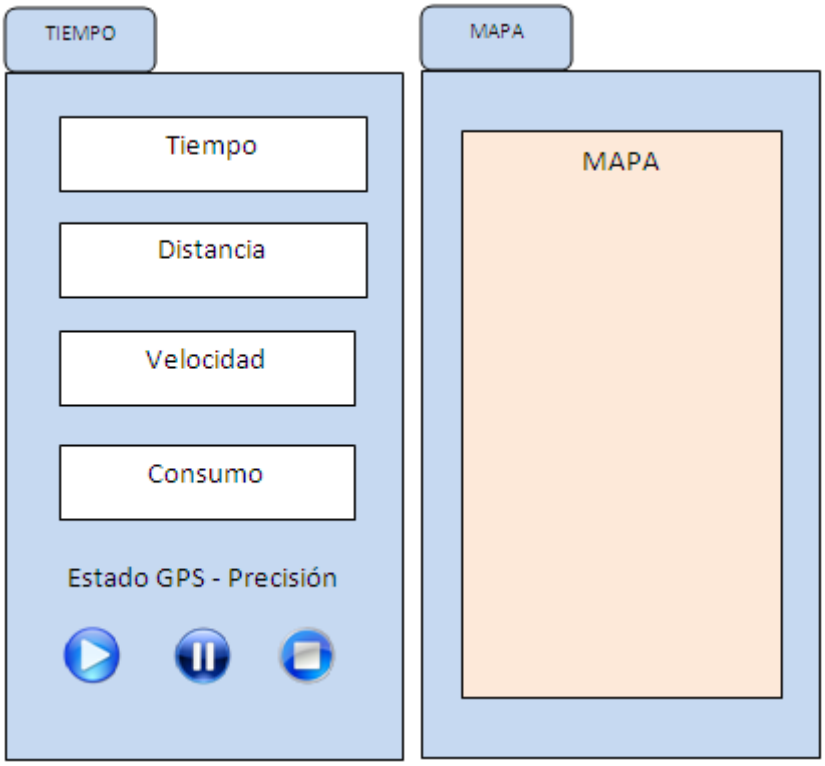

Ilustración 14 - Pantalla Recorrido (diseño)

Por defecto el mapa estará desactivado, se mostrará con una opción del menú.

Si desde el menú principal de la aplicación se selecciona “Histórico”, se mostrará la siguiente pantalla:

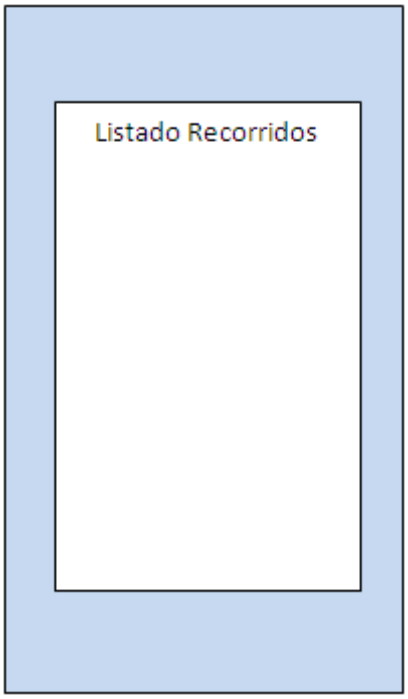

Ilustración 15 - Histórico (diseño)

Mediante una opción del menú, seleccionando un recorrido se mostrará la información del mismo:

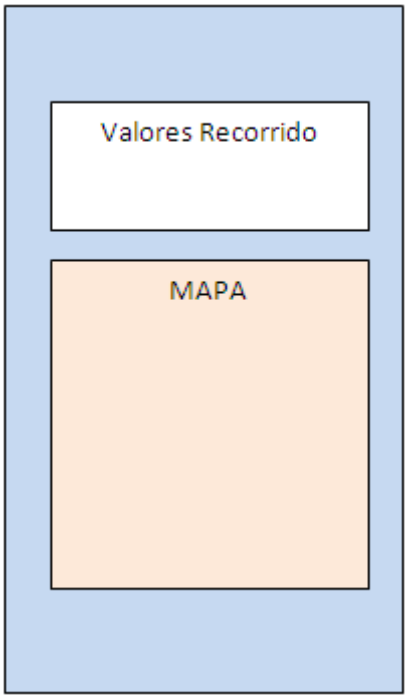

Ilustración 16 - Mostrar Recorrido (diseño)

En cambio, si desde "Histórico", se seleccionasen dos recorridos, mediante una opción de menú se podrá obtener una pantalla con una comparativa de ambos:

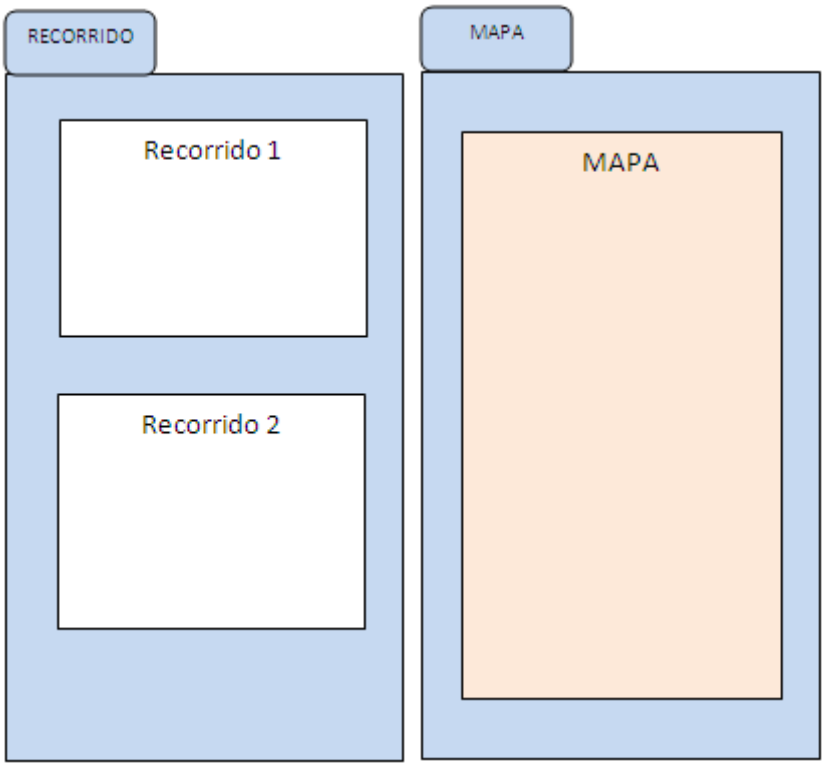

Ilustración 17 - Comparación Recorridos (diseño)

4.5 Diseño base de datos

La base de datos únicamente estará formada por una tabla, Recorrido, con la siguiente estructura:

Campo	Tipo	Observaciones
Id	Integer	Clave Primaria: Contendrá el identificador del registro
Tipo	Integer	Tipo de recorrido que se va a realizar
Fecha Inicio	DateTime	Fecha en formato dd/mm/yyyy HH:mm:ss en que se inicia el recorrido
Fecha Fin	DateTime	Fecha en formato dd/mm/yyyy HH:mm:ss en que se finaliza el recorrido
Tiempo	Long	Tiempo empleado para realizar el recorrido
Puntos Gps	String	String que contendrá mediante separación los puntos GPS obtenidos
Distancia	Long	Distancia que se ha recorrido
Consumo	Long	Consumo de calorías empleado en el recorrido

Tabla 2 - Estructura Tabla de recorridos

Aunque de acuerdo al diagrama de clases existen dos clases, se ha optado por este diseño para la base de datos con única tabla ya que para almacenar la configuración de la aplicación, al tratarse de un único registro, se hará uso de las estructuras XML que nos proporciona Android.

5. Implementación

5.1 Entorno de desarrollo

5.1.1 Entorno hardware

Para el desarrollo de la aplicación se han utilizado indistintamente dos ordenadores personales. A continuación se detallan las principales características técnicas:

- PC Portátil: procesador Intel Centrino 1,6GHz con 1,256 GB de memoria RAM y Windows XP SP3 de sistema operativo.
- PC Sobremesa: procesador Intel Pentium 4 a 2,6GHz con 3,768 GB de memoria RAM y Windows XP SP3 de sistema operativo.

Para la depuración y realización de pruebas se ha utilizado un dispositivo móvil:

- Huawei Ascend G300: procesador Qualcomm 1GHZ Cortex-A5 con 512MB de memoria RAM, pantalla de 4" y Android 4.0.3 (Ice Cream Sandwich).

5.1.2 Configuración entorno software

Como entorno de desarrollo se ha utilizado Eclipse, a continuación se detallará el proceso para su configuración para ser utilizado en la programación en Android.

Paso 1. Descarga e instalación de Java.

El equipo debe tener instalada una versión de Java Development Kit, aconsejable la última versión disponible que se puede descargar desde la web de Oracle. Se ha utilizado la versión 7 update 11.

Paso 2. Descarga e instalación de Eclipse.

Se ha descargado la versión Eclipse 4.2.1 (Eclipse Juno SR1), la versión de 32 bits de Eclipse IDE for Java Developers de la web de eclipse:

http://www.eclipse.org/downloads/

La instalación es tan sencilla como descomprimir el zip descargado en la ubicación deseada.

Paso 3. Descargar e instalación del SDK de Android.

El SDK de la plataforma Android se puede descargar desde la web de desarrollo de Android (http://developer.android.com), seleccionando "Use an existing IDE" → "Download the SDK Tools for Windows". Se ha utilizado la versión r21, que funciona perfectamente con Eclipse 4.2.1.

Paso 4. Descargar el plugin Android para Eclipse.

Google pone a disposición de los desarrolladores un plugin para Eclipse llamado Android Development Tools (ADT) que facilita el desarrollo de aplicaciones para la plataforma. Se puede descargar desde las opciones de

actualización de Eclipse "Help / Install new software..." indicando la siguiente URL de descarga: https://dl-ssl.google.com/android/eclipse/

Se deben seleccionar los dos paquetes disponibles "Developer Tools" y "NDK Plugins".

Paso 5. Configurar el plugin ADT.
Una vez instalado el plugin, se deberá configurar indicando la ruta donde ha sido instalado el SDK de Android. Desde la ventana de configuración de Eclipse (Window / Preferences...), en la sección de Android se indica la ruta.

Paso 6. Instalar las Platform Tools y los Platforms necesarios.
Aparte del SDK de Android instalado previamente que contiene las herramientas básicas para desarrollar en Android, también se deben descargar las Plataform Tools que contiene herramientas específicas de la última versión de la plataforma, y una o varias plataformas (SDK Platforms) de Android, que no son más que las librerías necesarias para desarrollar sobre cada una de las versiones concretas de Android.

Se aconseja instalar (en Eclipse desde "Window/ Android SDK Manager") al menos dos versiones, la mínima versión sobre la que se desea soporte para nuestra aplicación y la máxima versión disponible. En nuestro caso se han selecciona la versiones "Android 4.2 (API 17)" y la versión "Android 2.2 (API 8)" como mínima versión. También se ha seleccionado "Android Support Library", que es una librería que nos permitirá utilizar en versiones antiguas de Android características introducidas por versiones más recientes

Paso 7. Configurar Google Maps.
Para hacer uso del API de Google Maps v2 primeramente deberemos descarga el *Google Play services* desde SDK Manager de Eclipse.

El siguiente paso será obtener una *API Key* para poder utilizar el servicio de mapas de Google en nuestra aplicación. Dicha key la obtendremos en la consola APIs de Google: https://code.google.com/apis/console indicando datos identificativos de nuestra aplicación como el nombre del paquete o la huella digital SH1 con la que se firmará nuestra aplicación.

La API Key obtenida se añadirá al fichero de configuración *AndroidManifest.xml* de nuestra aplicación aparte de los permisos necesarios. Finalmente nuestro proyecto tendrá que tener una referencia a la librería previamente descargada en el SDK `google-play-services_lib`.

Paso 8. Configurar un AVD.
Parea realizar pruebas y depurar la aplicación desarrollada no es necesario disponer de un dispositivo físico. Es posible configurar un emulador o dispositivo virtual (Android Virtual Device o AVD) que nos proporciona el SDK

de Android. Podremos encontrarlo en Eclipse en “Window → AVD Manager”. Es posible indicar la versión de Android, resolución de pantalla, memoria disponible además de otras características hardwares como cámara, gps por ejemplo.

A pesar de la disponibilidad de un emulador de Android, debido a la limitación hardware disponible y al consumo de recursos (muchas veces no disponibles) que genera la ejecución del dispositivo virtual de Android se ha optado para la realización de las pruebas y la depuración del código en un dispositivo físico conectado al PC. Para ello es necesario activar la depuración USB en el móvil e instalar los drivers ADB Interface en Windows.

De este modo, al ejecutar la aplicación desde Eclipse, a parte de los AVD configurados, es posible seleccionar el dispositivo móvil conectado al ordenador.

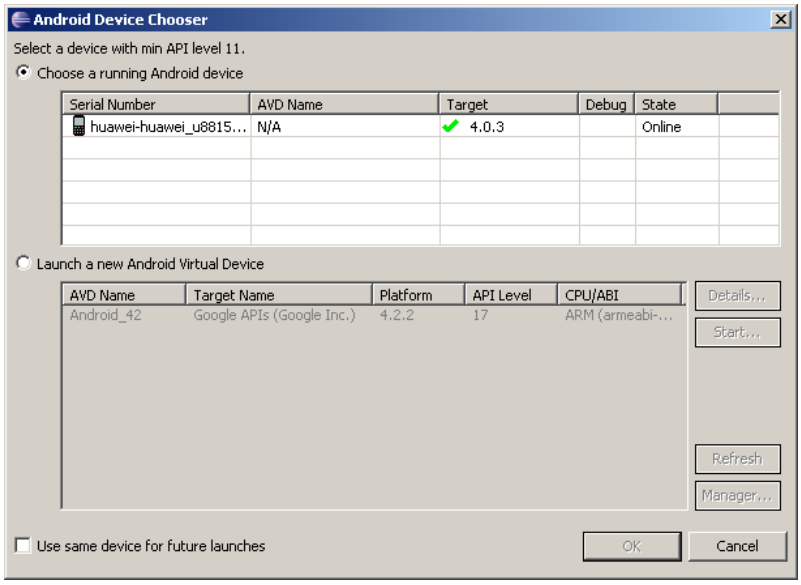

Ilustración 18 - Ejecución depurador Android

5.2 Estructura de un proyecto Android

En todo proyecto Android en eclipse nos podremos encontrar las siguientes carpetas o ficheros:

Carpeta /src/

Esta carpeta contendrá todo el código fuente Java de nuestra aplicación, código de la interfaz gráfica, así como las clases que vayamos creando.

Carpeta /res/

Contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, vídeos, cadenas de texto, etc. Los diferentes tipos de recursos se distribuyen entre las siguientes subcarpetas:

- /res/drawable/. Contienen las imágenes de la aplicación y otros elementos gráficos. Se dividen en subcarpetas /drawable-ldpi, /drawable-mdpi y /drawable-hdpi para definir los recursos dependiendo de la resolución de la pantalla del dispositivo.
- /res/layout/. Contienen los ficheros XML que definen las pantallas de la interfaz gráfica. Se puede dividir en /layout y /layout-land para definir distintos layouts dependiendo de la orientación del dispositivo (horizontal o vertical).
- /res/anim/. Contiene los ficheros XML que definen las animaciones utilizadas por la aplicación.
- /res/menu/. Contiene los ficheros XML que definen los menús de la aplicación.
- /res/values/. Contiene otros ficheros XML de recursos de la aplicación como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), etc.
- /res/xml/. Contiene otros ficheros de datos XML utilizados por la aplicación.
- /res/raw/. Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.

Carpeta /gen/

Contiene ficheros que generan automáticamente al compilar el proyecto. El más importante es el fichero R.java que contiene la clase R. Esta clase nos permite acceder a los recursos ubicados en la carpeta /res mediante el uso de identificadores.

Carpeta /assets/

Es un directorio que contiene recursos auxiliares utilizados en la aplicación y que se incluyen en el propio paquete de la aplicación. La diferencia entre los recursos de la carpeta /res y estos radica que en para los primeros se generará un ID en la clase R y se pueden acceder a ellos referenciando dicho ID. En cambio para acceder a estos recursos se haría con la ruta de acceso a cualquier fichero del sistema. Normalmente se incluyen aquellos recursos que no se tiene intención de modificar.

Carpeta /bin/

Contiene los ficheros compilados de la aplicación. Destacable el fichero apk, que es el ejecutable instalable de la aplicación.

Carpeta /libs/

Contendrá las librerías auxiliares, normalmente en formato “.jar” que nuestra aplicación debe hacer uso.

Fichero AndroidManifest.xml

Es el archivo XML de configuración de nuestra aplicación. En él se define los aspectos principales de nuestra aplicación (nombre, versión, icono, etc.), las activities o servicios que ejecutará o los permisos necesarios para su ejecución, por ejemplo, uso de GPS, estado de la conexión Wi-fi, etc.

5.3 Programación de la aplicación

A continuación se comentarán las clases, funciones y ficheros que conforman la aplicación, así como la configuración de la misma. Se destacará las partes del código más relevantes destacando sus particularidades.

5.3.1 AndroidManifest.xml

La configuración de la aplicación, así como los permisos necesarios se establecen en el fichero *AndroidManifest.xml*.

La mínima versión en la que podrá ser ejecutado el programa será un Android 3.0 (API 11):

```
<uses-sdk android:minSdkVersion="11"
android:targetSdkVersion="17" />
```

Los permisos necesarios son básicamente la escritura en la tarjeta de memoria, el acceso a internet, el posicionamiento gps, al estado de la conexión wifi y los servicios de google maps.

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVIC
ES" />
```

También se ha establecido que las activities tengan siempre una posición vertical, no siendo posible ponerlas en modo apaisado,

```
android:screenOrientation="portrait"
```

5.3.2 Clase FunctionsLog.java

La siguiente clase contiene funciones que son utilizadas de forma general en toda la aplicación.

La función *Comprobar_SD()* comprueba si la memoria SD de la aplicación está montada para poder escribir en ella una vez se ha establecido el permiso en el fichero *AndroidManifest.xml*.

La función *Save(String cadena)* se utiliza crear un fichero de texto en la tarjeta SD (en la ruta */SportTracking/Log/*) donde se guardará trazas y posibles errores de la aplicación para su posterior depuración por parte del desarrollador. Crea un fichero por día de la semana.

Mediante la función *Comprobar_Internet()* se comprueba si el dispositivo tiene conexión Wifi disponible si se ha determinado en los parámetros de configuración la utilización únicamente de Wifi.

Finalmente la funciones *DeleteGpsFile()* y *SaveGps(String cadena)* eliminan y crean respectivamente un fichero de texto en la ruta */SportTracking/Log/* donde se almacenan temporalmente los puntos que son recogidos por el posicionamiento gps.

5.3.2 Clase STSQLiteHelper.java

La siguiente clase que extiende `SQLiteOpenHelper` contiene las funciones estándar necesarias para la lectura, actualización o eliminación de registros en la base de datos SQLite.

Destacamos la función *copyDataBase()*. Esta función crea en el dispositivo la base de datos que previamente ha sido añadida en la carpeta *assets* del proyecto.

```
public void copyDataBase() throws IOException{

    //Llamando a este método se crea la base de datos vacía en la ruta
por defecto del sistema
    //de nuestra aplicación por lo que podremos sobreescribirla con
nuestra base de datos.
    this.getReadableDatabase();
    //Abrimos el fichero de base de datos como entrada
    InputStream myInput = contexto.getAssets().open(db_name);

    //Ruta a la base de datos vacía recién creada
    String outFileName = db_path + db_name;

    //Abrimos la base de datos vacía como salida
    OutputStream myOutput = new FileOutputStream(outFileName);

    //Transferimos los bytes desde el fichero de entrada al de salida
    byte[] buffer = new byte[1024];
    int length;
    while ((length = myInput.read(buffer))>0){
    myOutput.write(buffer, 0, length);
            }

    //Liberamos los streams
    myOutput.flush();
    myOutput.close();
    myInput.close(); }
```

5.3.3 Clase SportTracking_Main.java

La clase *SportTracking_Main* corresponde a la pantalla principal de la aplicación, siendo la activity asociada *sporttracking_main_activity.xml*.

Al ser la primera pantalla al iniciar la aplicación, después de haber sido instalada, comprueba si la base de datos existe, de lo contrario la crea.

```
STSQLiteHelper stdb = new STSQLiteHelper(this,"SportTracking_BD.s3db",
null, 1);
     if(!stdb.existeBD()){
         stdb.copyDataBase();
         FicheroLog.Save("Creamos BD: SportTracking_BD.s3db");
     }
```

5.3.4 Clase SportTracking_Configuracion.java

La clase *SportTracking_Configuracion* corresponde a la pantalla de configuración de la aplicación. La activity asociada a la clase es la *sporttracking_configuracion_activity.xml*.

Para almacenar los parámetros de configuración de la aplicación se podría haber escogido por guardarlos en la base de datos, pero al tratarse de un único registro se optó por utilizar la clase *SharedPrefences* que nos proporciona Android. Mediante esta clase los parámetros se almacenarán en un fichero xml en una ruta interna del dispositivo, siendo accesibles desde cualquier parte de la aplicación utilizando dicha clase.

Una vez comprobados que los parámetros introducidos son correctos se almacenan de la siguiente forma:

```
SharedPreferences PrefConfig =

getSharedPreferences("Configuracion",Context.MODE_PRIVATE);
SharedPreferences.Editor PrefEditor = PrefConfig.edit();
PrefEditor.putInt("vpeso", vpeso);
PrefEditor.putBoolean("vconexion", vconexion);
PrefEditor.putBoolean("vlog", vlog);
PrefEditor.commit();
```

Para recuperar unos parámetros de configuración guardados sería de la siguiente forma:

```
SharedPreferences PrefConfig =

getSharedPreferences("Configuracion",Context.MODE_PRIVATE);
int vpeso = PrefConfig.getInt("vpeso", 0);
boolean vconexion = PrefConfig.getBoolean("vconexion", false);
boolean vlog = PrefConfig.getBoolean("vlog", false);
```

5.3.5 Clase SportTracking_Recorrido.java

Una vez iniciado el recorrido en la pantalla principal, se accede a la activity *sporttracking_recorrido_activity.xml* que está asociada a la clase *SportTracking_Recorrido*.

En esta clase básica primeramente se comprueba si el gps está activo,

```
locManager =
(LocationManager)getSystemService(Context.LOCATION_SERVICE);
      if (!locManager.isProviderEnabled(LocationManager.GPS_PROVIDER))
{
            Mensaje_Activar_Gps();
      }
```

Si no fuese el caso, nos da la posibilidad de activarlo. Android no permite activarlo por código, así que se debe mostrar la pantalla de configuración de "Servicios de ubicación" del sistema,

```
Intent gps_intent =
new Intent(android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS);

startActivity(gps_intent);
```

Esta clase nos permite seleccionar el tipo de recorrido que se va a realizar mediante una lista desplegable que ha sido personalizada. Se ha definido gráficamente en el fichero *list_item_image.xml*. La lista se rellena con un adaptador definido en *ItemListImageAdapte.java* donde cada elemento es una clase *ItemListImage.java*. La clase básicamente es un nombre, subnombre y una imagen, la asignación y obtención de los mismos campos.

5.3.6 Clase SportTracking_TimeTab.java

SportTracking_TimeTab corresponde a la clase principal de la aplicación. Tiene asociada la activity *sporttracking_timetab_activity.xml*.

Gráficamente la activity está formada por dos pestañas. Una muestra el tiempo, la distancia, el consumo y la velocidad. La otra, por defecto está vacía y se añade por código un mapa de google maps,

```
private void Add_Mapa(){
    try{
    mMapFragment = SupportMapFragment.newInstance();
    mMapFragment.setRetainInstance(true);//Guarda instancia de mapa
    android.support.v4.app.FragmentTransaction fragmentTransaction =
getSupportFragmentManager().beginTransaction();
    fragmentTransaction.add(R.id.tabmap, mMapFragment);
    fragmentTransaction.commitAllowingStateLoss();
    MapAdd = true;

    final Handler handler = new Handler();
    handler.postDelayed(new Runnable() {
        @Override
        public void run() {
            Inicializar_Mapa();
        }
    }, 200);
    } catch(Exception
e){FicheroLog.Save("SportTracking_TimeTab.Add_Mapa:" + e.toString());}
}
```

Una vez se añade el mapa, se retrasa su inicialización 200 milisegundos ya que antes de trabajar con el mapa añadiendo puntos, trazos, etc. se debe esperar a que el fragment esté cargado de lo contrario se produce un error.

El mapa se inicializa con la última posición conocida o si no existiese se centra en España,

```
mapa = mMapFragment.getMap();

if (mapa != null) {
  if (LocationMap != null) {

MostrarPunto(LocationMap.getLatitude(),LocationMap.getLongitude());
  }else{
  CampMap = CameraUpdateFactory.newLatLngZoom(new LatLng(40.41, -
3.69),5);
  mapa.moveCamera(CampMap);}
}
```

El mapa puede eliminarse de la pestaña para evitar el consumo de datos,

```
FragmentManager fmanager = getSupportFragmentManager();
Fragment fragment = fmanager.findFragmentById(R.id.tabmap);
fmanager.beginTransaction().remove(fragment).commitAllowingStateLoss()
;
```

Gráficamente se comentarán los botones con imágenes. Para dotarlos de efectos al pulsarlos, o deshabilitarlos hay que establecer un selector. Por ejemplo, el botón play se definen en la activity así:

```
<ImageButton
    android:id="@+id/BtnIniciar"
    android:layout_width="70dp"
    android:layout_height="70dp"
    android:background="@drawable/start_selector"
    android:contentDescription="@string/btniniciar"
    android:layout_marginLeft="25dp" />
```

El selector *start_selector* definido en la carpeta /drawable contiene las imágenes dependiendo de si se presiona la imagen, está activo o deshabilitado:

```
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:state_pressed="true"
android:drawable="@drawable/ic_start_press"/>
    <item android:state_enabled= "false"
android:drawable="@drawable/ic_start_dis"/>
    <item android:state_enabled= "true"
android:drawable="@drawable/ic_start"/>
</selector>
```

Dos componentes principalmente forman la clase, el cronometro para medir tiempos y el gps para obtener la geolocalización.

El cronometro se inicia asociándolo al tiempo del sistema,

```
Cronometro.setBase(SystemClock.elapsedRealtime() + timeWhenStopped);
```

```
Cronometro.start();
isChronometerRunnig = true;
```

En caso de detenerlo o pausarlo se debe guardar el tiempo que marcaba cuando se detuvo,

```
timeWhenStopped = Cronometro.getBase()-SystemClock.elapsedRealtime();
Cronometro.stop();
isChronometerRunnig = false;
```

Al volverlo a iniciar se tiene en cuenta este tiempo, `timeWhenStopped`.

Con cada cambio de tiempo del cronometro se formatea su visualización en pantalla,

```
//Formato HH:MM:SS Cronómetro
Cronometro.setOnChronometerTickListener(new
OnChronometerTickListener() {
public void onChronometerTick(Chronometer cArg) {
    long t = SystemClock.elapsedRealtime() - cArg.getBase();
    timeCronometro = t;
    cArg.setText(FormatoHora.format(t));
    if ((t/1000)%60 == 0 && t != 0)
        Calcular_Consumo(t);
}
});
```

Y cada 60 segundos se calcula el consumo como tres variables: tiempo, el peso y una constante que depende de la actividad física que se realiza,

```
private void Calcular_Consumo(long tiempo){
    //Consumo = minutos * peso * constante
    consumo =  (tiempo/1000)/60 * vpeso * kenergia;
    TxtConsumo.setText(FormatoNumero.format(consumo) + " kcal");
}
```

El gps se inicia al crear la activity,

```
private void IniciarGPS()
{
//Obtenemos una referencia al LocationManager
locManager =
(LocationManager)getSystemService(Context.LOCATION_SERVICE);
//Comprobamos si el Gps está activo
if (!locManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
    gps_status = "GPS Off";
    LblEstado.setBackgroundColor(Color.RED);
}else{
    gps_status = "GPS On";
    LblEstado.setBackgroundColor(Color.GREEN);
}
LblEstado.setText(gps_status);

//Si el gps se inicia de nuevo después de volver crear la activity
if (LocationExist){
    //Obtenemos la última posición conocida
    LocationMap =
locManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
```

```
    //Mostramos la última posición conocida

MostrarPunto(LocationMap.getLatitude(),LocationMap.getLongitude());
}

//Nos registramos para recibir actualizaciones de la posición
locListener = new LocationListener() {
    public void onLocationChanged(Location location) {
            mostrarPosicion(location);
    }
    public void onProviderDisabled(String provider){
        gps_status = "GPS Off";
        LblEstado.setText(gps_status);
        LblEstado.setBackgroundColor(Color.RED);
    }
    public void onProviderEnabled(String provider){
        gps_status = "GPS On";
        LblEstado.setText(gps_status);
        LblEstado.setBackgroundColor(Color.GREEN);
    }
    public void onStatusChanged(String provider, int status, Bundle
extras){
    }
};

locManager.requestLocationUpdates(
        LocationManager.GPS_PROVIDER, 0, 0, locListener);
}
```

Una vez el gps está activo y tiene conexión, con cada cambio de posición que nos proporciona se llama a la función `mostrarPosicion`,

```
private void mostrarPosicion(Location loc) {
    float distancia=0;
    if(loc != null)
    {
    float precision = loc.getAccuracy();
    //El punto que se muestra en el mapa se actualiza cuando la
distancia es mayor que la precisión
    if (LocationMap != null){
        distancia = loc.distanceTo(LocationMap);
        if (distancia > precision){
            LocationMap = loc;
            MostrarPunto(loc.getLatitude(),loc.getLongitude());

            //añadimos nuevo punto a la Lista de ptos
            LatLng pto = new LatLng(loc.getLatitude(),
loc.getLongitude());
            lstptos.add(pto);
            //Guardamos puntos en fichero txt,
            //Guardamos lat##lng
            String strGps =  loc.getLatitude() + "##" +
loc.getLongitude() + ";";
            FicheroLog.SaveGps(strGps);

            MostrarTrazo();
            //Aumentamos distancia recorrida
            if(isChronometerRunnig){
                distanciaTotal = distanciaTotal + distancia;
                if(distanciaTotal<1000)
```

```
TxtDistancia.setText(FormatoNumero.format(distanciaTotal) + " m");
                else

TxtDistancia.setText(FormatoNumero.format(distanciaTotal/1000) + "
Km");
            }
        }
    }else{
        LocationMap = loc;
        LatLng pto = new LatLng(loc.getLatitude(),
loc.getLongitude());
        lstptos.add(pto);
        MostrarTrazo();
    }

    //Mostramos Precisión
    LblEstado.setText(gps_status + " Precisión:" + loc.getAccuracy() +
" m");
    if(isChronometerRunnig){
        //Mostramos velocidad dependiendo del tipo
        if (oTipo==0 || oTipo ==2){
            //corriendo/caminando: metros/min

TxtVelocidad.setText(FormatoEntero.format(loc.getSpeed()*60) + " m/min
(" + FormatoNumero.format(loc.getSpeed()*3600/1000) + " Km/h)");
        }
        else{
            //bicicleta: km/h

TxtVelocidad.setText(FormatoNumero.format(loc.getSpeed()*3600/1000) +
" Km/h");
            }
    }}
    else
    {
        TxtVelocidad.setText(vel_ini);
    }
}
```

Si la distancia de la posición obtenida comparada con la última posición guardada es superior a la precisión del gps, se guarda la posición y la distancia recorrida. De lo contrario se espera a que la distancia sea superior a la precisión que nos proporciona el gps.

Los puntos gps se almacenan en un fichero de texto utilizando la función definida en la clase *FunctionsLog.java*

Además de almacenar los puntos, se muestran en un mapa de google maps mediante un icono la última posición obtenida,

```
private void MostrarPunto(double lat, double lng)
{
try {
if (MapAdd){
    if (mapa!= null){
        LatLng punto;
        punto = new LatLng(lat,lng);
```

```
        if (marca!=null){
            //Eliminamos marca previa
            marca.remove();
        }
        marca = mapa.addMarker(new MarkerOptions()
            .icon(BitmapDescriptorFactory.fromResource(intimage))
            .position(punto));

        //centramos mapa en punto
        if(CenterZoom){
           //Zoom no se modifica. Usuario puede que lo haya modificado
            CampMap = CameraUpdateFactory.newLatLng(punto);
        }
        else{
            //1a vez zoom 16
            CampMap = CameraUpdateFactory.newLatLngZoom(punto,16);
            CenterZoom = true;
        }
        mapa.moveCamera(CampMap);
        }
    else
        Inicializar_Mapa();
}
}catch (Exception e){
FicheroLog.Save("SportTracking_TimeTab.MostrarPunto:" + e.toString());
}
}
```

Del mismo modo se muestra en el mapa el recorrido que se va realizando mediante una polylínea,

```
private void MostrarTrazo(){
try {
    if (MapAdd){
        //mapa añadido
        if(mapa!=null){
            //existe mapa
            if (trazo==null){
                //creamos Polylinea con 1er pto
                PolylineOptions lineOptions = new
PolylineOptions().width(5).color(Color.BLUE);
                trazo = mapa.addPolyline(lineOptions);
            }else{
                //añadimos nuevo punto
                trazo.setPoints(lstptos);
            }

        }
    }
}catch (Exception e){
FicheroLog.Save("SportTracking_TimeTab.MostrarTrazo:" + e.toString());
}
}
```

Android monitoriza las operaciones que se realizan en el hilo principal, y detecta las que superen los 5 segundos mostrando el típico mensaje de “La aplicación no responde”, dando la posibilidad al usuario de detener la aplicación o esperar a que finalice el proceso. Para evitarlo se ha creado un hilo que se encargará de

guardar el recorrido mostrando una barra de progreso que evita la sensación de bloqueo si el proceso es de larga duración.

```
private class TaskGuardar extends AsyncTask<Void, Integer, Boolean> {

@Override
protected Boolean doInBackground(Void... params) {
    //Proceso del hilo
    Guardar_Recorrido();
    //Apagamos Gps
    Apagar_Gps();
    return true;
}

@Override
protected void onProgressUpdate(Integer... values) {
    int value = values[0].intValue();
    switch(value)
    {
        case 0:
        {
            Toast.makeText(SportTracking_TimeTab.this, "Recorrido
almacenado correctamente", Toast.LENGTH_LONG).show();
            break;
        }

        case 1:
        {
            Toast.makeText(SportTracking_TimeTab.this, "Error
almacenando recorrido", Toast.LENGTH_LONG).show();
            break;
        }
    }
}

@Override
protected void onPreExecute() {
    //Antes de Iniciar, muestra barra de progreso
    PrgDiag = new ProgressDialog(SportTracking_TimeTab.this);
    PrgDiag.setTitle("Guardando");
    PrgDiag.setMessage("Espere por favor...");
    PrgDiag.setProgressStyle(ProgressDialog.STYLE_SPINNER);
    PrgDiag.setCancelable(false);
    PrgDiag.setIndeterminate(true);
    PrgDiag.show();
}

@Override
protected void onPostExecute(Boolean result) {
    //Al finalizar, oculta barra de progreso
    PrgDiag.dismiss();
    //Finalizamos pantalla
    finish();
}

private void Guardar_Recorrido(){

    String sql = "";
    String ptos_gps = "";
    String linea = "";
```

```
        try{
        FicheroLog.Save("Inicio Guardar");
        //Abrimos la base de datos 'SportTracking_BD' en modo
escritura
        STSQLiteHelper stdb = new
STSQLiteHelper(SportTracking_TimeTab.this, "SportTracking_BD.s3db",
null, 1);
        SQLiteDatabase db = stdb.getWritableDatabase();
        //Si hemos abierto correctamente la base de datos
        if(db != null)
        {
            if(oFechaFin==null){
                oFechaFin = new Date();
            }

            //Leemos puntos almacenados en gps.txt
            File ruta_sd = Environment.getExternalStorageDirectory();
            File directorioLOG = new
File(ruta_sd.getAbsolutePath()+"/SportTracking/Log/");
            File archivo = null;

            String filename = "gps.txt";

            archivo = new File(directorioLOG, filename);

            if (archivo.exists()) {
                FileReader FileR = new FileReader(archivo);
                BufferedReader BufferR = new BufferedReader(FileR);

                if ((linea = BufferR.readLine())!=null)
                    ptos_gps = linea;
                else
                    ptos_gps = "";
                BufferR.close();
            }

            //Insertamos los datos en la tabla Usuarios
            sql = "Insert Into STRecorrido(RECTipo,RECFechaInicio,"
                    + "RECFechaFin, RECTime,RECGps, "
                    + "RECDistancia, RECConsumo)"
                          + "Values (" + oTipo + ",'" +
      FormatoFecha.format(oFechaIni) + "','"
                    + FormatoFecha.format(oFechaFin) + "'," +
timeCronometro + ",'" + ptos_gps + "',"
                    + distanciaTotal + "," + consumo +")";

            db.execSQL(sql);
            //Cerramos la base de datos
            db.close();
            FicheroLog.Save("Fin Guardar");
            publishProgress(0);
        }
    }
    catch (Exception e) {
        publishProgress(1);
        FicheroLog.Save("SportTracking_Time.Guardar_Recorrido:" +
e.toString());
        FicheroLog.Save("Guardar_Recorrido - InsertSQL:" + sql);
        }

    }}
```

Una vez guardado el recorrido se apaga el gps ejecutando,

```
locManager.removeUpdates(locListener);
```

Finalmente, ya que esta activity probablemente estará abierta mucho tiempo, y puede que entre una llamada, que el usuario inicie otro programa como un reproductor de música, etc. o que el propio sistema Android la destruya en ese caso deberemos controlar el ciclo de vida de la activity.

Así el estado de la actividad lo guardaremos en,

```
@Override
protected void onSaveInstanceState(Bundle savedInstanceState) {
    FicheroLog.Save("onSaveInstanceState");
    super.onSaveInstanceState(savedInstanceState);
    savedInstanceState.putLong("TIME", Cronometro.getBase());
    savedInstanceState.putLong("TIME_STOP",timeWhenStopped);
    savedInstanceState.putBoolean("IS_RUNNING", isChronometerRunnig);
    savedInstanceState.putBoolean("MAPA_ADD", MapAdd);
    savedInstanceState.putDouble("CONSUMO", consumo);
    savedInstanceState.putFloat("DISTANCIA", distanciaTotal);
    if (LocationMap != null){
        savedInstanceState.putBoolean("LOCATION_EXIST", true);
    }else{
        savedInstanceState.putBoolean("LOCATION_EXIST", false);
    }
}
```

Al crear la actividad, en el método *onCreate* comprobamos si ya existía previamente recargando la variables guardadas o son iniciadas por primera vez,

```
//Controlar ciclo de vida
if (savedInstanceState!=null){
    //Estado guardado
    FicheroLog.Save("OnCreate");

    isChronometerRunnig = savedInstanceState.getBoolean("IS_RUNNING");
    if (isChronometerRunnig){
        Cronometro.setBase(savedInstanceState.getLong("TIME"));
        Cronometro.start();
        BtnIniciar.setEnabled(false);
        BtnParar.setEnabled(true);
        BtnPausar.setEnabled(true);
    }else{
        Cronometro.stop();
        timeWhenStopped = savedInstanceState.getLong("TIME_STOP");
        Cronometro.setBase(SystemClock.elapsedRealtime() +
timeWhenStopped);
        BtnIniciar.setEnabled(true);
        BtnParar.setEnabled(false);
        BtnPausar.setEnabled(false);
        //Formateamos cronometro
        long t = SystemClock.elapsedRealtime() - Cronometro.getBase();
        timeCronometro = t;
        Cronometro.setText(FormatoHora.format(t));
    }
    MapAdd = savedInstanceState.getBoolean("MAPA_ADD");
```

```
    consumo = savedInstanceState.getDouble("CONSUMO");
    TxtConsumo.setText(FormatoNumero.format(consumo) + " kcal");
    distanciaTotal = savedInstanceState.getFloat("DISTANCIA");
    if(distanciaTotal<1000)
        TxtDistancia.setText(FormatoNumero.format(distanciaTotal) + "
m");
    else
        TxtDistancia.setText(FormatoNumero.format(distanciaTotal/1000)
+ " Km");

    if(MapAdd){
        //El mapa estaba añadido
        Add_MapaExist();
    }
    //Obtenemos último punto del mapa
    LocationExist = savedInstanceState.getBoolean("LOCATION_EXIST");
}
else{
    //Creada primera vez
    //Eliminamos fichero gps
    FicheroLog.DeleteGpsFile();
    //Iniciamos Campos
    Cronometro.setText("00:00:00");
    Cronometro.start();
    BtnIniciar.setEnabled(false);
    BtnParar.setEnabled(true);
    BtnPausar.setEnabled(true);
    isChronometerRunnig = true;
}
```

5.3.7 Clase SportTracking_Historico.java

La clase *SportTracking_Historico* lista los recorridos almacenados en la base de datos. La activity asociada es *sporttracking_historico_activity.xml*.

La tabla con los registros se crea dinámicamente por código,

```
private void Leer_BD(){
try{

    STSQLiteHelper stdb = new STSQLiteHelper(this,
"SportTracking_BD.s3db", null, 1);
    SQLiteDatabase db = stdb.getReadableDatabase();
    String sql;
    sql = "Select RECId, RECTipo, strftime('%d-%m-%Y', RECFechaInicio)
As Fecha, RECTime, "
          + "RECDistancia "
          + "From  STRecorrido";

    //Abrimos cursor para recorrer registros
    Cursor cursor = db.rawQuery(sql, null);

   Integer count=0;
   while (cursor.moveToNext()) {
       int vId = cursor.getInt(0); //Identificador
       Integer vtipo = cursor.getInt(1); //Tipo
       String vfecha = cursor.getString(2);//fecha
       Double  vtiempo = cursor.getDouble(3);//tiempo
       Double vdistancia =  cursor.getDouble(4)/1000;
       String strimage;
```

```
        //Creamos fila de la Tabla
        TableRow tr = new TableRow(this);

       //Al hacer click sobre una fila
        tr.setClickable(true);
        tr.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
             // TODO Auto-generated method stub
             ColorDrawable Color_row = (ColorDrawable)
v.getBackground();
             if(Color_row.getColor() == Color.CYAN){
                 v.setBackgroundColor(Color.GRAY);//fila no
seleccciona
             }else if(Color_row.getColor() == Color.GRAY){
                 v.setBackgroundColor(Color.CYAN);//fila seleccionada
             }else if(Color_row.getColor() == Color.TRANSPARENT){
                 v.setBackgroundColor(Color.GREEN);//fila seleccionada
             }else{
                 v.setBackgroundColor(Color.TRANSPARENT);//fila no
seleccionada
             }
             comprobar_seleccion();
         }
        });

        if(count%2!=0) {
            tr.setBackgroundColor(Color.GRAY);
        }else{
            tr.setBackgroundColor(Color.TRANSPARENT);
        }
        tr.setId(vId + 1000);
        tr.setLayoutParams(new LayoutParams(
        LayoutParams.FILL_PARENT,
        LayoutParams.WRAP_CONTENT));

        //Creamos las columnas que añadimos a la tabla
        //TIPO
        switch (vtipo){
        case 0:
            strimage = "drawable/ic_running";
            break;
        case 1:
            strimage = "drawable/ic_bicycle";
            break;
        case 2:
            strimage = "drawable/ic_walking";
            break;
        default:
            strimage = "drawable/ic_running";
            break;
        }

        ImageView labelTIPO = new ImageView(this);
        int imageResource = this.getResources().getIdentifier(strimage,
null, this.getPackageName());

labelTIPO.setImageDrawable(this.getResources().getDrawable(imageResour
ce));
        tr.addView(labelTIPO);
```

```
        //FECHA
        TextView labelDATE = new TextView(this);
        labelDATE.setId(10+count);
        labelDATE.setText(vfecha);
        labelDATE.setGravity(Gravity.CENTER);
        labelDATE.setPadding(2, 0, 5, 0);
        labelDATE.setTextColor(Color.BLACK);
        tr.addView(labelDATE);
        //TIEMPO
        TextView labelTIME = new TextView(this);
        labelTIME.setId(10+count);
        labelTIME.setText(FormatoHora.format(vtiempo));
        labelTIME.setGravity(Gravity.CENTER);
        labelTIME.setTextColor(Color.BLACK);
        tr.addView(labelTIME);
        //DISTANCIA
        TextView labelDISTANCE = new TextView(this);
        labelDISTANCE.setGravity(Gravity.CENTER);
        labelDISTANCE.setId(10+count);
        labelDISTANCE.setText(FormatoNumero.format(vdistancia) + "
Km");
        labelDISTANCE.setTextColor(Color.BLACK);
        tr.addView(labelDISTANCE);


        //Añadimos la fila a la tabla
        tbl.addView(tr, new TableLayout.LayoutParams(
                            LayoutParams.FILL_PARENT,
                            LayoutParams.WRAP_CONTENT));
                    count++;
    }
}   catch (Exception e) {
    FicheroLog.Save("SportTracking_Historico.Leer_BD:" +
e.toString());}
}
```

Asociando a cada fila como identificador el id de la base de datos. Si se selecciona una fila de la tabla se modifica el color de la misma.

Para eliminar, mostrar o comparar recorridos se recorre la tabla comprobando los colores de las filas. Por ejemplo para realizar un comparativo de recorridos,

```
private void Comparar_Historico(){
      //Buscamos Id seleccionado
        int vId1 = -1;
        int vId2 = -1;
        int count = tbl.getChildCount();
        for (int i = 1; i < count; i++) {
            View row = tbl.getChildAt(i);
            ColorDrawable Color_row = (ColorDrawable)
row.getBackground();
            if (Color_row.getColor() == Color.GREEN ||
Color_row.getColor() == Color.CYAN){
                //El id de la fila contiene es 1000 + id de la BD
                if (vId1==-1)
                    vId1 = row.getId() - 1000;
                else {
                    vId2 = row.getId() - 1000;
                    break;}
            }
        }
```

```
        //Creamos el Intent
        Intent intent = new Intent(SportTracking_Historico.this,
SportTracking_Comparar.class);

        //Creamos la información a pasar entre actividades
        Bundle b = new Bundle();
        b.putInt("RECId1", vId1);
        b.putInt("RECId2", vId2);

        //Añadimos la información al intent
        intent.putExtras(b);

        //Iniciamos la nueva actividad
        startActivity(intent);
}
```

En cambio, para eliminar recorridos se pasa el *vId* seleccionado a la siguiente función,

```
private void Eliminar_Registros(String oWhere){
    try{
    STSQLiteHelper stdb = new STSQLiteHelper(this,
"SportTracking_BD.s3db", null, 1);
    SQLiteDatabase db = stdb.getReadableDatabase();
    String sql;
    sql = "Delete From STRecorrido " + oWhere ;
    db.execSQL(sql);
    db.close();
    }   catch (Exception e) {
        FicheroLog.Save("SportTracking_Historico.Eliminar_Registros:"
+ e.toString());}
}
```

5.3.8 Clase SportTracking_Comparar.java

Si se han seleccionado dos recorridos en la activity de Histórico se accederá a la clase *SportTracking_Comparar*.

Está clase está asociada a la activity *sporttracking_comparar_activity.xml*

Primeramente se comprueba si la activity es creada por primera vez o ya había sido creada y su estado había sido guardado previamente.

Si es la primera vez que se crea se comprueba que la conexión de internet está disponible, si no fuese el caso se eliminará de la pantalla el fragment correspondiente al mapa de google.

```
if (savedInstanceState==null){
if (FicheroLog.Comprobar_Internet()){
    vInternet = true;
    //Cargamos mapa GoogleMaps
    SupportMapFragment mMapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.mapC);
    mMapFragment.setRetainInstance(true);
    mapa = mMapFragment.getMap();
    mapa.setOnCameraChangeListener(new OnCameraChangeListener() {
        @Override
```

```
        public void onCameraChange(CameraPosition arg0) {
            try{
                //centramos mapa en trazo

mapa.moveCamera(CameraUpdateFactory.newLatLngBounds(builder.build(),
10));
            }catch (IllegalStateException e){
                //Excepcion: no han sido añadidos puntos a builder
                mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(new
LatLng(40.41, -3.69),5));}
            finally{
            //Eliminamos evento listener, ya se ha centrado, no se
debe volver a llamar
mapa.setOnCameraChangeListener(null);}
        }
    });

    Comparar_Recorrido(vId1,vId2, true);
}else{
    vInternet = false;
    //Solo Wifi y no está disponible
    //Eliminamos Fragment Mapa
    FragmentManager fmanager = getSupportFragmentManager();
    Fragment fragment = fmanager.findFragmentById(R.id.mapC);

fmanager.beginTransaction().remove(fragment).commitAllowingStateLoss()
;
    Toast.makeText(this, "No existe una conexi□□da de internet",
Toast.LENGTH_LONG).show();
    Comparar_Recorrido(vId1,vId2, false);
    }
}else
    Comparar_Recorrido(vId1,vId2, false);
```

El evento `onCameraChange` se utiliza para centrar el mapa una vez se ha añadido el trazo.

Seguidamente se leen los registros almacenados en la base de datos,

```
private void Comparar_Recorrido(int vId1, int vId2, boolean vMostrar){
try{
String[] ptos_gps;
int iCount=0;

ptos_gps = new String[2];

STSQLiteHelper stdb = new STSQLiteHelper(this,
"SportTracking_BD.s3db", null, 1);
SQLiteDatabase db = stdb.getReadableDatabase();
String sql,strimage;
sql = "Select RECTipo, strftime('%d-%m-%Y %H:%M:%S', RECFechaInicio)
As FechaIni, " +
        "strftime('%d-%m-%Y %H:%M:%S', RECFechaFin) As FechaFin,
RECTime, "
      + "RECDistancia,RECGps, RECConsumo "
      + "From  STRecorrido "
      + "Where RECId In (" + vId1 + "," + vId2 + ")";

Cursor cursor = db.rawQuery(sql, null);
```

```
while (cursor.moveToNext()) {
   iCount = iCount + 1;
   String strvelocidad;
   Integer vtipo = cursor.getInt(0);
   String vfechaIni = cursor.getString(1);
   String vfechaFin = cursor.getString(2);
   Double vtiempo = cursor.getDouble(3);
   Double vdistancia =  cursor.getDouble(4);//metros
   Double vConsumo = cursor.getDouble(6);
   double minutos = (vtiempo/1000)/60;//minutos
```

Y se muestran en los views que componen la activity,

```
LblImagenC2.setImageDrawable(this.getResources().getDrawable(imageReso
urce));
LblImagenC2.setScaleType(ScaleType.CENTER_INSIDE);

this.LblFechaIniC2.setText("Inicio: " + vfechaIni);
this.LblFechaFinC2.setText("Fin:    " + vfechaFin);
this.LblDistanciaC2.setText("Distancia: " +
FormatoNumero.format(vdistancia/1000) + " Km");
this.LblTiempoC2.setText("Tiempo: " + FormatoHora.format(vtiempo));
this.LblVelocidadC2.setText("Vel.: " + strvelocidad);
this.LblConsumoC2.setText("Consumo: " + FormatoNumero.format(vConsumo)
+ " kcal");
```

Debido a que los puntos gps se almacenan en un campo texto en la base de datos y puede contener varios miles de puntos, para su procesado se ha creado un hilo que evite la sensación de bloqueo de la aplicación,

```
private class TaskLeerPtos extends AsyncTask<Void, Integer, Boolean> {
String[] ptos_gps;

public TaskLeerPtos(String[] vecstr_ptos){
    this.ptos_gps = vecstr_ptos;
}

@Override
protected Boolean doInBackground(Void... params) {
    Leer_Ptos();
    return true;
}

private void Leer_Ptos(){
String[] linea;
String[] campos;
lstptos1 = new ArrayList<LatLng>();
lstptos2 = new ArrayList<LatLng>();
LatLng pto;
builder = new LatLngBounds.Builder();

for (int ip = 0; ip < ptos_gps.length; ip++) {
    //campos almacenados: lat##lng;lat##lng
    linea = ptos_gps[ip].split(";");
    if (linea!=null){
        //Al menos debe haber 2 puntos
        for (int i = 0; i < linea.length-1; i++) {
            campos = linea[i].split("##");
```

```
            if (campos.length==2){
                pto = new LatLng(Double.valueOf(campos[0]),
Double.valueOf(campos[1]));
                if (ip==0)
                    lstptos1.add(pto);
                else
                    lstptos2.add(pto);
                builder.include(pto);
            }
        }
    }
}}

@Override
protected void onPreExecute() {
    PrgDiag = new ProgressDialog(SportTracking_Comparar.this);
    PrgDiag.setTitle("Leyendo informaci□/span>");
    PrgDiag.setMessage("Espere por favor...");
    PrgDiag.setProgressStyle(ProgressDialog.STYLE_SPINNER);
    PrgDiag.setCancelable(false);
    PrgDiag.setIndeterminate(true);
    PrgDiag.show();
}

@Override
protected void onPostExecute(Boolean result) {
    PrgDiag.dismiss();
    Mostrar_Trazo();
}}
```

Se insertan los puntos de cada trazo a una lista separada con la que posteriormente se creará una polylinea que será añadida al mapa. Los puntos también se añaden a un objeto `builder` que se utiliza para centrar el mapa sobre todos los puntos que lo componen ejecutándose el evento anteriormente mencionado `onCameraChange`.

```
private void Mostrar_Trazo(){
try{

if (mapa!=null){
    //Existe mapa
    //Añadimos trazo recorrido 1
    PolylineOptions lineOptions1 = new
PolylineOptions().width(5).color(Color.BLUE);
    Polyline trazo1 = mapa.addPolyline(lineOptions1);
    if (lstptos1.size() > 0) {
        //Añadimos trazo
        trazo1.setPoints(lstptos1);
    }
    // Añadimos trazo recorrido 2
    PolylineOptions lineOptions2 = new
PolylineOptions().width(5).color(Color.RED);
    Polyline trazo2 = mapa.addPolyline(lineOptions2);
    if (lstptos2.size() > 0) {
        // Añadimos trazo
        trazo2.setPoints(lstptos2);
    }
}
}catch (Exception e) {
```

```
FicheroLog.Save("SportTracking_Comparar.Mostrar_Trazo:" +
e.toString());}
}
```

Se puede modificar el tipo de mapa a mostrar,

```
private void Tipo_Mapa(){
    try{
    //Función para modificar el tipo de mapa
        tipomapa = (tipomapa + 1) % 4;
        switch(tipomapa)
        {
            case 0:
                mapa.setMapType(GoogleMap.MAP_TYPE_NORMAL);
                break;
            case 1:
                mapa.setMapType(GoogleMap.MAP_TYPE_HYBRID);
                break;
            case 2:
                mapa.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
                break;
            case 3:
                mapa.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
                break;
        }
    }catch (Exception e) {
    FicheroLog.Save("SportTracking_Mostrar.Tipo_Mapa:" +
e.toString());}
}
```

5.3.9 Clase SportTracking_Mostrar.java

La clase *SportTracking_Mostrar* tiene básicamente la misma funcionalidad que la clase *SportTracking_Comparar*. Únicamente difiere que se muestra la información de un único recorrido tanto en pantalla como en google maps.

Está asociada a la activity *sporttracking_mostrar_activity.xml*.

6. Pruebas

Cómo se indicó al principio del proyecto, la metodología utilizada en el desarrollo de la aplicación ha sido el modelo en espiral.

El programa se va desarrollando en versiones incrementales produciendo con cada iteración una versión más completa del sistema diseñado. Así que con cada iteración se han realizado una serie de test unitarios antes de continuar con la siguiente.

Por ejemplo, para comprobar el ciclo de vida en una activty se ha forzado que se pudiese rotar la pantalla, comentando está línea en el *AndroidManifest.xml* en la activity en cuestión*:*

```
android:screenOrientation="portrait"
```

De este modo al cambiar la orientación de la pantalla Android crea de nuevo la activity así que hemos podido comprobar si el estado se ha guardado correctamente y se vuelve a restaurar sin problemas.

Una vez completado el desarrollo y habiendo superado todos los test unitarios vamos a detallar las pruebas finales que hemos realizado para comprobar el correcto funcionamiento de la aplicación.

Se han realizado 3 recorridos en coche con una distancia corta, media y larga de acuerdo a las actividades deportivas a la que va dirigida la aplicación.

Prueba 1: Distancia aproximada de 55 Km

En esta primera prueba comprobaremos la distancia entre la salida a la A7 en Xàtiva y la entrada a Valencia por Ausias March.

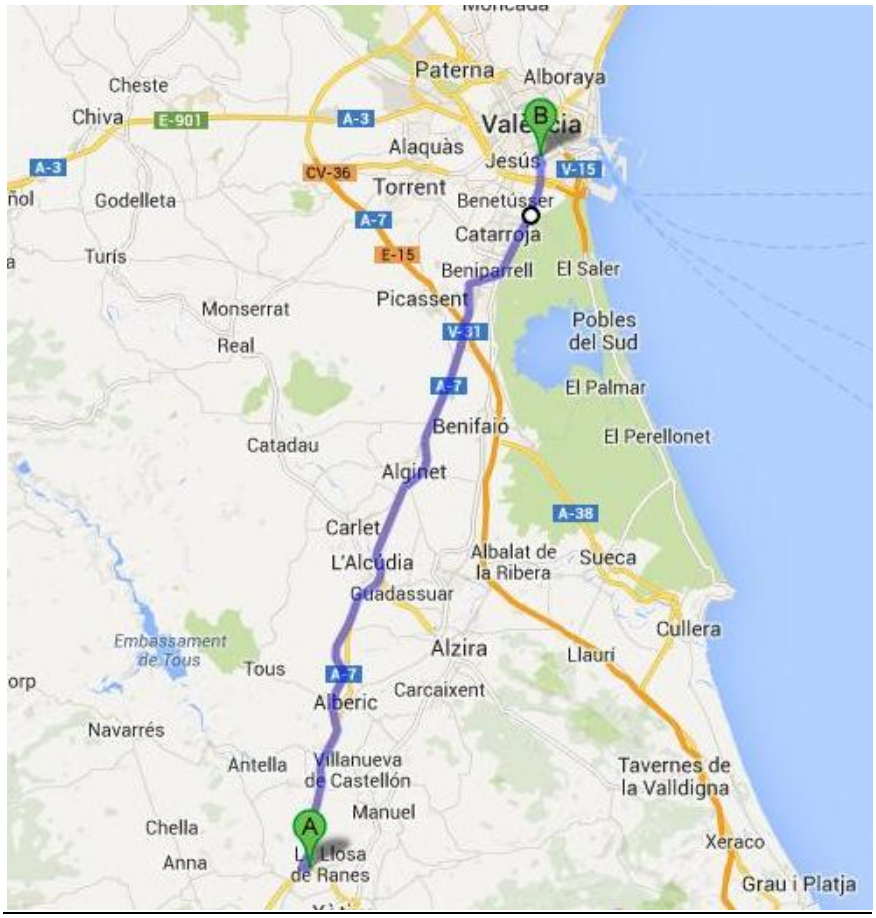

Ilustración 19 - Prueba 55 Km

Del trazado dibujado en Google Maps obtenemos una distancia de 54,3 Km.

A continuación, de nuestra aplicación hemos obtenido los siguientes resultados:

Ilustración 20 - Recorrido 55 Km aplicación

Obteniendo una distancia de 55,78 Km, con una diferencia aproximadamente de un 3% respecto al mismo trazado dibujado en Google Maps.

Prueba 2: Distancia aproximada de 25 Km

En esta segunda prueba comprobaremos la distancia entre el parque tecnológico en Paterna y la entrada a Valencia por la Av. de Ausias March.

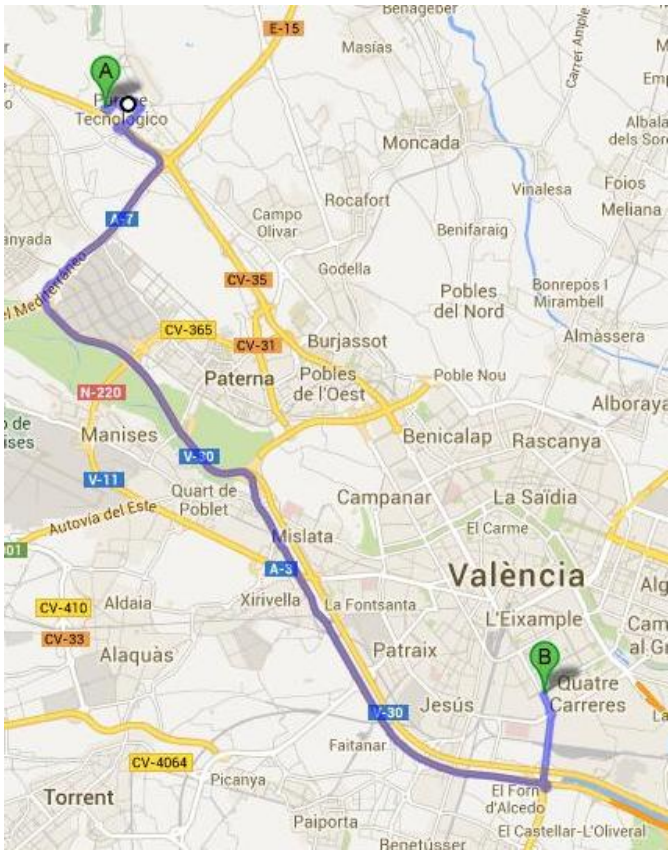

Ilustración 21 - Prueba 23 Km

Google Maps nos proporciona una distancia 22,5 Km.

El nuestra aplicación, realizando la misma ruta obtenemos los siguientes resultados:

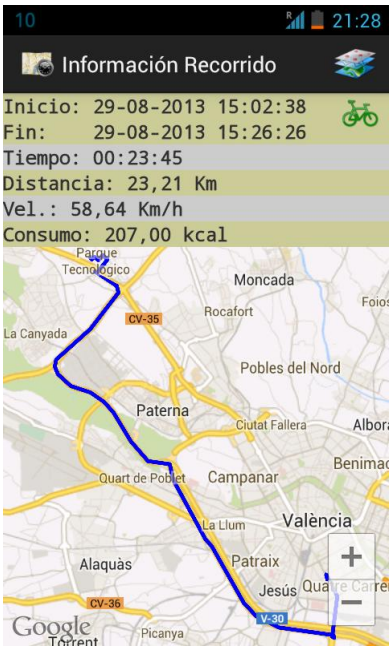

Ilustración 22 - Recorrido 23 Km aplicación

Obteniendo en este caso una distancia recorrida de 23,21 Km, es decir, una diferencia de entre un 3%-4% respecto a la distancia teórica.

<u>Prueba 3: Distancia aproximada de 5 Km</u>

La tercera y última prueba realizada sobre una distancia corta correspondiente a una zona interurbana en San Juan (Alicante).

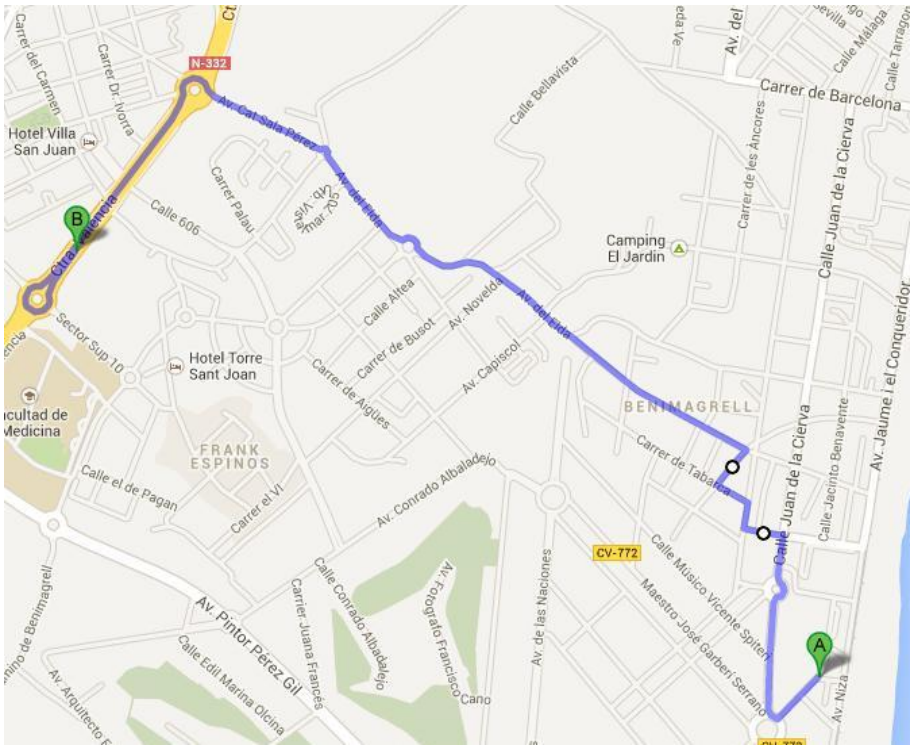

Ilustración 23 - Prueba 5 Km

Donde Google Maps calcula que la distancia recorrida es de 4,1 Km.

De nuestra aplicación obtenemos el siguiente resumen del mismo recorrido:

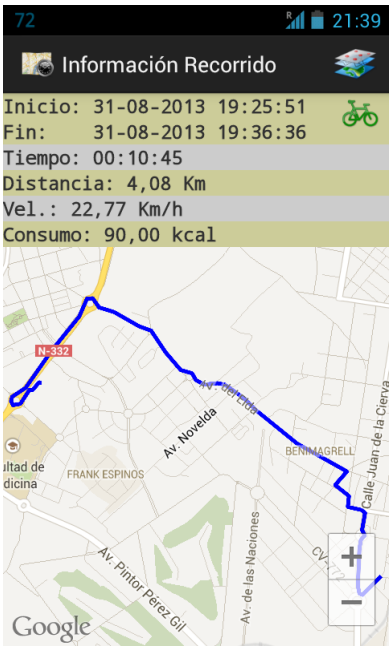

Ilustración 24 - Recorrido 5 Km aplicación

Siendo la distancia recorrida de 4,08, es decir, de una diferencia de 0,5% de la distancia teórica que nos proporciona Google Maps.

Tanto la velocidad como el consumo de calorías son calculadas a partir de variables como el tiempo, distancia y/o peso.

Todas estas pruebas se han realizado sobre un Huawei Ascend G300, un móvil que por sus características técnicas se podría catalogar como de gama baja en el momento actual.

Hemos tenido la posibilidad de realizar pruebas a pie con un móvil de gama alta como es el Samsung Galaxy S3 y poder comprobar cómo influye la calidad del receptor GPS ya que es vital para el cálculo de las distancias y los puntos que se obtienen para dibujar el trazo del recorrido realizado.

Recorrido a Pie 1

El siguiente trazado en Google Maps tiene una distancia de 1,1 Km

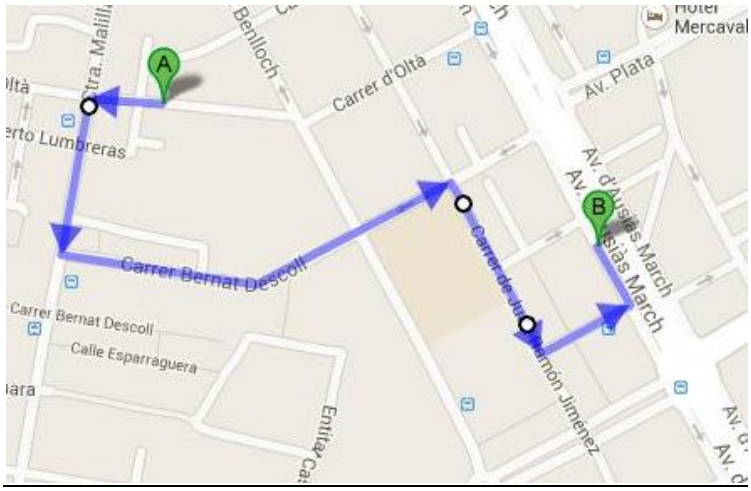

Ilustración 25 - Recorrido 1 teórico a pie

En nuestra aplicación sobre el Huawei hemos obtenido los siguientes resultados:

Ilustración 26 - Recorrido 1 a pie Huawei

Obteniendo una distancia de 1,07 Km, es decir, una diferencia de un 3% respecto a la distancia teórica.

En cambio en un Samsung Galaxy S3 hemos obtenido el siguiente recorrido:

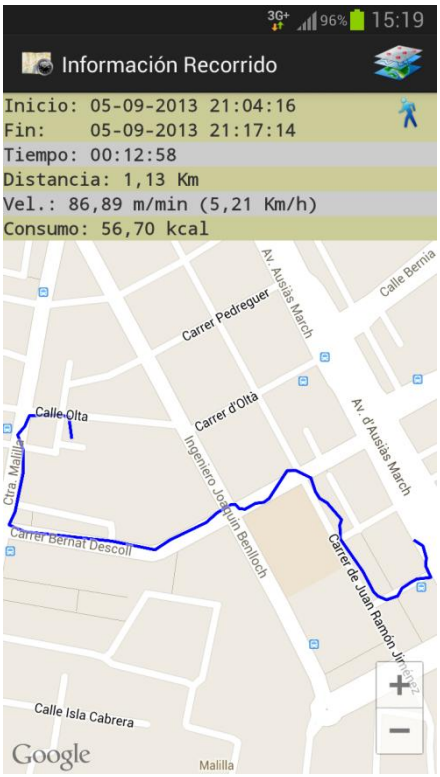

Ilustración 27 - Recorrido 1 a pie Samsung Galaxy S3

Obteniendo una distancia de 1,13 Km, es decir, igualmente una diferencia de un 3%. En cambio, el trazado, aún siendo muy parecido, es más exacto y el trazo de la línea es más continuo con menos variaciones.

Recorrido a Pie 2

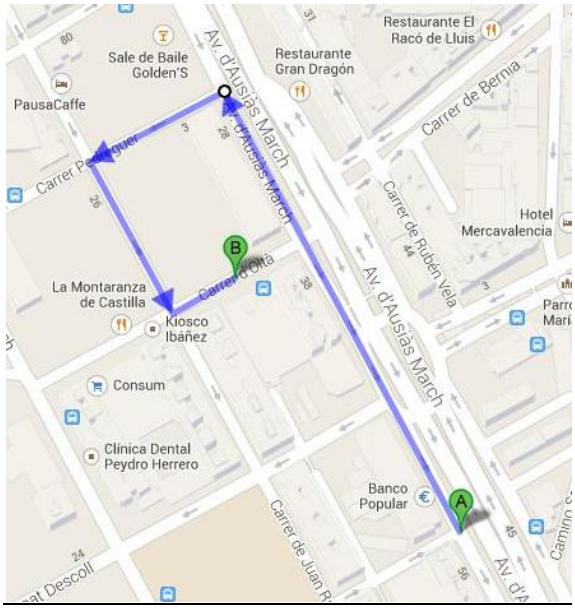

Ilustración 28 - Recorrido 2 teórico a pie

Teóricamente el siguiente recorrido comprende una distancia de 0,65 Km.

En nuestra aplicación, realizando el mismo itinerario hemos obtenido el siguiente resultado:

Ilustración 29 - Recorrido 2 a pie Huawei

Siendo la distancia de 0,70 Km, es decir, una diferencia de un 8% respecto a la distancia teórica. Esta mayor diferencia de todas las analizadas es achacable a que es la menor distancia recorrida y al principio el GPS a pesar de tomar señal necesita de un tiempo prudencial para que su precisión sea mayor.

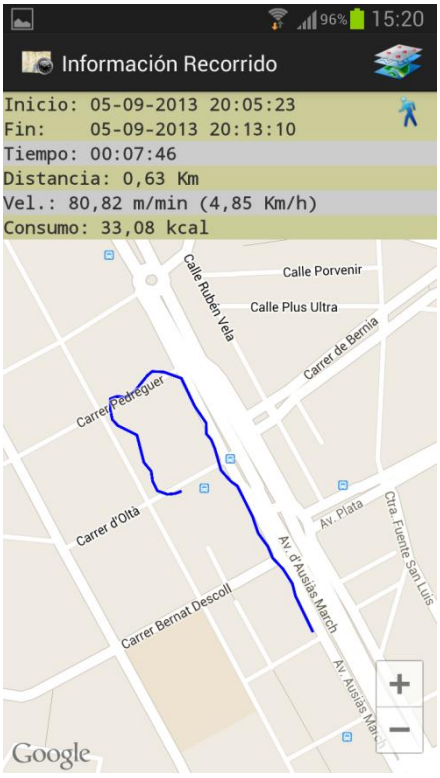

Ilustración 30 - Recorrido 2 a pie Samsung Galaxy S3

En cambio en el Samsung Galaxy S3 hemos obtenido una distancia 0,63 Km, aproximadamente una diferencia de un 3% de la distancia teórica. Un recorrido más preciso y lineal que el obtenido sobre la misma distancia realizado con el Huawei que entra dentro de la lógica por la diferencia de características técnicas entre ambos dispositivos. El receptor GPS del Samsung Galaxy S3 a parte de tomar la señal mucho antes, obtiene una mayor precisión en menos tiempo.

Después de todas las pruebas realizadas podemos concluir que nuestra aplicación tiene un diferencia inferior a un 5% en distancias medias y largas comparada con la distancia teórica, aumentando en distancias cortas pero siendo siempre inferior a un 10%. Aunque no de forma determinante, la calidad del receptor GPS puede influir en los datos que obtiene la aplicación.

7. Conclusiones

Varios fueron los objetivos que se plantearon al iniciar este proyecto, así que una vez finalizado estamos en predisposición de analizar si se han cumplido.

Se partía de unos conocimientos avanzados a nivel de usuario de la plataforma para dispositivos móviles Android, pero se han profundizado en su estudio de forma teórica. De este modo de forma breve hemos conocido la plataforma, su origen y cuál es su arquitectura.

Una vez asentada la base teórica de qué es Android internamente, se ha expuesto otros conceptos necesarios que hay que tener en cuenta antes de desarrollar una aplicación. Así hemos conocido los componentes que lo forman, unos inexistentes en otros sistemas y otros que aún teniendo una funcionalidad parecida su nomenclatura es diferente. Importante también es conocer las distintas versiones que existen, así como el ciclo de vida de una activity ya que de este modo podremos crear aplicaciones estables y compatibles con el mayor número de dispositivos.

Con la realización del proyecto nos ha permitido recuperar conceptos estudiados de la ingeniería del software. Una vez elegida la metodología a aplicar se ha procedido con ella con el análisis (con los requisitos y casos de uso), el diseño (con los diagramas de clase, de secuencia, base de datos e interfaz gráfica), implementación y pruebas.

Pero antes de realizar la implementación hemos tenido que conocer y preparar el entorno software, instalando Eclipse y configurándolo para hacer uso del SDK de Android, es decir, del kit de desarrollo de software que nos permite programar y depurar aplicaciones para Android.

Se ha desarrollado una aplicación para entrenos deportivos. En el mercado se pueden encontrar muchas y variadas aplicaciones sobre la misma temática, más completas y complejas, así que nuestra intención no era inventar algo sino realizar una aplicación sencilla con una gran funcionalidad.

Teniendo los conocimientos teóricos, con el desarrollo de la aplicación hemos conseguido afianzarlos. Nos ha permitido entender cómo se programa en Android, utilizar una base de datos SQLite, el API de Google Maps y el posicionamiento por GPS. Además, al programarse en Java, nos ha posibilitado recuperar un lenguaje de programación que no solíamos utilizar hoy en día, siendo de los más importantes.

Con todo, podemos concluir que se han cumplido sobradamente los objetivos que se plantearon en un principio.

Programando habitualmente en aplicaciones de escritorio, desarrollar una aplicación de Android destinada a dispositivos móviles no nos ha parecido tarea sencilla por determinadas particularidades.

Primeramente el entorno gráfico. Aunque Eclipse nos permite crear activities arrastando y soltando los views como los botones por ejemplo, el comportamiento no es el deseado y al final se opta por crear la interfaz por código siendo menos intuitiva y más costosa. La parte positiva que si un control no existe con la funcionalidad con mayor o menor esfuerzo es posible crearlo.

Otra de las particularidades, es que Android está diseñado para dispositivos móviles donde nos podemos encontrar multitud de tamaños de pantalla, desde 2" a más de 10". Las pantallas no se pueden redimensionar, se deberán redibujar las activities dependiendo del rango de resolución de las pantallas en que vayan a ser mostradas.

Controlar el ciclo de vida de las activities tampoco es tarea sencilla, ya que Android puede en cualquier momento destruir nuestra aplicación por escasez de recursos y se debe guardar el estado y volver a recuperarlo de una forma transparente al usuario.

Como se comentó la aplicación creada es una aplicación sencilla abierta a crecer con mayor funcionalidad. Posibles mejoras podrían ser:

- Disponer de un mayor número de actividades deportivas, mejorando los cálculos de distancias, posicionamientos y consumos dependiendo de la actividad realizada y la intensidad de la misma.
- Compartir nuestros recorridos y resultados en las redes sociales.
- Publicarla en el Play Store de Google

Aunque la informática es un mundo en constante evolución donde las cosas cambian a pasos agigantados, lo que hoy es moderno mañana será antiguo, lo que hoy domina el mercado mañana puede ser algo residual. Como profesionales del sector debemos estar en continua formación y la realización de este proyecto nos ha permitido enriquecer nuestros conocimientos y aprender a desarrollar aplicaciones en la plataforma móvil dominante en el mercado actual, Android. Esperemos que nos abra nuevos horizontes.

8. Manual de usuario

Al entrar la aplicación, la pantalla principal mostrará las principales opciones de la aplicación, pudiendo seleccionar entre configurar la aplicación, realizar un nuevo recorrido o listar los recorridos ya realizados.

Ilustración 31 - Pantalla Principal

8.1 Configuración

Primeramente deberemos configurar la aplicación, introduciendo los parámetros necesarios. Para ello deberemos seleccionando "Configuración" en el menú principal.

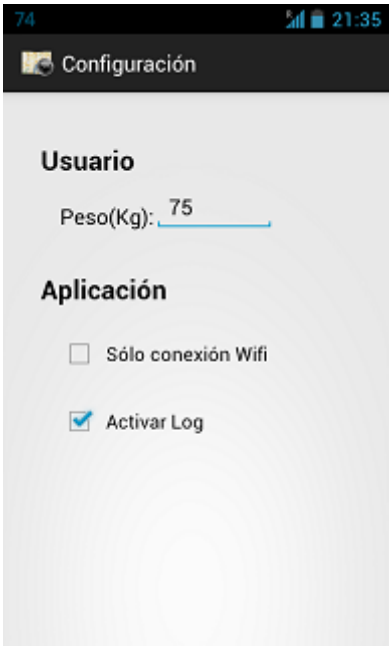

Ilustración 32 - Pantalla de Configuración

Se deberá indicar el peso en Kilogramos del usuario de la aplicación. Con este dato se aplicará la fórmula del consumo de calorías en la realización de la actividad física seleccionada.

Si deseamos que sólo se pueda utilizar la conexión Wifi al mostrar la información en un mapa de google maps deberemos activar la opción “solo conexión wifi”. De este modo evitaremos costes indeseados de nuestra tarifa de datos en la descarga de mapas. Si la opción está marcada y nuestro dispositivo no tuviese una conexión Wifi disponible al intentar visualizar un mapa aparecerá un mensaje de aviso “no existe conexión válida de internet”.

Activando el parámetro “Activar Log” se creará un fichero en la carpeta “Sport Tracking\Log” de la tarjeta de memoria de nuestro dispositivo con la traza y posibles errores en la ejecución de la aplicación necesarios para su posterior corrección o mejora. No se almacenará ningún dato privado ni característica del dispositivo.

8.2 Iniciar un nuevo recorrido

Una vez la aplicación ya ha sido configurada podremos iniciar un recorrido seleccionando “Recorrido” en la pantalla principal.

Primeramente deberemos seleccionar el tipo de recorrido que vamos a realizar, pudiendo seleccionar “Corriendo”, “Bicicleta” o “Caminando”.

Ilustración 33 - Pantalla selección recorrido

Si el GPS no estuviese activo en el dispositivo se mostrará un mensaje de aviso dándonos la posibilidad de activarlo desde la pantalla “Servicios de ubicación” de los ajustes del sistema.

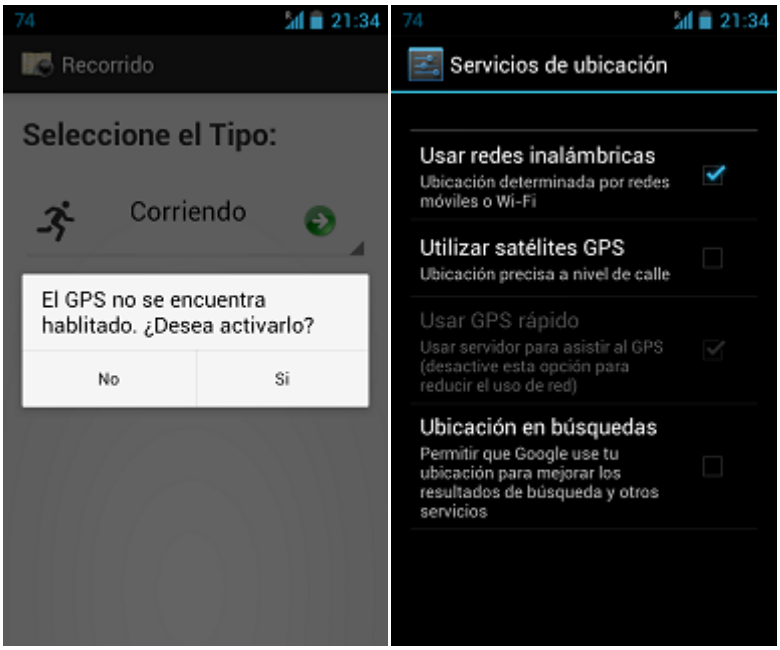

Ilustración 34 - Activación GPS

Una vez iniciado el recorrido el cronometro se activará. Se mostrará la distancia que se ha recorrido, así como la velocidad instantánea y el consumo de calorías hasta el momento. En todo momento es posible pausar o detener el recorrido.

Ilustración 35 - Pantalla tiempo

Del mismo modo aparecerá indicado si el GPS está activo y la precisión en metros del mismo. El receptor GPS puede tardar varios minutos en conseguir una señal válida de los satélites.

Si seleccionásemos  se mostrará un mapa con nuestra posición actual y el recorrido que estamos realizando:

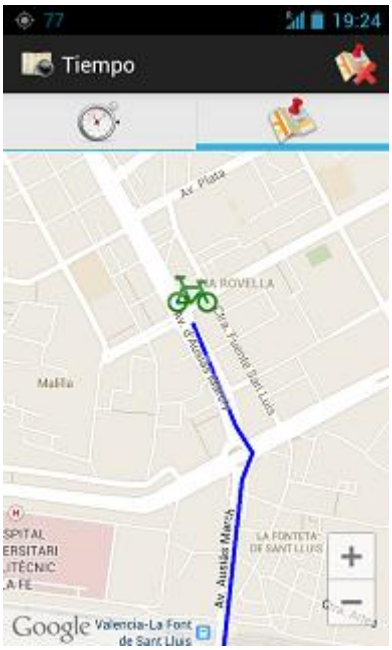

Ilustración 36 - Mapa con posición y recorrido

Si el mapa ha sido cargado, al seleccionar  se eliminará de la pantalla. De este reduciremos el posible consumo de datos de nuestra tarifa móvil que conlleva la descarga de mapas.

Una vez que lo consideremos oportuno podremos guardar el recorrido con el botón stop o con el botón atrás del dispositivo.

Ilustración 37 - Guardar recorrido

El mensaje que se muestra siempre será cancelable con el botón atrás del dispositivo.

8.3 Histórico de recorridos

Para consultar los recorridos realizados y almacenados en la base de datos deberemos acceder a la opción “Histórico” del menú principal.

Ilustración 38 - Pantalla Histórico

Desde aquí podremos eliminar uno o varios recorridos, así como visualizar la información almacenada de un recorrido en concreto o realizar un comparativo de recorridos.

Si se selecciona un recorrido se activará el botón  ,

Ilustración 39 - Selección de registro

Al pulsarlo nos mostrará cuando se realizo el recorrido, la distancia, la duración, la velocidad media y el consumo de calorías. Si se hubiesen almacenado puntos GPS y la conexión a internet está disponible se mostrará en un mapa dicho recorrido.

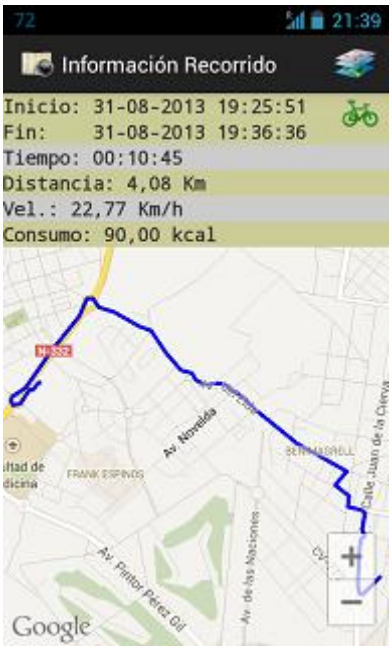

Ilustración 40 - Información recorrido

Si pulsásemos el botón  podremos modificar el tipo de mapa que se muestra, cambiando a satélite por ejemplo,

Ilustración 41 - Cambio tipo de mapa

De la misma forma, si se seleccionasen dos recorridos se activará el botón  ,

Ilustración 42 – Selección de dos recorridos

Pulsándolo podremos obtener en una nueva pantalla el comparativo de ambos

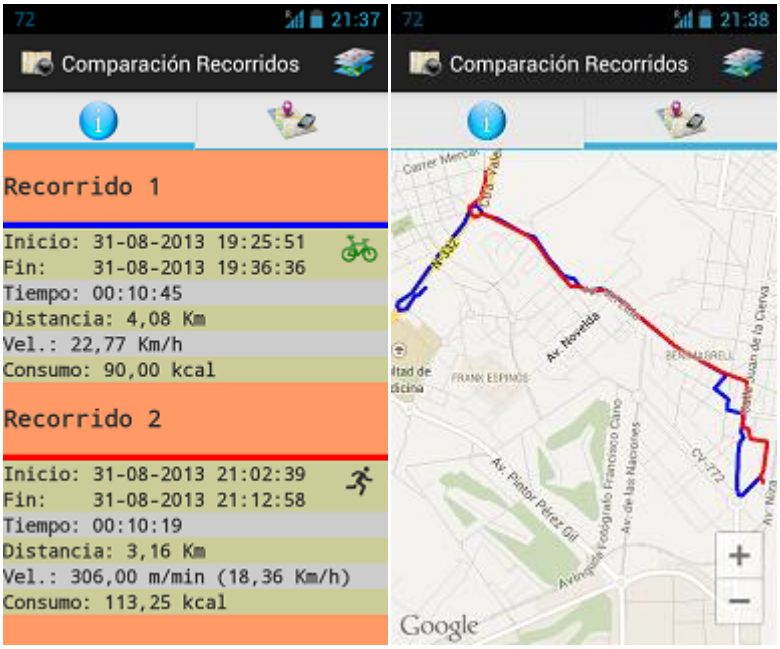

Ilustración 43 - Comparativo recorridos

Seleccionando uno o más de un recorrido tendremos la posibilidad de eliminarlos con el botón. Por ejemplo:

Ilustración 44 - Eliminación recorridos

Antes de eliminar cualquier recorrido siempre nos pedirá confirmación:

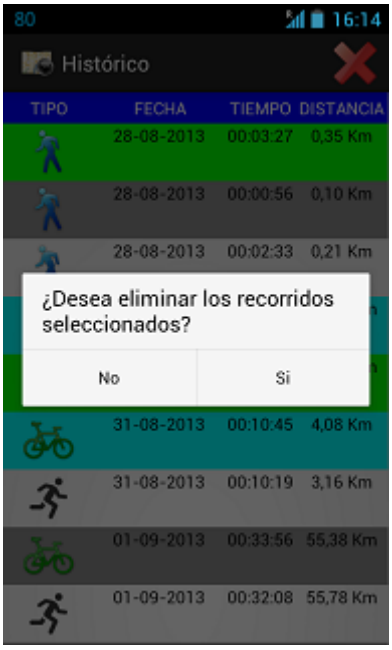

Ilustración 45 - Confirmación eliminación

Desde el botón menú del dispositivo podremos eliminar todos los recorridos no siendo necesario seleccionarlos uno a uno.

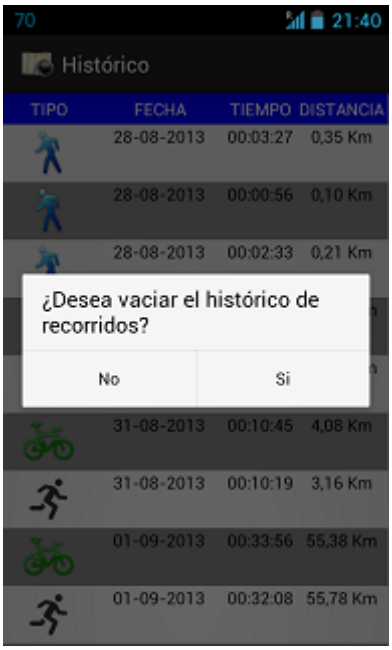

Ilustración 46 - Eliminar todos los recorridos

9. Bibliografía

Hashimi S., Komatineni S., *Pro Android,* Apress, 2009.

Karch ,M., *Android for Work Productivity for Professionals*, Apress, 2010.

Hashimi, S., Komatineni, S., MacLean, D., *Pro Android 2*, Apress, 2010.

Ableson, F., Sen R., King C., *Android in Action*, Manning Publications, 2011.

Gargenta, M., *Learning Android*, O'Reilly Media, 2011.

García de Jalón, J., Rodríguez, J.I., Mingo, I., Imaz A., Brazález A., Larzabal, A., Calleja, J., García, J., *Aprenda Java como si estuviera en primero*, Universidad de Navarra, 1999.

Google Maps Android API v2 Developer Guide, disponible en https://developers.google.com/maps/documentation/android/ (Agosto 2013).

SQLite, *SQLite Syntax*, disponible en http://www.sqlite.org/lang.html (Julio 2013).

Blog Independiente de Programadores Decantados, disponible en http://codecriticon.com/ (Julio 2013).

Gómez Oliver, S., *Curso Programación Android*, disponible en www.sgoliver.net (Junio - Septiembre 2013).

Cancela J., *Programando en Android,* disponible en http://javiercancela.com (Agosto 2013).

Pavón Pulido, N., *Componentes básicos de Android*, disponible en http://www.uhu.es/nieves.pavon/documentos/android/basicoandroid.pdf (Agosto 2013).

El baúl del programador, *Programación Android*, disponible en http://elbauldelprogramador.com (Agosto 2013).

Pamarke, *Componentes de una aplicación Android,* disponible en http://pamarke.com (Agosto 2013).

Androideity, *Programación Android*, disponible en http://androideity.com/category/programacion/ (Agosto 2013).

Plaza Alonso, N., *Estructura de un proyecto Android en Eclipse,* disponible http://www.weterede.com/2010/10/estructura-de-un-proyecto-android-en-eclipse/ (Agosto 2013).

Tur Badenas, S., *Android*, disponible en http://acacha.org/mediawiki/index.php/Android (Agosto 2013).

Geeky Theory, *Tutorial Android*, disponible en http://www.geekytheory.com/category/geeky-theory-2/tutoriales-2/android-2/ (Agosto 2013).

Droideando, *Tutorial*, disponible en http://droideando.blogspot.com.es/2011/02/tutorial-del-primer-proyecto-en-android.html (Agosto 2013).

Universidad Politécnica de Valencia, *Android: Programación de aplicaciones para móviles*, disponible en http://www.androidcurso.com/ (Agosto 2013)

Institut Puig Castellar, *Android: Ciclo de vida de las actividades y otros detalles,* disponible en http://elpuig.xeill.net/Members/vcarceler/misc/android/ciclo-de-vida-de-las-actividades-y-otros-detalles-de-la-plataforma/index_html (Agosto 2013).

Terminales Android, *Ciclo de vida en una actividad Android*, disponible en http://www.terminalesandroid.com/Ciclo-de-vida-de-una-actividad-en-android (Agosto 2013).

Medina M., *Android*, disponible en http://telekita.wordpress.com/category/android/s-o-android/ (Agosto 2013).

Wikipedia, *Android*, disponible en http://es.wikipedia.org/wiki/Android (Agosto 2013).

Jummp: Gestión de Proyectos y desarrollo del Software, disponible en http://jummp.wordpress.com/ (Agosto 2013).

Acerta Software, *Metodología de desarrollo en espiral,* disponible en http://www.acertasoftware.com/mspiral.html (Agosto 2013).

Rodríguez David, *Tecnología software libre y otras inquietudes,* disponible en http://davidjguru.com/ (Agosto 2013).

Blog de tecnología e ingeniería web, *Personalizar ListView en Android,* disponible en http://miltecnologia.blogspot.com.es/2013/04/personalizar-listview-en-android.html (Julio 2013).

NetRunners, *Usar nuestra propia base de datos SQLite en Android,* disponible en http://blog.netrunners.es/usar-nuestra-propia-base-de-datos-sqlite-en-android/ (Agosto 2013).

Android Developer, disponible en http://developer.android.com/index.html (Junio - Agosto 2013).

Stackoverflow, disponible en http://stackoverflow.com/ (Junio – Septiembre 2013).

Android Community For Application Development, disponible en http://androiddev.orkitra.com (Agosto 2013).

Carbajal A., *La nutrición en la red*, disponible en http://pendientedemigracion.ucm.es/info/nutri1/carbajal/manual.htm (Agosto 2013).

Android Developers Blog, disponible en http://android-developers.blogspot.kr (Julio 2013).

Tech Notes, *Android Dynamically Add rows to Table Layout,* disponible en http://technotzz.wordpress.com/2011/11/04/android-dynamically-add-rows-to-table-layout/ (Julio 2013).

Tutorialspoint, *SQLite Tutorial*, disponible en http://www.tutorialspoint.com/sqlite/index.htm (Julio 2013).