ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

FEET3

SEGUIMIENTO GEOGRÁFICO DIARIO AUTOMATIZADO DEL USUARIO

AUTOMATIC USER DAILY GEOGRAPHIC TRACKING

Realizado por
David Bandera García
Tutorizado por
Enrique Alba Torres
Francisco Javier Ferrer Urbano
Departamento
LENGUAJE Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Noviembre de 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: El análisis de nuestra actividad diaria puede ser una fuente de información muy valiosa para conocer características de nuestros hábitos cotidianos que pueden pasar desapercibidas. En el presente Trabajo Fin de Grado se pretende detectar de forma automática la rutina diaria del usuario mediante el registro de las localizaciones en las que se detiene. Mediante el uso de un dispositivo móvil Android podemos detectar estas paradas utilizando los sensores disponibles en la mayoría de los dispositivos móviles inteligentes (*smartphones*). Estas paradas se almacenarán de forma totalmente automática y de manera local en el dispositivo móvil del usuario, pudiendo ser consultadas en cualquier momento. Adicionalmente, el usuario puede solicitar registrar una parada en su posición de forma manual si lo desea. También se podrá obtener información sobre la fecha y hora de las paradas, además de disponer de información sobre las redes inalámbricas detectadas en cada parada, tanto fijas (*routers*) como otros dispositivos móviles cercanos. También se podrán consultar las paradas de forma gráfica en un mapa de la zona mediante el uso de la API GoogleMaps. La aplicación, además, se puede configurar para adaptarla a distintos usos, gracias a los parámetros de distancia mínima a la que se considera una parada como nueva y tiempo mínimo de detección para que se determine si se ha producido una parada. El objetivo final es hacer consciente al usuario de su propio comportamiento, para permitir mejoras en su vida.

Palabras claves: automatización, aplicación móvil, rutina, detección, GPS

Abstract: The analysis of our daily routine can be a source of very valuable information to learn about the characteristics of our daily habits that are not so obvious. This degree thesis aims to automatize the detection of the daily routine of the user by registering the locations where he/she stops. With an Android mobile device we can detect these stops by employing the sensors most smartphones have nowadays. These stops will be automatically and locally stored in the device and able to be consulted at any time. Additionally, the user will be able to register a stop at their current position manually, should they desire it. The user will be able to obtain information about the date and time of every stop, and consult information about the detected wireless networks at each stop, be it a fixed network (routers) or another nearby mobile device. Stops will also be shown graphically on a map through the use of GoogleMaps API. The application can be adjusted to different uses thanks to two parameters, the minimum distance to consider a stop as a new one and minimum detection time to determine if a stop has been made. The final goal is to make the user aware of his/her own routines, so as to improve his/her own live.

Keywords: automatization, mobile application, routine, detection, GPS

Índice

Capítulo 1. Introducción 1
1.1. Motivación 1
1.2. Objetivos 1
1.3. Estructura de la memoria 2
Capítulo 2. Tecnologías 3
2.1. Repositorio de código fuente 3
2.2. Aplicación móvil 3
2.2.1. Plataforma móvil 3
2.2.2. Base de datos 3
2.2.3. Bibliotecas externas 4
2.3. Entorno de desarrollo 4
Capítulo 3. Especificación y análisis 5
3.1. Requisitos funcionales de la aplicación Android 5
3.2. Requisitos no funcionales de la aplicación Android 6
3.3. Casos de uso 7
3.4. Base de datos 12
Capítulo 4. Diseño 13
4.1. Modelo relacional de la base de datos 13
4.2. Diagrama de clases 14
Capítulo 5. Implementación y pruebas 17
5.1. Estructura de la aplicación Android 17
5.2. Desarrollo del proyecto 19
5.2.1. Inicio 19
5.2.2. Solicitar almacenamiento de parada 20
5.2.3. Consultar información de parada 22
5.2.4. Edición de nombre 24
5.2.5. Visualización gráfica mediante mapa 25
5.2.6. Automatización 27
5.2.7. Preferencias 31
5.3. Pruebas 33

Capítulo 6. Conclusiones y líneas futuras 37
6.1. Conclusiones 37
6.2. Líneas futuras 38
Referencias 39
Apéndices 41
A. Manual de usuario 41
Índice de figuras 44
Índice de tablas 45

Capítulo 1. Introducción

1.1. Motivación

El número de usuarios de *smartphones* es cada vez más elevado (1). Sin embargo, gran parte de estas personas no aprovechan todo el potencial de dichos dispositivos, ya sea por desconocimiento o por la falta de necesidad. Dichos dispositivos son una gran fuente de información a la que se le puede sacar mucho partido. En las ciudades inteligentes la automatización y el análisis de datos cobran una gran importancia puesto que ayuda a la optimización de los recursos y el aumento de la productividad (2).

Las aplicaciones capaces de realizar un seguimiento de las actividades diarias del usuario son muchas, pero la mayoría necesitan datos introducidos por el mismo o *hardware* adicional, como en el caso de las pulseras de *fitness*.

En este Trabajo Fin de Grado (TFG) se pretende acabar con estas limitaciones aprovechando todo el potencial de los dispositivos móviles inteligentes, utilizando tanto la tecnología GPS como los sensores de movimiento y la interfaz WiFi. Vamos a crear una aplicación automática y fácil de usar para el seguimiento de la rutina diaria, permitiendo al usuario mantener un control de su horario y ayudándole a recordar las visitas realizadas con anterioridad.

1.2. Objetivos

El objetivo del presente TFG es el desarrollo de una aplicación móvil Android capaz de detectar la posición geográfica del usuario y determinar cuándo éste se ha detenido en algún sitio durante un tiempo significativo, para posteriormente registrar dicha parada junto con la fecha y hora en la que se realizó. Adicionalmente, se realizará un escaneo de redes inalámbricas existentes en dicho lugar, las cuales se clasificarán como redes de punto fijo (routers) o dispositivos móviles (otros *smartphones*). Posteriormente se podrán consultar en forma de lista o de forma gráfica mediante un mapa de la zona. Todo el proceso se realizará de forma automática mediante el uso de los sensores del dispositivo móvil y GPS. Además, el usuario podrá solicitar manualmente el registro de una parada en su ubicación actual.

1.3. Estructura de la memoria

La estructura de la presente memoria de TFG ha sido estructurada según los siguientes capítulos:

Capítulo 2: Tecnologías

En este capítulo se detallarán las tecnologías empleadas para el desarrollo de la aplicación, junto con las herramientas necesarias para ello, además de ofrecer una explicación sobre las decisiones tomadas.

Capítulo 3: Especificación y análisis

Se expondrán los requisitos del sistema mediante una lista de requisitos funcionales y no funcionales de la aplicación, además de los casos de uso posibles.

Capítulo 4: Diseño

En este apartado se ofrece una explicación sobre el funcionamiento de la aplicación mediante diagramas UML (3) y el diseño de la base de datos utilizada.

Capítulo 5: Implementación y pruebas

Aquí se verán detalladas las partes más importantes de la implementación de la aplicación Android junto con los problemas encontrados y pruebas realizadas.

Capítulo 6: Conclusiones y líneas futuras

Finalmente se ofrecerán las conclusiones extraídas del proyecto y las posibles vías explorables en un futuro.

Capítulo 2. Tecnologías

En este capítulo se explicarán las tecnologías y herramientas utilizadas en la elaboración del proyecto, además de una justificación de las elecciones realizadas.

2.1. Repositorio de código fuente

Se ha optado por el uso de un repositorio de código basado en Git, más concretamente la plataforma GitHub (4). Dicha plataforma es sencilla de usar y su rendimiento es mucho más que suficiente. Una funcionalidad muy interesante de esta plataforma es la de poder colaborar con otros desarrolladores de forma fácil, manteniendo un seguimiento de los cambios realizados. Esto nos permite continuar mejorando la aplicación en un futuro e incluso añadir colaboradores. El código de la aplicación Android desarrollada se encuentra alojado en dicha plataforma, al cual se puede acceder mediante el siguiente enlace.

-Enlace: https://github.com/Kotomaro/Feet3

2.2. Aplicación móvil

En esta sección procedemos a explicar las tecnologías y plataformas elegidas para el desarrollo de la aplicación móvil, y los motivos principales de estas elecciones.

2.2.1. Plataforma móvil

La aplicación móvil ha sido implementada para el sistema operativo móvil Android, ya que se trata del sistema operativo más extendido en dispositivos *smartphone* (5). Además, esta plataforma consta de una extensa documentación y gran soporte, lo que la convierte en una opción muy atractiva a la hora de elegir entre todos los sistemas operativos móviles existentes.

2.2.2. Base de datos

La decisión de utilizar Android como sistema operativo ha influido en el uso de SQLite (6) como sistema gestor de bases de datos. SQLite es relativamente sencillo de usar y posee una gran cantidad de documentación disponible sobre el mismo, además de su eficiencia a la hora de almacenar la información.

2.2.3. Bibliotecas externas

Para la implementación de algunas de las funcionalidades de la aplicación se han empleado bibliotecas externas de Google: Google Play Services (7) para el uso de la geolocalización por GPS y la detección de paradas de forma automática, y GoogleMaps Android API (8) para la representación gráfica mediante mapas del historial de paradas. Ambas bibliotecas requieren obtener una clave de Google para la autorización del uso de dichos servicios, la cual se puede obtener de forma gratuita visitando sus respectivos sitios web.

2.3. Entorno de desarrollo

El entorno de desarrollo utilizado en este proyecto ha sido Android Studio (9), incluyendo las herramientas de las que dispone, ya que se trata del entorno más conocido para el desarrollo de aplicaciones Android para móviles, y posee una amplia documentación. Esta plataforma incluye además multitud de herramientas para el desarrollo de este tipo de aplicaciones, gestionadas de forma automática por la misma plataforma, tales como las bibliotecas Java necesarias para las diferentes versiones del sistema operativo Android y los drivers necesarios para utilizar las APIs de Google mencionadas anteriormente.

Para la elaboración de los diagramas UML y esquemas de diseño de la base de datos se han utilizado dos herramientas:

- **Modelio** (10)**:** Se trata de un programa *open source* de diseño y documentación de proyectos *software* capaz de realizar una gran variedad de esquemas útiles en este ámbito, tales como diagramas UML, además de contar con un buen soporte mediante el uso de módulos opcionales. Los diagramas UML de clases y casos de uso se elaboraron mediante el uso de esta herramienta.
- **MySQL Workbench** (11)**:** Este programa ofrece una herramienta visual para el diseño y desarrollo de bases de datos. Permite realizar modelado de datos, desarrollo SQL e incluso posee herramientas de configuración de servidores de bases de datos. Proporcionado por Oracle, se puede obtener de forma gratuita en la página oficial de la herramienta.

Dichas herramientas fueron seleccionadas basándose en su facilidad de obtención y uso, además de su capacidad para producir buenos resultados, lo que las convirtieron en opciones muy atractivas y eficaces para el propósito deseado.

Capítulo 3. Especificación y análisis

Este capítulo describe los requisitos funcionales y no funcionales de la aplicación Android, además del diseño de la base de datos y los casos de uso del sistema necesarios para cumplir con los requisitos de la aplicación.

3.1. Requisitos funcionales de la aplicación Android

En este apartado se especificarán los requisitos funcionales pactados con los tutores para la aplicación Android, así como una breve explicación de cada uno. Estos requisitos especifican la funcionalidad del sistema, y son nombrados y descritos en la Tabla 1.

ID	Nombre	Descripción
RF-01	Detección de parada manual.	El usuario podrá solicitar la detección de su posición actual como parada.
RF-02	Detección de parada automática.	La aplicación detectará de forma automática las paradas realizadas por el usuario acorde a unos parámetros.
RF-03	Detección de redes.	La aplicación detectará redes cercanas cuando se registre una parada.
RF-04	Almacenamiento de parada.	La información sobre la posición, fecha y redes detectadas se almacenará en la base de datos local.
RF-05	Borrado del historial.	El usuario podrá eliminar el historial de las paradas almacenadas.
RF-06	Consulta de paradas.	El usuario podrá consultar información sobre las paradas en forma de lista, junto con las redes detectadas.
RF-07	Consulta gráfica de paradas.	El usuario podrá consultar sus paradas de forma gráfica representadas en un mapa.
RF-09	Editar nombre parada.	El usuario podrá editar el nombre de las paradas almacenadas.
RF-10	Editar nombre red.	El usuario podrá editar el nombre de las redes almacenadas.
RF-11	Ajustar parámetros de aplicación.	El usuario podrá ajustar los parámetros de tiempo y distancia mínimos para la detección de paradas.

Tabla 1 - Requisitos funcionales

Los requisitos han sido clasificados agrupándolos según el objetivo de cada uno de ellos:

- Detección: Requisitos relacionados con la detección de elementos o estados. En este grupo se encuentran los RF 01, 02, y 03.
- Gestión: Requisitos relacionados con la gestión de los datos e información almacenados en la aplicación. En este grupo se encuentran los RF 04 y 05.
- Consulta: Requisitos relacionados con la consulta de los datos por parte del usuario. En este grupo se encuentran los RF 06 y 07.

- Ajuste: Requisitos relacionados con los diferentes ajustes y personalizaciones que el usuario puede realizar a la aplicación. En este grupo se encuentran los RF 9, 10 y 11.

3.2. Requisitos no funcionales de la aplicación Android

En esta sección se describirán los requisitos no funcionales del sistema, los cuales establecen las condiciones y restricciones sobre las que tiene que funcionar el sistema. En la Tabla 2 incluimos la descripción de los requisitos no funcionales. Podemos observar que se ha tenido en cuenta la especificación de IEEE-Std 830-1993 para realizar la clasificación según categorías.

ID	Categoría	Descripción
RNF-01	Usabilidad	La aplicación debe ser lo más sencilla y cómoda de utilizar posible.
RNF-02	Fiabilidad	Los datos de la aplicación no deben verse comprometidos incluso si se produce un fallo durante la ejecución.
RNF-03	Fiabilidad	La aplicación debe seguir funcionando en segundo plano, incluso cuando otras aplicaciones se encuentren funcionando al mismo tiempo.
RNF-04	Mantenimiento	El código de la aplicación debe estar debidamente documentado, y los algoritmos empleados claramente explicados.
RNF-05	Documentación	La aplicación incluirá un manual de usuario en el que se detalle todo lo necesario para su uso correcto.
RNF-06	Almacenamiento	Los datos obtenidos deberán almacenarse en una base de datos local del dispositivo empleado.
RNF-07	Interfaz	La interfaz de usuario empleada debe ser adecuada para la actividad en uso
RNF-08	Rendimiento	La aplicación debe realizar una gestión adecuada de los recursos disponibles en el dispositivo.

Tabla 2 - Requisitos no funcionales

El cumplimiento de dichos requisitos resulta indispensable para asegurar la calidad del producto final. Es por tanto muy importante que estos requisitos sean correctos, no se contradigan entre si y sean realistas, es decir, se puedan alcanzar adecuadamente dentro de las restricciones del proyecto.

3.3. Casos de uso

En esta sección se muestra el diagrama de casos de uso del sistema ilustrado mediante la Figura 1. Este diagrama cubre todos los requisitos funcionales requeridos para el correcto y adecuado funcionamiento de la aplicación.

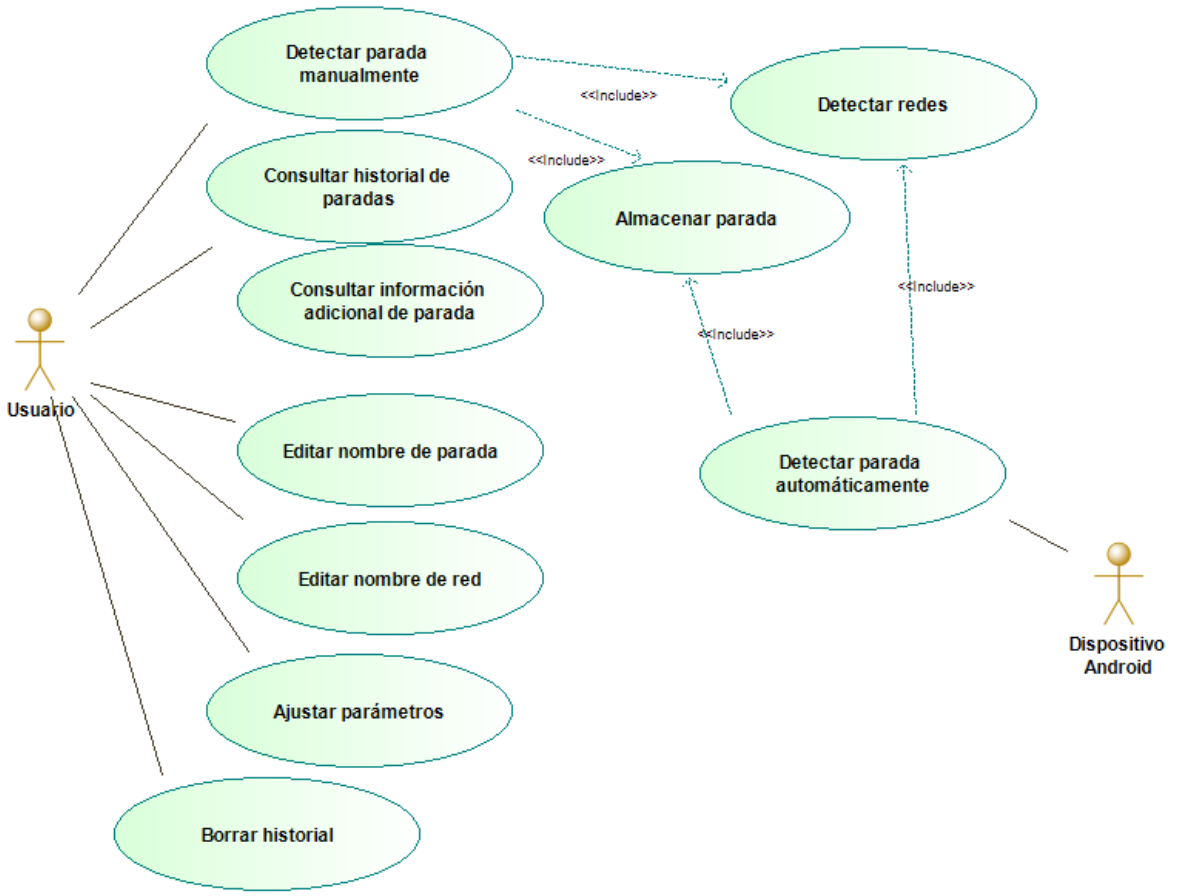

Figura 1 - Casos de uso

A continuación se procede a explicar con detalle cada uno de ellos, especificando el procedimiento seguido para su correcta conclusión y los escenarios alternativos en posibles casos de error. Cada caso de uso se muestra mediante una tabla en la que se detalla su identificador, los requisitos a los que se encuentra asociado, los actores involucrados, la descripción del mismo con sus precondiciones y postcondiciones y los posibles escenarios que pueden ocurrir.

En el Caso de uso 1 el usuario registra una parada de forma manual.

Detectar parada manualmente	
Identificador	UC-1.
Requisitos asociados	RF-01, RF-03, RF-04.
Actores	Usuario.
Descripción	El usuario solicita registrar una parada inmediatamente en su posición actual.
Precondiciones	La aplicación debe encontrarse en la pantalla principal.
Postcondiciones	La parada se almacena correctamente en la base de datos.
Escenario principal	1- El usuario pulsa el botón Guardar localización. 2- La aplicación detecta la posición geográfica actual. 3- La aplicación detecta las redes cercanas al dispositivo. 4- La parada generada se almacena en la base de datos del dispositivo.
Escenario alternativo	La geolocalización (GPS) está desconectada o no se encuentra disponible: 2a- La aplicación muestra un mensaje informando de que no se pudo obtener la localización.

Tabla 3: Caso de uso 1

En los Casos de uso 2 y 3 el usuario consulta la información recopilada sobre las paradas realizadas.

Consultar historial de paradas	
Identificador	UC-2.
Requisitos asociados	RF-06.
Actores	Usuario.
Descripción	El usuario consulta el historial de paradas guardado en el dispositivo.
Precondiciones	Debe existir al menos una parada guardada en la base de datos. La aplicación debe encontrarse en la vista principal.
Postcondiciones	Ninguna.
Escenario principal	1- La aplicación muestra una lista con el historial de paradas.
Escenario alternativo	Ninguno.

Tabla 4 - Caso de uso 2

Consultar información adicional de parada	
Identificador	UC-3.
Requisitos asociados	RF-06, RF-07.
Actores	Usuario.
Descripción	El usuario consulta información adicional sobre las paradas, y consulta el mapa de paradas.
Precondiciones	Debe existir al menos una parada almacenada en la base de datos. La aplicación debe encontrarse en la vista principal.
Postcondiciones	Ninguna.
Escenario principal	1- El usuario selecciona una parada de la lista de paradas de la vista principal. 2- Se muestra la vista información adicional de la parada. 3- El usuario pulsa el botón Mapa en la vista de información adicional de la parada. 4- La aplicación abre la vista de mapa en la que se muestran las paradas realizadas.
Escenario alternativo	Ninguno.

Tabla 5 - Caso de uso 3

Los Casos de uso 4 y 5 de uso permiten editar información de las paradas.

Editar nombre de parada	
Identificador	UC-4.
Requisitos asociados	RF-09.
Actores	Usuario.
Descripción	El usuario edita el nombre de una parada.
Precondiciones	Debe existir al menos una parada almacenada en la base de datos.
Postcondiciones	El nombre se actualiza correctamente en la base de datos.
Escenario principal	1 - El usuario selecciona una parada de la lista de paradas de la vista principal. 2 - Se muestra la vista información adicional de la parada. 3 - El usuario pulsa sobre el nombre de la parada. 4 – Se muestra la vista editar nombre de parada. 5 - El usuario edita nombre y pulsa el botón Guardar. 6 - El nombre se actualiza y guarda en la base de datos.
Escenario alternativo	El usuario introduce un nombre no válido: 5a - El usuario deja el nombre vacío y pulsa el botón guardar. 6a - No se realiza ningún cambio en la base de datos.

Tabla 6 - Caso de uso 4

Editar nombre de red	
Identificador	UC-5.
Requisitos asociados	RF-10.
Actores	Usuario.
Descripción	El usuario edita el nombre de una red (dispositivo fijo o móvil).
Precondiciones	Debe existir al menos una parada almacenada en la base de datos. Debe existir al menos una red asociada a una parada almacenada en la base de datos.
Postcondiciones	El nombre se actualiza correctamente en la base de datos.
Escenario principal	1 - El usuario selecciona una parada de la lista de paradas de la vista principal. 2 – La aplicación muestra la vista información adicional de la parada. 3 - El usuario pulsa sobre el nombre de una de las redes disponibles en la lista de redes. 4 - La aplicación muestra la vista de editar nombre de red. 5 - El usuario escribe el nuevo nombre y pulsa el botón Guardar. 6 - El nombre es actualizado y guardado en la base de datos.
Escenario alternativo	El usuario introduce un nombre no válido 5a- El usuario deja el campo de nombre vacío y pulsa el botón guardar. 6a - El nombre no se almacena y se mantiene la vista de editar nombre. El usuario retrocede sin pulsar el botón Guardar 6b - Se mantiene el nombre anterior sin realizar modificaciones a la base de datos.

Tabla 7 - Caso de uso 5

El siguiente Caso de uso 8 describe el ajuste de los parámetros de la aplicación por el usuario. La aplicación consta de dos parámetros, pero se ha decidido englobar los dos parámetros en un solo caso de uso, puesto que el escenario es exactamente el mismo desde el punto de vista del actor principal, el usuario.

Ajustar parámetros	
Identificador	UC-6.
Requisitos asociados	RF-11.
Actores	Usuario.
Descripción	El usuario ajusta los parámetros de configuración de la aplicación.
Precondiciones	Ninguna.
Postcondiciones	Los parámetros se actualizan correctamente.
Escenario principal	1 - El usuario pulsa el botón de preferencias de la vista principal. 2 - La aplicación muestra la vista de preferencias. 3 - El usuario pulsa en una preferencia para modificar. 4 - La aplicación muestra los valores posibles de dicha preferencia. 5 - El usuario elige el valor deseado. 6 - La aplicación actualiza correctamente el valor.
Escenario alternativo	Ninguno.

Tabla 8 - Caso de uso 6

En el Caso de uso 9 se describe el proceso de borrado del historial.

Borrar historial	
Identificador	UC-7.
Requisitos asociados	RF-05.
Actores	Usuario.
Descripción	El usuario borra el historial de paradas almacenado en la base de datos.
Precondiciones	Ninguna.
Postcondiciones	Los datos son borrados correctamente de la base de datos.
Escenario principal	1 - El usuario pulsa el botón Borrar historial. 2 - La aplicación muestra la vista de confirmación de borrado. 3 - El usuario selecciona la opción Aceptar. 4 - La aplicación borra los datos guardados en la base de datos.
Escenario alternativo	El usuario decide no borrar los datos 3a - El usuario selecciona la opción Cancelar 4a - La aplicación cierra la vista de confirmación de borrado sin alterar la base de datos.

Tabla 9 - Caso de uso 7

El último Caso de uso 10 describe cómo se detecta una parada de forma automática mediante el uso de los sensores del dispositivo móvil.

Detectar parada automáticamente	
Identificador	UC-8.
Nombre	Detectar parada automáticamente.
Requisitos asociados	RF-02.
Actores	Dispositivo móvil.
Descripción	La aplicación detecta automáticamente una parada.
Precondiciones	La aplicación se encuentra activa o en segundo plano.
Postcondiciones	La parada se almacena correctamente en la base de datos.
Escenario principal	1 - Los sensores del dispositivo móvil detectan que el usuario se ha detenido. 2 - La aplicación recibe la señal de que el usuario se encuentra detenido. 3 - Tras cumplirse los parámetros establecidos, la aplicación registra una parada.
Escenario alternativo	Ninguno.

Tabla 10 - Caso de uso 8

3.4. Base de datos

En este apartado se dará una breve explicación sobre la estructura de la base de datos que hemos decidido emplear en nuestra aplicación, ofreciéndose en el siguiente capítulo una explicación en mayor profundidad.

Para cumplir con los requisitos de la aplicación hemos optado por utilizar 4 entidades que van a almacenar la información que necesitará nuestra aplicación Feet3:

- **Position:** Representa un lugar físico, es la entidad utilizada para guardar información sobre la localización geográfica del usuario.
- **Devices:** Representa dispositivos inalámbricos, distinguiendo entre dispositivos de punto fijo o dispositivos móviles.
- **Finding:** Es la entidad más importante, relaciona las posiciones geográficas con los dispositivos encontrados en ellas, además de almacenar información de la parada realizada tal como la fecha y la hora.
- **Finding_has_Devices:** Se trata de una entidad que representa la relación entre varias paradas y varios dispositivos, ya que en el caso de un dispositivo móvil, este puede ser encontrado en diferentes posiciones.

Capítulo 4. Diseño

4.1. Modelo relacional de la base de datos

Para almacenar los datos de forma persistente, se ha empleado una base de datos basada en SQLite. En nuestra aplicación esta información se guarda de forma local en el dispositivo donde se encuentre instalada la aplicación. En la Figura 2 podemos observar el modelo relacional de la base de datos:

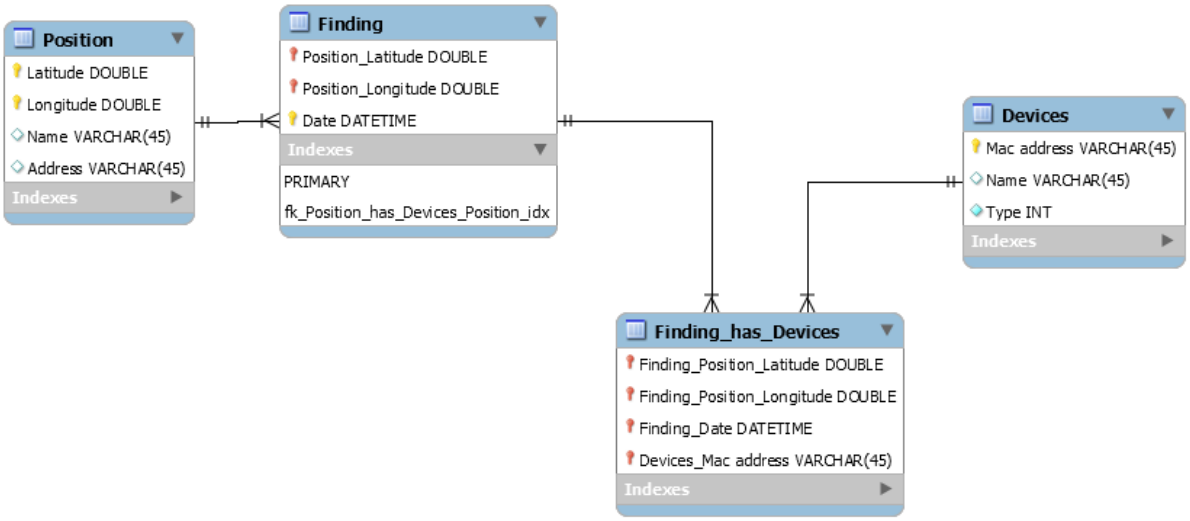

Figura 2 - Modelo relacional de la base de datos

Las entidades mostradas en la Figura 2 son detalladas a continuación:

- **Position:** Representa una posición geográfica. Sirve para registrar los lugares físicos en los que se detecta una parada.
  - Los valores **Latitude** y **Longitude** consisten en las coordenadas geográficas del lugar, y son usados como clave primaria para esta tabla, por lo que son valores obligatorios. Son registrados como valores reales.
  - **Name:** Es el nombre asignado a dicha posición, introducido por el usuario si lo desea. Es un campo opcional, por lo que puede encontrarse vacío.
  - **Address:** Consiste en la dirección que la API GoogleMaps devuelve para dicha localización. Se trata de una cadena de caracteres, y también es totalmente opcional.

- **Devices**: Representa un dispositivo de red inalámbrico, tanto fijo como móvil.
    - **MAC address:** Contiene la dirección MAC del dispositivo detectado, y es usado como clave primaria para la tabla. Su tipo es de cadena de caracteres y es de carácter obligatorio.
    - **Name:** De nuevo, es el nombre asignado al dispositivo. Por defecto se asigna el SSID del dispositivo detectado, pero puede ser alterado por el usuario. Igualmente se trata de una cadena de caracteres opcional.
    - **Type:** Sirve para determinar si un dispositivo es fijo o móvil. Es un valor booleano obligatorio.

- **Finding:** Representa una parada, asociando una posición geográfica con los dispositivos que han sido encontrados en dicha posición.
    - Los valores **Position_Latitude** y **Position_Longitude** son claves foráneas a la tabla *Position* para referenciar la posición detectada. Son guardados en forma de valores enteros obligatorios.
    - **Date:** Almacena la fecha y hora en la que se detectó la parada mediante un *datetime*, y es el identificador principal de esta tabla, por lo que es un valor obligatorio.

  Cabe mencionar que esta tabla se relaciona con los dispositivos mediante la tabla *Finding_has_Devices*.

- **Finding_has_Devices:** Esta entidad representa la relación de muchos a muchos entre una parada y los dispositivos que se hayan detectado. Todos sus parámetros son claves foráneas a las respectivas tablas.

4.2. Diagrama de clases

En esta sección se muestra el diagrama de clases UML de la aplicación Android desarrollada en la Figura 3. Se han incluido únicamente las clases más relevantes del sistema, obviando clases para implementación de interfaces de usuario en Android.

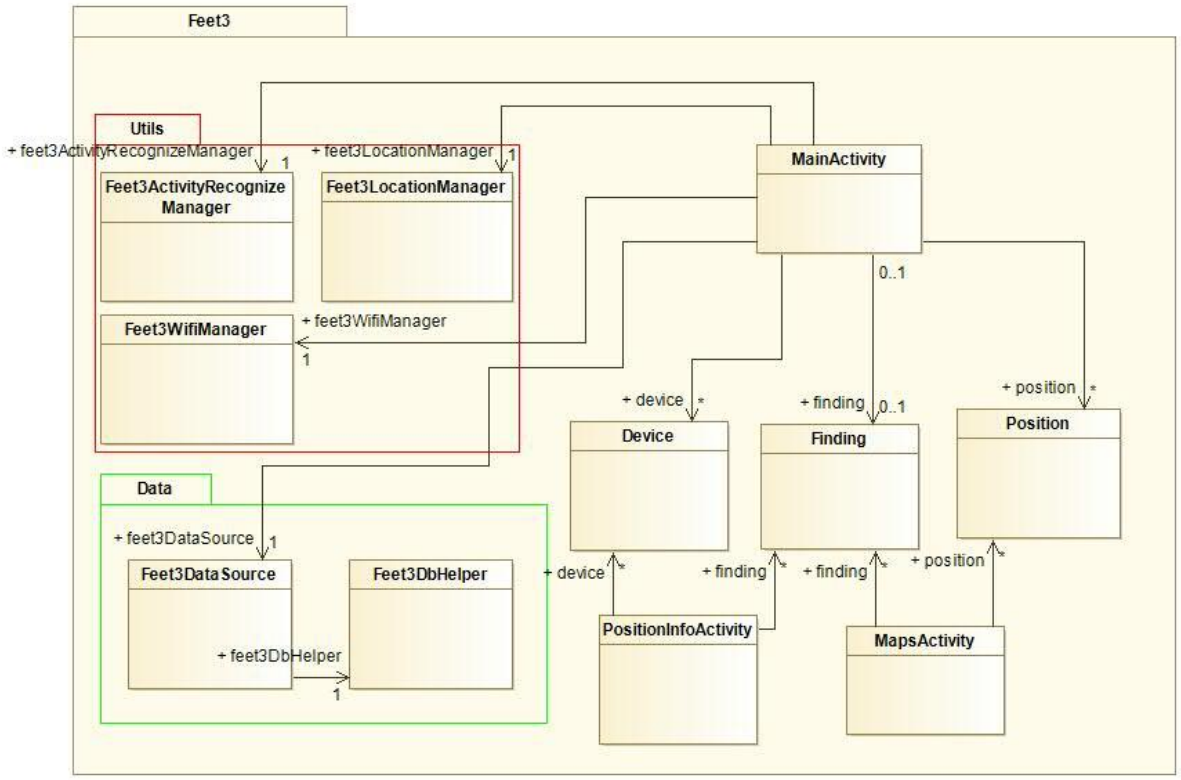

Figura 3 - Diagrama de clases

La aplicación ha sido dividida en tres grupos, un paquete general llamado Feet3 en el cual se encuentran las clases de lógica de aplicación y los otros dos paquetes. Las clases que implementan funcionalidades y utilidades están situadas en el paquete *Utils*, remarcado en color rojo, y las clases de gestión de la base de datos en el paquete *Data*, remarcado en color verde. Las clases más destacables del diagrama son las siguientes:

- **MainActivity:** Gestiona la mayor parte de la aplicación, empleando funcionalidad necesaria de las clases del paquete *Utils* y gestionando la detección e inclusión de las paradas en la base de datos mediante las clases representantes de las entidades y la clase gestora de datos Feet3DataSource.
- **Feet3DataSource:** Contiene todos los scripts de creación de la base de datos, las sentencias *sql* necesarias para realizar peticiones a la base de datos y la lógica para gestionarlas.
- **Feet3WifiManager:** Gestiona la detección de redes WiFi cercanas y obtiene información sobre ellas, pudiendo devolver una lista con la información.
- **Feet3ActivityRecognizeManager:** Es la clase encargada de la detección automática de paradas mediante el uso de los sensores del dispositivo móvil.

- **Feet3LocationManager:** Clase responsable de gestionar la posición geográfica del dispositivo, mediante el uso de GPS o redes móviles.

Con esta estructura de la aplicación se ha conseguido implementar la totalidad de los requisitos funcionales mencionados anteriormente.

Capítulo 5. Implementación y pruebas

En este capítulo se detalla la estructura de la aplicación Android junto con los detalles de la implementación más críticos e importantes. Finaliza con una recopilación de las pruebas llevadas a cabo.

5.1. Estructura de la aplicación Android

La siguiente imagen muestra la estructura de la aplicación Android del proyecto. El paquete principal, llamado *main.feet3*, contiene las clases principales de la aplicación además de los paquetes de las clases encargadas de la gestión de datos y de ofrecer utilidades y funciones.

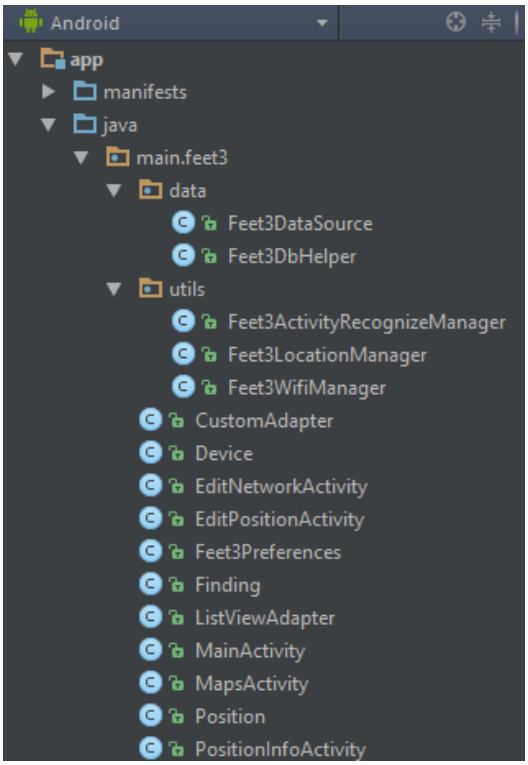

Figura 4 - Estructura de la aplicación Android

Procederemos a dar una explicación detallada sobre la funcionalidad de las clases esenciales para el funcionamiento de la aplicación.

- **Data**: Es el paquete que contiene las clases gestoras de la base de datos.
    - **Feet3DataSource:** Es la clase encargada de realizar las peticiones a la base de datos. Cuando otra clase necesita interactuar con la base de datos, lo hace a través de ésta clase.
    - **Feet3DbHelper:** Se encarga de generar la base de datos en el dispositivo Android, utilizando los scripts de creación encontrados en la clase Feet3DataSource.

- **Utils:** Este paquete contiene las clases encargadas de aportar herramientas necesarias para el correcto funcionamiento de la aplicación Android.
    - **Feet3ActivityRecognizeManager:** Esta clase se encarga de la detección de la actividad del usuario. Es capaz de detectar si el dispositivo se encuentra en reposo durante un tiempo determinado o si éste se encuentra en movimiento.
    - **Feet3LocationManager:** La función de esta clase es la de detectar la localización geográfica del usuario mediante el uso del GPS del dispositivo móvil.
    - **Feet3WifiManager:** En esta clase se encuentra la funcionalidad que permite detectar redes inalámbricas WiFi cercanas y generar una lista con la información asociada a cada una de ellas.

- En el paquete principal *main.feet3* se encuentran además las clases que contienen la lógica de la aplicación, así como clases necesarias para el correcto funcionamiento de la interfaz del proyecto Android.
    - **Device:** Clase representante de la entidad *Device* de la base de datos, es decir, las redes detectadas por la aplicación.
    - **Position:** Clase representante de la entidad Position de la base de datos, necesaria para representar las localizaciones físicas en las que se encuentra el usuario.
    - **Finding:** Clase representante de la entidad *Finding* de la base de datos. Con ella podemos relacionar las posiciones y los dispositivos encontrados, además de asignar una fecha y hora a dicho encuentro.
    - **EditNetworkActivity:** Se encarga de mostrar la interfaz de edición de nombre de un dispositivo o red, y llama a la clase *Feet3DataSource* para guardar de forma persistente el cambio.
    - **EditPositionActivity:** Igual que la clase anterior, contiene la interfaz de edición de nombre de una posición y la lógica necesaria para el correcto almacenamiento del cambio en la base de datos.
    - **Feet3Preferences:** Esta clase gestiona las preferencias de la aplicación, aportando una interfaz con la que el usuario puede interactuar para elegir los valores deseados, y almacenando los cambios realizados en el archivo *SharedPreferences* del proyecto Android. Este archivo sirve para gestionar fácilmente los diferentes parámetros de la aplicación entre las actividades de la misma.

- **MainActivity:** La clase principal de la aplicación Android. En esta clase se inicializan todas las demás clases necesarias para el funcionamiento de la aplicación, además de generar la vista principal de la misma. También contiene la lógica para el funcionamiento del historial de paradas y accesos a las demás vistas de la aplicación, como el menú de preferencias o la vista de información adicional de la parada. Además, en ella se encuentra el método principal para el almacenamiento de una parada, el cual es llamado cuando se pulsa el botón Guardar localización por el usuario o se detecta automáticamente una parada.
- **MapsActivity:** En esta clase se encuentra la funcionalidad necesaria para mostrar un mapa mediante el uso de la API GoogleMaps, además de mostrar todas las paradas registradas en la base de datos de forma gráfica en el mapa.
- **PositionInfoActivity:** Se encarga de mostrar todas las fechas en las que se detectó una parada en una determinada localización y las redes y dispositivos detectados en cada una de ellas, permitiendo cambiar entre diferentes fechas, además de permitir la edición de nombres de paradas y redes. También contiene el acceso para la vista gráfica de las paradas sobre un mapa.
- Las clases **CustomAdapter** y **ListViewAdapter** son necesarias para la implementación de las vistas de la aplicación Android. Permiten personalizar el funcionamiento de elementos como listas o tablas.

5.2. Desarrollo del proyecto

En esta sección se hablará sobre el proceso seguido para la implementación del proyecto, los problemas encontrados y las soluciones que se han decidido utilizar para resolverlos. Se hablará también del funcionamiento de la aplicación completa, siguiendo el flujo de procesos y actividades generados por la misma. Dado que la aplicación funciona fundamentalmente de forma automática, se explicará la lógica de los algoritmos empleados para este propósito.

5.2.1. Inicio

La primera vista que aparece cuando se inicia la aplicación es la vista de la actividad principal *MainActivity*, la cual se puede observar en la Figura 5.

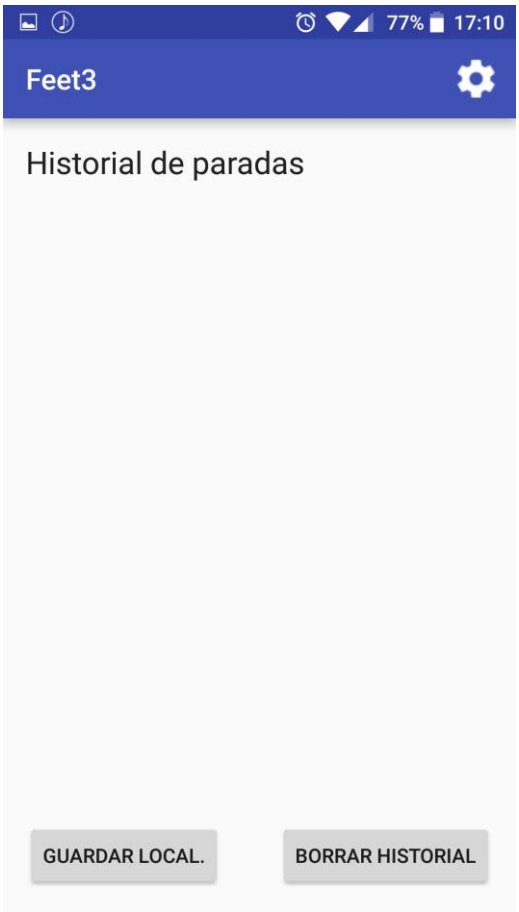

Figura 5 - Main Activity

En esta pantalla podemos observar los botones de guardar localización, borrar historial y preferencias (el engranaje en la parte superior derecha de la pantalla). Siendo esta la primera vez que se inicia la aplicación, el historial de paradas se encuentra vacío.

5.2.2. Solicitar almacenamiento de parada

El usuario puede solicitar el almacenamiento de una parada en la posición actual en la que se encuentra pulsando el botón Guardar localización. Este botón solicita inmediatamente un escaneo de las redes inalámbricas cercanas y la obtención de la posición geográfica actual. Una vez ha finalizado, la nueva parada se muestra en la lista del Historial de paradas tal como se muestra en la Figura 6.

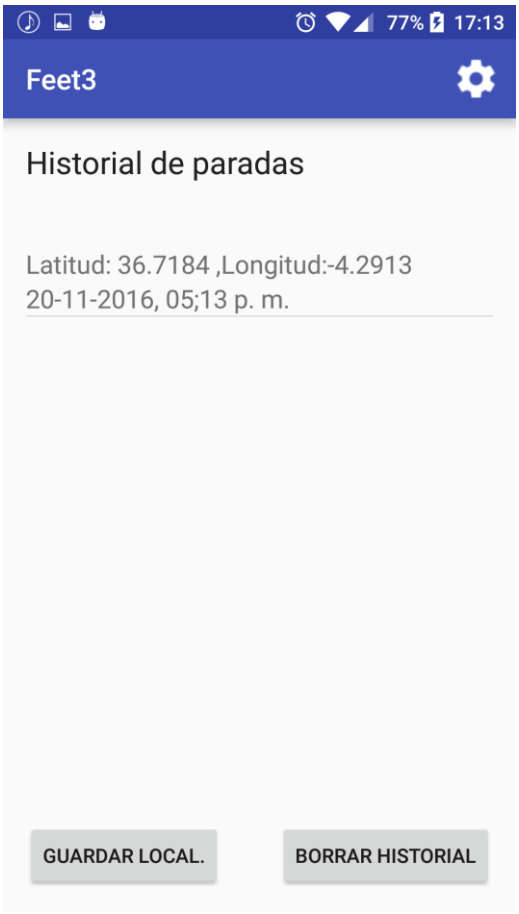

Figura 6 - Historial de paradas

La parada almacenada muestra información relevante sobre ella. Por defecto, se ha elegido la latitud y la longitud de la posición detectada como nombre de la parada. Además, se muestra la fecha y la hora en la que se realizó dicho parada.

Coordinar la obtención de la posición geográfica con el escaneo de redes inalámbricas presentó ciertos problemas, ya que el escaneo de las redes inalámbricas es realizado por el sistema operativo Android, el cual en un intervalo de tiempo no determinado devuelve la lista de redes obtenidas. Esto conllevaba en un principio a la obtención de listas de redes inalámbricas vacías siempre que se realizaba un primer escaneo, ya que el sistema operativo aún no había devuelto la lista poblada a la hora de insertar la información en la base de datos.

Se probó a realizar una espera mientras el servicio del sistema operativo recopilaba las redes utilizando el método *Thread.sleep()*, pero rápidamente se descartó esta opción debido a que mientras se esperaba a la obtención de resultados, la aplicación se encontraba completamente congelada. La solución obvia a este problema consistió en convertir la operación de búsqueda de redes en un *Thread* independiente, el cual esperaría a obtener los resultados del sistema

operativo sin bloquear la funcionalidad de la aplicación durante ese tiempo. Esa operación se encuentra ilustrada en el siguiente fragmento de código.

```
new Thread(new Runnable(){//this operation is done in a thread to avoid
    //freezing the UI while waiting for results
    public void run(){
        while(wifiManager.isScanFinished() == false){

            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        if(wifis == null || wifis.size() == 0){
            //No wifi detected, insert only the finding
            fDataSource.insertFinding(f);
        }else {
            //insert finding and associate with detected devices
            fDataSource.insertFindingWithDevices(f, wifis);
        }
```

Código 1 - Thread de detección de redes

El tiempo de espera depende del dispositivo móvil en el que se ejecute la aplicación. Los dispositivos de los que dispuse para probar tardaron alrededor de dos segundos, por lo que se eligió ese valor como tiempo de *tick* para comprobar la finalización de la búsqueda.

5.2.3. Consultar información de parada

Una vez disponemos de paradas registradas en nuestra base de datos, podemos consultar toda la información disponible sobre ellas. Para ello sólo tenemos que pulsar la parada de la cual deseemos ver la información y se mostrará la vista de información adicional observable en la Figura 7.

Figura 7 - Información adicional

En esta pantalla se puede ver, de nuevo, el nombre de la parada en la parte superior izquierda (en este ejemplo se muestra el nombre por defecto, las coordenadas de la posición). Tenemos dos listas navegables, el Historial de paradas y las Redes detectadas. En la lista del Historial de paradas se muestran todas las fechas en las que se detectó una parada en esa misma *Position*, mientras que la lista Redes detectadas muestra todas las redes inalámbricas que se detectaron en esa parada (esa posición en la fecha determinada). Por defecto se muestran las redes de la parada que se seleccionó en la vista principal. La figura 8 muestra cómo el usuario puede seleccionar otras fechas en la lista del Historial de paradas para ver las redes que se detectaron en esa fecha en concreto.

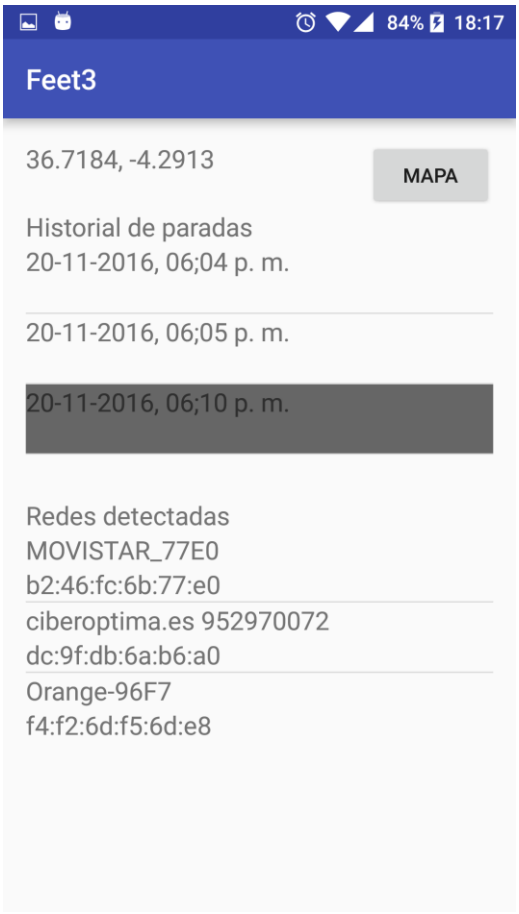

Figura 8 - Información adicional remarcada

La parada seleccionada es remarcada mediante un fondo oscuro para que el usuario pueda saber fácilmente cuál se encuentra seleccionada.

5.2.4. Edición de nombre

El usuario puede editar el nombre de posiciones y redes que se hayan detectado. Para ello, sólo tiene que pulsar sobre el nombre de la parada en la pantalla de información adicional de parada mostrada anteriormente, o en el elemento de la lista de Redes detectadas deseado. Esta acción abrirá un *Dialog* en el que se muestra el valor actual, y un cuadro de texto editable en el que el usuario puede escribir el nombre que desee. Dicho *Dialog* se puede apreciar en la Figura 9. Cuando el usuario finalice, solo tiene que pulsar el botón Guardar. En caso de que el usuario retroceda sin pulsar el botón o introduzca un nombre vacío, no se realizará ningún cambio.

Figura 9 - Edición de nombre

5.2.5. Visualización gráfica mediante mapa

Como se ha podido observar en las imágenes anteriores, en la misma vista de información adicional existe un botón llamado Mapa. Al pulsar este botón, se abre la vista de mapa en la que se muestran las paradas de forma gráfica, la cual es mostrada en la Figura 10.

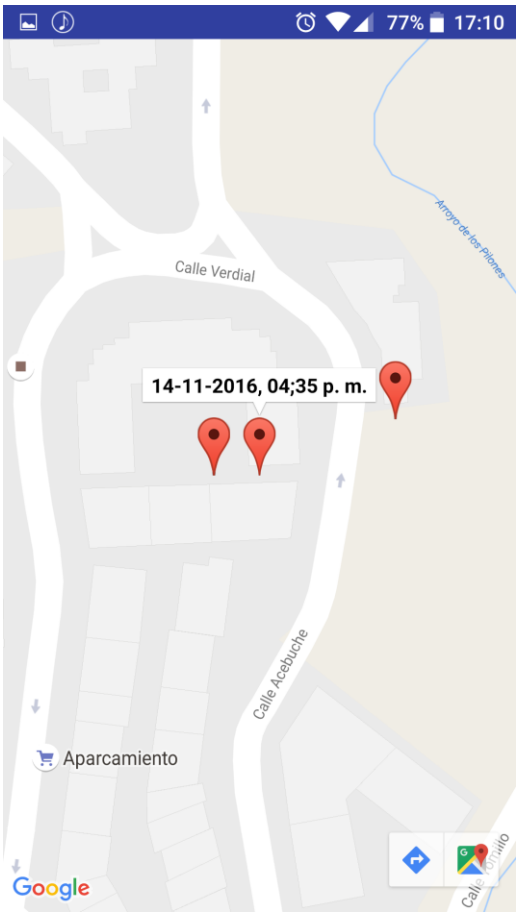

Figura 10 - Visualización mediante mapa

Las paradas son representadas mediante marcadores posicionados en el mapa en los lugares donde se detectaron. Al seleccionar un marcador aparece información sobre la fecha y la hora en la que se realizó esa parada y el nombre de la posición, en caso de que el usuario haya decidido editar el nombre por defecto.

El usuario puede moverse libremente por el mapa y hacer zoom para visualizar marcadores que queden fuera del área mostrada en todo momento, como se puede observar en la Figura 11. Por defecto, el mapa se centra en la primera parada realizada.

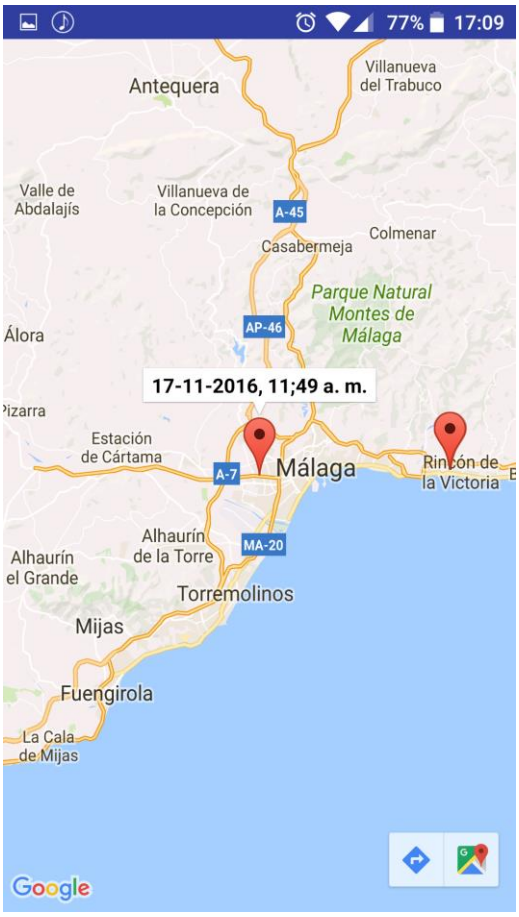

Figura 11 - Mapa alejado

5.2.6. Automatización

Hasta ahora hemos explicado las acciones que el usuario puede realizar de forma manual, pero gran parte del trabajo de este proyecto ha sido dirigido para conseguir que la aplicación Android detecte paradas de forma automática.

Una de las acciones más complejas de automatizar fue detectar cuándo el usuario se ha detenido en un lugar, y no confundir una parada con simplemente una corta detención (como por ejemplo esperar en un semáforo).

Para detectar la actividad del usuario, se ha empleado la API de GooglePlayServices, la cual ofrece un servicio de detección de actividad mediante el uso de los sensores y acelerómetros de los dispositivos móviles actuales. Esta API es capaz de reconocer diversas actividades que el usuario podría estar realizando y devolver una lista con todas las posibles actividades. Las actividades van acompañadas de un intervalo de confianza para indicar la probabilidad de que realmente se esté realizando esa actividad.

El proceso de detección automática se lleva a cabo en la clase *Feet3ActivityRecognizeManager*. Cuando esta clase es inicializada, se realiza una conexión con la API de Google mediante el uso de un *GoogleApiClient,* tal como se muestra en el Código 2.

```
private GoogleApiClient mGoogleApiClient;

public Feet3ActivityRecognizeManager(Context context, Activity activity){

    this.context = context;
    this.activity = activity;
    mainActivity = (MainActivity) activity;

    SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(context);
    String aux = preferences.getString("stop_detection_time", String.valueOf(defaultTime));

    time = Integer.parseInt(aux);
    mGoogleApiClient = new GoogleApiClient.Builder(context)
            .addApi(ActivityRecognition.API)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .build();

    startConnection();


}

private void startConnection(){
    mGoogleApiClient.connect();
}
```

Código 2 - Constructor Feet3ActivityRecognizeManager

Una vez inicializado el constructor, se procede a llamar al método *connect()*, el cual realiza las operaciones necesarias para conectar con la API y llama al método *onConnected()* que podemos ver en el Código 3.

```
@Override
public void onConnected(@Nullable Bundle bundle) {
    Intent intent = new Intent(context, ActivityRecognizedService.class);
    PendingIntent pendingIntent = PendingIntent.getService(context, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
    ActivityRecognition.ActivityRecognitionApi.requestActivityUpdates(mGoogleApiClient, time, pendingIntent);
}
```

Código 3 - Método onConnected

La acción que se realiza aquí es solicitar que los resultados de la detección de actividad sean enviados a la subclase *ActivityRecognizedService*. El parámetro *time* será el que determine la frecuencia con la que se realicen los análisis de actividad, del cual hablaremos más adelante.

La clase *ActivityRecognizedService* escuchará las respuestas de la API y las mandará a un método llamado *handleDetectedActivies*, mostrado en el Código 4. Este método es el responsable de la automatización de la detección de las paradas.

```
private static void handleDetectedActivities(List<DetectedActivity> activityList){

    for(DetectedActivity activity: activityList){
        if(activity.getType() == DetectedActivity.STILL && activity.getConfidence() >= 90){
            if(stopped){//if we detect being stopped twice in a row , mark as stop

                moving = false; //we are not moving
                if(!registered) {//check if we already registered the stop. If not, register it
                    mainActivity.registerStop();
                    registered = true;//mark this stop as registered
                }else{
                    //if it was already registered, don't register again

                }
            }else{//we detected a stop for the first time, change the variable to true so
                  //we can detect if we were stopped for a while
                stopped = true;
            }

        }else{
            if(moving = true) {//If we detected we were moving twice, we are moving, reset
                //the user is not stopped, so reset every variable
                stopped = false;
                registered = false;
            }else{//we detected movement once, set moving to true
                moving = true;
            }

        }
    }
}
```

Código 4 - Algoritmo de automatización

Este método recibe una lista de actividades posibles en *activityList*. Contamos con 3 variables booleanas para ayudarnos a detectar la parada. Estas variables son *stopped, moving* y *registered,* inicializadas a falso. La lista recibida es iterada, y por cada elemento se comprueba si se trata de la actividad *still* (el dispositivo se encuentra quieto, la situación típica en la que se ha dejado encima de una superficie como una mesa o el usuario está sentado con su móvil en el bolsillo). Si en efecto la actividad detectada se trata de *still* y tenemos una confianza del 90% o mayor, entraremos por la primera condición. En ese momento se comprobará si la variable *stopped* se encuentra a cierto o falso. Esta variable representa si en la última detección que realizamos se detectó al usuario como *still* o no. En el caso de que sea falsa, se cambia a verdadera y se finaliza el proceso.

En la siguiente comprobación la variable *stopped* se encuentra a verdadero. Si volvemos a detectar *still* como actividad realizaremos una comprobación de la variable *registered*. Esta variable sirve para controlar si la parada en la que nos encontramos ya ha sido registrada o no y evitar registrar la misma parada repetidas

veces (esto es, sin habernos puesto realmente en movimiento). En el caso de que la parada no esté registrada (*registered* es falso) se llamará a la función *registerStop()* la cual almacenará la parada en la base de datos de forma normal. Tras esta operación la variable *registered* es cambiada a verdadero para evitar registrar la parada de nuevo. A partir de este momento todas las detecciones consecutivas de *still* no realizarán ninguna acción.

Vayamos al caso contrario. Cuando la actividad detectada no es *still* o no tiene al menos un 90% de confianza entramos por la segunda parte de la condición. En ese momento entra en acción la variable *moving,* la cual es comprobada. En esta situación se encuentra en falso, ya que es inicializada a falso y también es declarada como falso cuando entramos por la rama de detección de *still*. La siguiente acción es declarar la variable *moving* como verdadero sin realizar ninguna otra acción. El objetivo de esta variable es evitar que dejemos de considerar una parada en el caso de que el usuario decida usar su dispositivo móvil momentáneamente (como por ejemplo, mirar la hora o consultar mensajes) sin que realmente haya cambiado de lugar. Esta variable nos indica si se ha detectado dos veces seguidas que el usuario se encuentra en movimiento.

Si en la siguiente comprobación de actividad se vuelve a detectar al usuario en movimiento, la variable *moving* se encuentra ahora a verdadero, por lo que entramos por la otra rama en la cual se reinician las variables *stopped* y *registered* y dejamos de considerar que aún permanecemos en la parada detectada. Tras esta acción el proceso lógico queda reiniciado al estado inicial (en el que esperamos que el usuario se detenga para registrar una nueva parada).

En el caso de que el usuario haya realizado una parada y ésta se haya registrado correctamente, las variables *stopped* y *registered* se encontrarían en verdadero y la variable *moving* en falso. Si el usuario utilizase su dispositivo móvil se detectaría que el usuario no se encuentra *still*, por lo que la variable *moving* sería declarada a verdadero. Sin embargo, si el dispositivo móvil vuelve a encontrarse en reposo se detectaría la actividad *still,* por lo que como se puede ver en la imagen la variable *moving* sería reiniciada a falso y se seguiría considerando la misma parada.

El intervalo en el que se realizan las comprobaciones de actividad es determinado por la variable *timer* vista anteriormente, la cual puede ser determinada por el usuario mediante el uso del menú de preferencias, tal como será explicado en el siguiente apartado

5.2.7. Preferencias

El usuario puede ajustar dos parámetros de la aplicación. Para ello, se debe encontrar en la vista principal de la aplicación Android visible en la Figura 12 y pulsar el botón con forma de engranaje situado en la parte superior derecha de la pantalla.

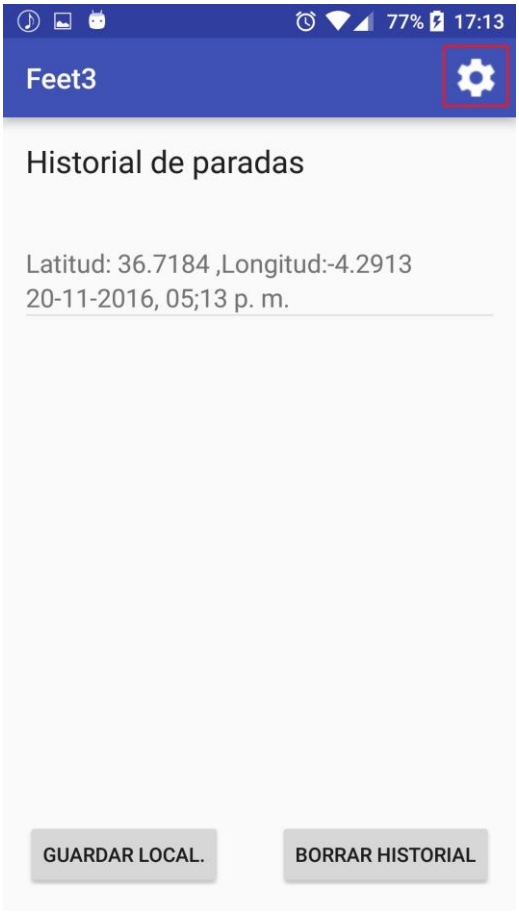

Figura 12 - Botón de preferencias

Tras pulsar el botón se abrirá la vista de Preferencias en la cual se muestran los dos valores que pueden ser ajustados tal como se muestra en la Figura 13.

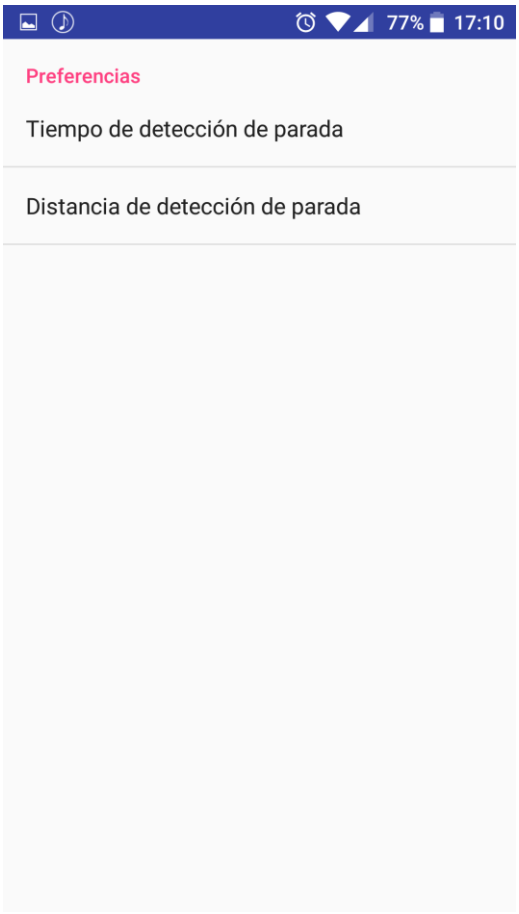

Figura 13 – Preferencias

En las figuras 14 y 15 se visualizan ambos menús desplegables. El primer valor se trata del Tiempo de detección de parada, el cual determina cada cuánto tiempo se realizará una comprobación automática de actividad para la detección de paradas automática. Este valor se podrá ajustar en intervalos de un minuto, y tomará valores entre 1 minuto y 10, siendo 6 minutos el valor por defecto. Para ello solo hay que pulsar sobre el texto y elegir el valor deseado.

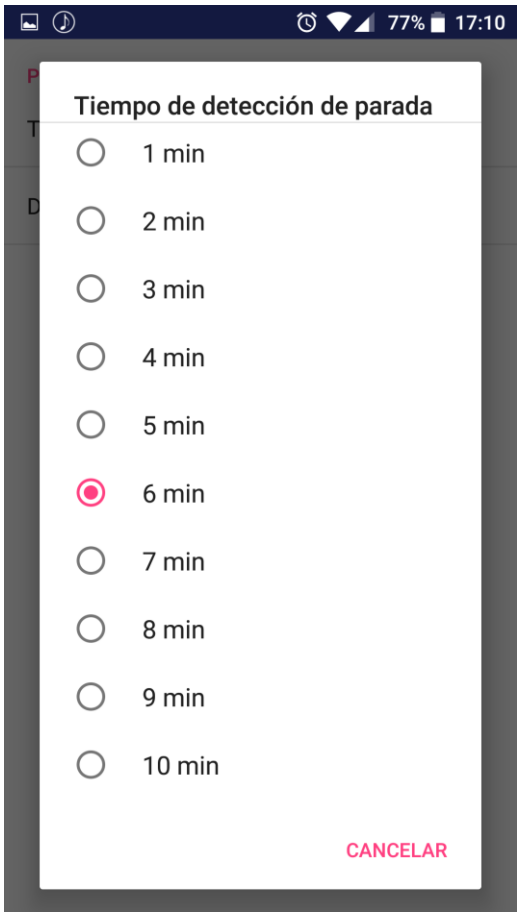

Figura 14 - Tiempo de detección de parada

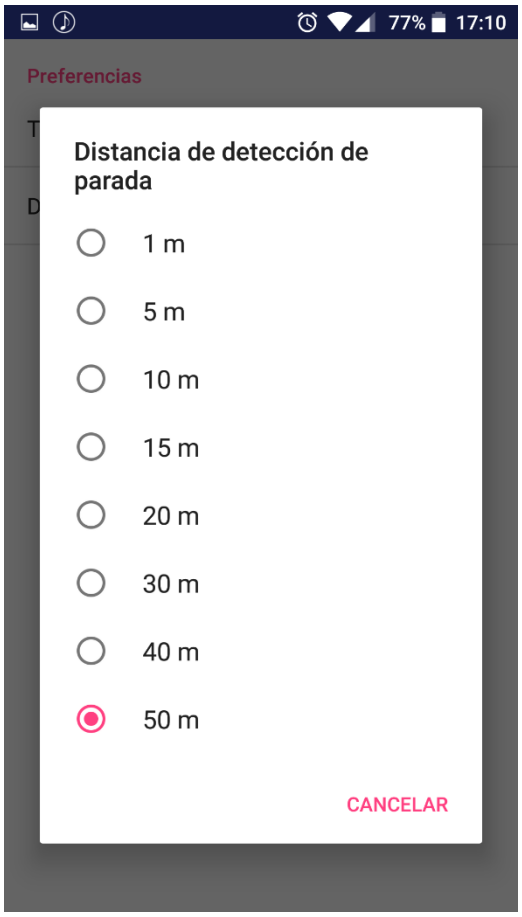

Figura 15 - Distancia de detección de parada

El segundo parámetro se trata de la Distancia de detección de parada. Este valor determina a qué distancia mínima las paradas serán consideradas realizadas en la misma posición o se guardarán como una posición nueva. Este parámetro podrá tomar valores entre 1 metro y 50 metros, siendo 20 metros el valor por defecto, y para cambiarlo solo se debe pulsar el texto y elegir el valor deseado. Cabe destacar que este parámetro afectará tanto a las paradas manuales como a las automáticas.

5.3. Pruebas

La realización de pruebas de una aplicación Android que emplea geolocalización y mapas resulta complicada y costosa en tiempo. Los emuladores que se suelen emplear para esta plataforma no tienen la capacidad de simular

localizaciones geográficas correctamente. Por esta razón, las pruebas realizadas han consistido en el uso de un dispositivo móvil real y el empleo de la aplicación durante una rutina diaria normal.

Recorrido con pausas

La Figura 17 muestra una prueba de un recorrido realizado, en el cual se realizaron dos paradas de 10 minutos en un banco de la calle, además del uso normal durante todo el día.

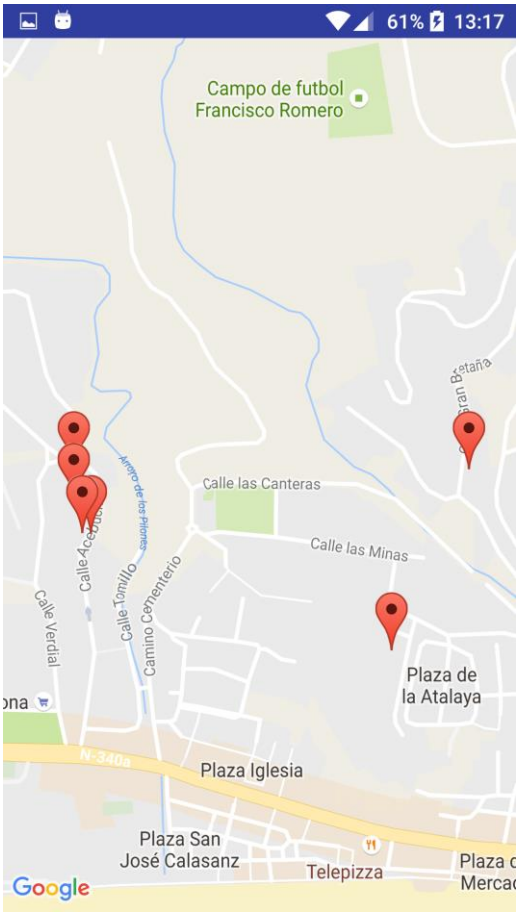

Figura 16 - Prueba 1

Se puede apreciar una aglomeración de paradas en la parte izquierda de la imagen, a la vez que dos paradas más separadas en la parte derecha. Las dos paradas independientes corresponden a las paradas realizadas durante el recorrido a pie, por lo que se puede observar que la aplicación detecta correctamente cuándo el usuario se detiene y cuándo se pone en movimiento.

La aglomeración de paradas en la zona izquierda concuerda con una estancia normal en el domicilio. El hecho de que haya varias posiciones diferentes se debe a la diferencia de posicionamiento dentro de las habitaciones de una misma casa y las posibles desviaciones causadas por el dispositivo GPS, puesto que en un principio la distancia mínima para considerar dos posiciones como diferentes era muy baja. Este problema fue la principal razón para incluir el parámetro de distancia mínima para detectar nuevas posiciones cercanas.

Distancia mínima detectada

Tras la inclusión de la funcionalidad de distancia mínima y su respectivo parámetro se procedió a realizar una prueba, visible en la Figura 18.

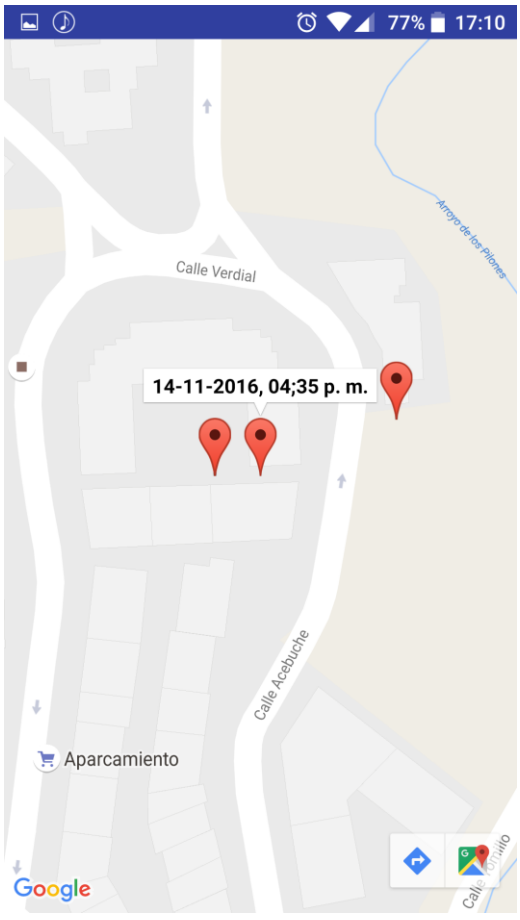

Figura 17 - Prueba 2

En esta prueba se estableció la distancia mínima de detección al mínimo de 1 metro. Tras ello, se solicitó almacenar una parada manualmente en dos habitaciones distintas de la casa. Como se puede observar, cada parada se registró como una posición diferente. Se procedió pues, a aumentar la distancia mínima de detección al

valor de 40 metros y se solicitó el almacenamiento de paradas por todo el domicilio. Sin embargo no se declararon posiciones nuevas hasta que se salió del domicilio y nos desplazamos al bloque de enfrente, en el cual se registró la posición que se puede ver en la parte derecha.

Habría sido interesante realizar más pruebas de diferente tipo, como por ejemplo un viaje largo en coche, pero debido al alto coste en tiempo de tales pruebas, se optaron por pruebas más sencillas y menos costosas.

Capítulo 6. Conclusiones y líneas futuras

6.1. Conclusiones

En las *Smartcities*, la optimización de recursos, el análisis de los datos y el tratamiento automático de los mismos cobra una gran importancia, más aún gracias a las nuevas tecnologías y dispositivos al alcance del usuario. Estas tecnologías se encuentran mayormente aún por explotar ya que poseen una gran potencia que bien utilizada puede servirnos para mejorar nuestra rutina y hábitos diarios. En este Trabajo Fin de Grado se ha propuesto aprovechar la tecnología que poseemos en nuestros dispositivos móviles para crear un sistema capaz de automatizar procesos que anteriormente debían ser realizados de forma manual. Estos procesos nos pueden ofrecer información valiosa, como las rutas que tomamos diariamente y las actividades que realizamos, junto con el horario.

La aplicación Android desarrollada permite al usuario mantener un historial de los lugares que visita de forma completamente autónoma, sin tener que preocuparse de tener que gestionar confirmaciones o introducir datos de forma manual, todo gracias al uso de las tecnologías disponibles en los dispositivos móviles que tan extendidos se encuentran, como la tecnología GPS, la tecnología WiFi y los acelerómetros y sensores de movimiento.

Hemos podido comprobar también el gran potencial de las aplicaciones desarrolladas para Android, ya que además de poseer un gran mercado en el mundo de los *smartphones*, se pueden llevar a cabo proyectos interesantes. La aplicación desarrollada se puede aplicar a diferentes ámbitos, como por ejemplo ayudar a personas que padezcan alzhéimer a mantener un historial sobre la ruta que realizan cada día o dónde estuvieron en un momento determinado.

Durante la etapa de investigación de este TFG, se han podido comparar diferentes herramientas y tecnologías entre sí, y aunque nos hemos decantado por unas en concreto, disponemos de información sobre las demás herramientas que pueden resultar útiles en otros proyectos de diferente ámbito.

6.2. Líneas futuras

La finalización de este Trabajo Fin de Grado deja abiertas muchas puertas para posibles ampliaciones de la aplicación o investigaciones futuras. Las más notables son las siguientes:

- **Automatización de detección de actividad más extensa:** Tal como se puede automatizar la detección de paradas, se podría ampliar el proyecto para incluir la automatización de diversas actividades, tales como andar o montar en bicicleta. Mediante el uso de los sensores de los dispositivos móviles y APIs como GooglePlayServices se podría lograr una ampliación bastante atractiva.
- **Mejorar la precisión GPS:** En algunas ocasiones el sistema GPS puede dar valores incorrectos, como posiciones ligeramente desviadas de la posición real. En un principio se ha tratado este tema con la distancia mínima de detección, pero se podrían investigar otros métodos de mejora de precisión tales como algoritmos de corrección basados en la toma de múltiples mediciones.
- **Exportar la aplicación a otros sistemas operativos móviles:** El sistema operativo Android se trata de uno de los más extendidos en el mercado de dispositivos móviles, pero también hay un gran número de terminales con otros sistemas operativos tales como IOS o Windows Mobile, por lo que si desarrollamos esta aplicación para los demás sistemas operativos tendremos una comunidad mayor de potenciales usuarios.
- **Añadir un servidor de datos:** Una vía interesante puede ser la de implementar un sistema de recolección de datos, los cuales pueden ser utilizados para elaborar estadísticas de gran interés, como las zonas más visitadas de una ciudad en momentos determinados del día.

Estas líneas de trabajo son solo un pequeño esbozo de la gran cantidad de oportunidades disponibles en el mundo de la tecnología móvil y su relación con las ciudades inteligentes y el tratamiento de datos.

Referencias

1. Aragón FN. External analysis of the smartphone industry in Spain. 2014..
2. Sanjeev Kumar AP. Role of Big Data and Analytics in Smart Cities. 2016..
3. Página oficial de UML. http://www.uml.org/.
4. Página oficial de GitHub. https://github.com/.
5. International Data Corporation (IDC). 2016 [cited 2016 Octubre 18. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.
6. Página oficial de SQLite. https://sqlite.org/.
7. Documentación Google Play Services. https://developers.google.com/android/guides/setup.
8. Documentación GoogleMaps. https://developers.google.com/maps/documentation/android-api/?hl=es-419.
9. Página oficial de Android Studio. https://developer.android.com/studio/index.html?hl=es-419.
10. Página oficial de Modelio. [Online]. Available from: https://www.modelio.org/.
11. Página oficial MySQL workbench. http://www.mysql.com/products/workbench/.

Apéndices

A. Manual de usuario

La aplicación se proporciona en el CD incluido junto a la memoria entregada. Sin embargo si se desea se puede descargar un APK de la aplicación utilizando el siguiente enlace:

https://drive.google.com/open?id=0B44rCJ2SNeBJVlZWYXI0MFpNMTQ

Una vez dispongamos del APK sólo habrá que copiar el mismo en el dispositivo móvil Android y ejecutarlo, lo que procederá a instalar la aplicación en el terminal. Es posible que sea necesario permitir la instalación de aplicaciones de origen desconocido. Dicha opción se encuentra generalmente en el apartado de seguridad del dispositivo.

Ya instalada la aplicación lo único que hay que hacer es pulsar en el icono para abrirla. Lo primero que ocurrirá, en terminales con versión de Android 6.0 o superior, será la solicitud de los permisos de GPS, los cuales deben ser aceptados para el correcto funcionamiento de la aplicación. Una vez hecho esto la aplicación estará lista para funcionar. En el caso de tener una versión de Android inferior a 6.0 no hará falta otorgar permisos a la aplicación, puesto que se realiza de forma automática al instalarla.

Feet3 comenzará a detectar paradas automáticamente sin necesidad de ninguna interacción por parte del usuario. Siempre que la aplicación se encuentre activa o en segundo plano, seguirá detectando paradas. Sin embargo, si la aplicación es terminada mediante el menú de finalización de aplicaciones, se detendrá la detección automática completamente.

Es importante comprobar que el servicio de Ubicación del dispositivo se encuentre activo, puesto que sin él no se podrán detectar posiciones geográficas, ya sea mediante el uso de GPS o de ubicación por redes.

La interfaz de la aplicación es sencilla de entender, en las Figuras 19 y 20 se encuentran marcados los elementos más importantes:

Figura 18 - Manual Main Activity

1. **Botón de preferencias:** Abre el menú de preferencias
2. **Lista de Historial de paradas:** Lista navegable en la que se muestran todas las paradas realizadas.
3. **Botón de guardar localización:** Al pulsar el botón se guardará inmediatamente una parada en el lugar en el que se encuentre el usuario.
4. **Botón de borrar historial:** Permite borrar todo el historial de paradas almacenado en la base de datos local del dispositivo.

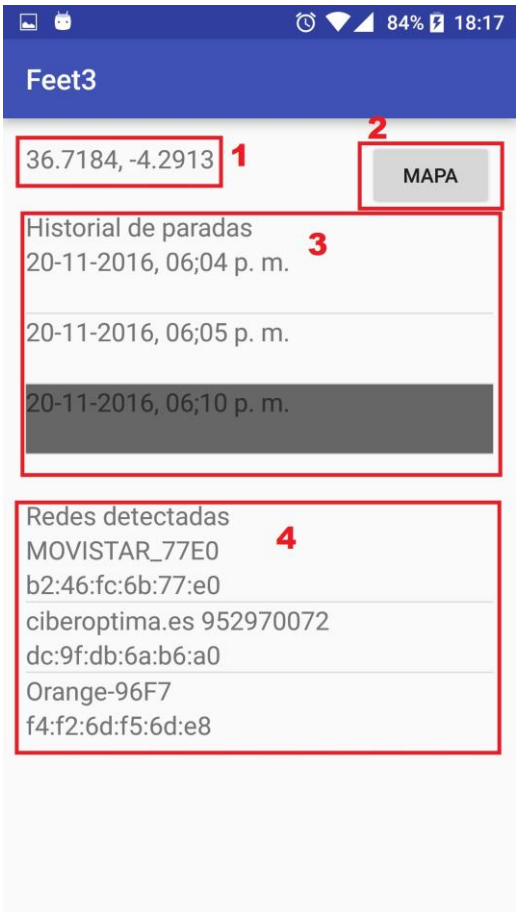

Figura 19 - Manual Información adicional

1. **Nombre de posición:** Nombre asignado a la posición en la que se encuentra la parada. En el caso de que no se le haya asignado ningún nombre se mostrará por defecto las coordenadas de la posición.
2. **Botón de mapa:** Botón que abre la vista gráfica de paradas mediante mapa.
3. **Lista de Historial de paradas:** Lista navegable en la que se muestran todas las fechas en las que se detectó una parada en dicha posición.
4. **Lista de Redes detectadas:** Lista navegable que muestra las redes detectadas en la fecha seleccionada. Si se pulsa sobre una red se abrirá el menú para editar su nombre.

Índice de figuras

Figura 1 - Casos de uso 7
Figura 2 - Modelo relacional de la base de datos 13
Figura 3 - Diagrama de clases 15
Figura 4 - Estructura de la aplicación Android 17
Figura 5 - Main Activity 20
Figura 6 - Historial de paradas 21
Figura 7 - Información adicional 23
Figura 8 - Información adicional remarcada 24
Figura 9 - Edición de nombre 25
Figura 10 - Visualización mediante mapa 26
Figura 11 - Mapa alejado 27
Figura 12 - Botón de preferencias 31
Figura 13 - Preferencias 32
Figura 14 - Tiempo de detección de parada 33
Figura 15 - Distancia de detección de parada 33
Figura 16 - Prueba 1 34
Figura 17 - Prueba 2 35
Figura 18 - Manual Main Activity 42
Figura 19 - Manual Información adicional 43

Índice de tablas

Tabla 1 - Requisitos funcionales..... 5
Tabla 2 - Requisitos no funcionales..... 6
Tabla 3: Caso de uso 1 8
Tabla 4 - Caso de uso 2 8
Tabla 5 - Caso de uso 3 9
Tabla 6 - Caso de uso 4 9
Tabla 7 - Caso de uso 5 10
Tabla 8 - Caso de uso 6 11
Tabla 9 - Caso de uso 7 11
Tabla 10 - Caso de uso 8 12