TRABAJO FINAL DE CARRERA

TÍTULO DEL TFC: Localizador de Objetos G-DROID

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Lidia Ibern Ortega

DIRECTOR: Dolor Royo Vallés

FECHA: 23 de Mayo de 2012

Titulo: Localizador de Objetos G-DROID

Autor: Lidia Ibern Ortega

Director: Dolors Royo Vallés

Data: 23 de Mayo de 2012

Resumen

En este proyecto se presenta el diseño y creación de G-DROID, una aplicación móvil para todos los dispositivos con plataforma Android, siempre y cuando dispongan de conexión a Internet y señal GPS.

La aplicación se basará en un sistema cliente-servidor que permitirá el acceso y escritura en una base de datos, el cliente tendrá acceso a la base de datos del servidor para poder utilizar los puntos de interés que otros usuarios hayan compartido en el servidor y así mismo compartir nuevos puntos.

Title: Object Locator G-DROID

Author: Lidia Ibern Ortega

Director: Dolors Royo Vallés

Date: May, 23th 2012

Overview

This project presents the design and creation of G-DROID, a mobile application for all Android devices, if they have internet and GPS signal.

The application is based on a client-server system that will allow access and write to a database, the client can access the database server to use the points that other users have shared on the server and share new points to.

Agradecimientos

Me gustaría dar las gracias a mis amigas/os por estar escuchando día sí y día también como llevaba el proyecto y agradecerle a Mireia Català, Patricia Anacleto y Ester González por su gran ayuda y motivación.

A mis padres por estar en todo momento tanto en los buenos como en los malos, que al ser tantas horas de dedicación han vivido a mi lado los más malos.

A toda mi familia, y sobre todo a vosotros dos, Salvador y Tomas, que sé que seguís estando a mi lado.

ÍNDICE

ÍNDICE FIGURAS Y TABLAS

FIGURAS

TABLAS

INTRODUCCIÓN

Este proyecto se basa en el desarrollo de una aplicación Android para dispositivos móviles que permita al usuario guardar en una base de datos la ubicación del dispositivo utilizando el posicionamiento GPS, pudiendo así compartirla en un servidor para que otros usuarios puedan ver la ubicación.

El objetivo del proyecto es aprender a diseñar una aplicación Android con el lenguaje de programación Java y conseguir la aplicación preparada para la instalación en los dispositivos móviles.

Una de las razones que me han llevado a realizar el proyecto ha sido mi interés por la programación. El hecho de que las aplicaciones móviles hayan crecido tanto en los últimos tiempos ha permitido que me motivara más tratar de desarrollar este proyecto, y así poder comprobar que el resultado final es una aplicación que puede estar al alcance de todos los usuarios que dispongan de un dispositivo móvil y que les sea de utilidad.

El objetivo personal planteado con este proyecto es capacitarme y prepararme para el desarrollo de aplicaciones Android en el futuro. De esta manera conseguiré la especialización necesaria para poder posicionarme en el mercado laboral dentro de la rama de programación.

CAPÍTULO 1. DISEÑO Y ARQUITECTURA DE LA APLICACIÓN

1.1. Localizador Objetos (G-DROID)

Crearemos una aplicación Android que sea capaz de guardar la localización GPS en la que el usuario, por algún motivo, necesite tenerla en la memoria del dispositivo móvil.

Por ejemplo, cuando una persona esté en una ciudad que no conoce ya no tendrá que preocuparse de la calle en la que ha dejado el coche. Simplemente con utilizar la aplicación y guardar la ubicación tendrá suficiente, ya que la aplicación le mostrará la ruta para volver al punto de interés.

La localización la podremos guardar tanto en local como en un servidor remoto, en el caso de que queramos compartir la ubicación para que otros usuarios puedan verla.

La aplicación se basa en un sistema Cliente-Servidor el cual se describe a continuación.

1.2. Funcionalidades del Terminal

El cliente será un Smartphone, donde instalaremos la aplicación creada.

La aplicación permite al usuario:

- Localización GPS

  El programa nos permitirá conocer nuestra ubicación, siempre y cuando tengamos señal GPS, visualizándola a través de un mapa que nos proporciona *Google Maps.*
  Si no encuentra señal GPS se utilizarán las redes móviles o Wi-fi, por defecto, para obtener la ubicación.

- Guardar Posición

  Permite al usuario guardar la posición en la que se encuentra, con la opción de quererla compartir en el servidor añadiendo, si lo desea, la descripción del punto y número de teléfono de contacto.
  Todos los puntos se guardarán en una base de datos SQLite del propio dispositivo móvil.

- Buscar Punto

  Tendremos la opción de Buscar el punto que hayamos guardado en nuestra base de datos, o que hayan compartido otros usuarios.

- Mostrar mapa o ruta

  Cuando seleccionemos el punto que estábamos buscando, tendremos la posibilidad de solicitar la localización de éste como la ruta al punto desde la posición en la que nos encontremos.

- Otras opciones

  La aplicación también nos permitirá solicitar la información del punto, renombrarlo, borrarlo, compartirlo y nos mostrará toda la información del punto en caso de que la solicitemos.

1.3. Funcionalidades externas

El cliente se conectará al Servidor MySQL que utilizaremos como base de datos para guardar todos los puntos de los clientes que quieran compartirlos.

Al conectarnos al Servidor podremos:

- Actualizar BD

  Actualizar nuestra lista de puntos, el servidor nos informará si se han añadido nuevos puntos o borrado de la base de datos del servidor.

- Otras opciones

  Si nuestro punto está compartido, cuando deseemos borrarlo o renombrarlo tendremos que solicitar al servidor esta acción para que el punto se actualice para los demás usuarios.

La aplicación, por lo tanto, utiliza servicios de Internet y posicionamiento GPS. Internet lo utilizaremos para acceder a la base de datos del Servidor con la información de todos los puntos compartidos por los usuarios y a los mapas suministrados por *Google Maps*.

Con el servicio de GPS obtendremos nuestra ubicación actual en el mapa, para poder guardar la ubicación o mostrarnos la ruta al destino seleccionado.

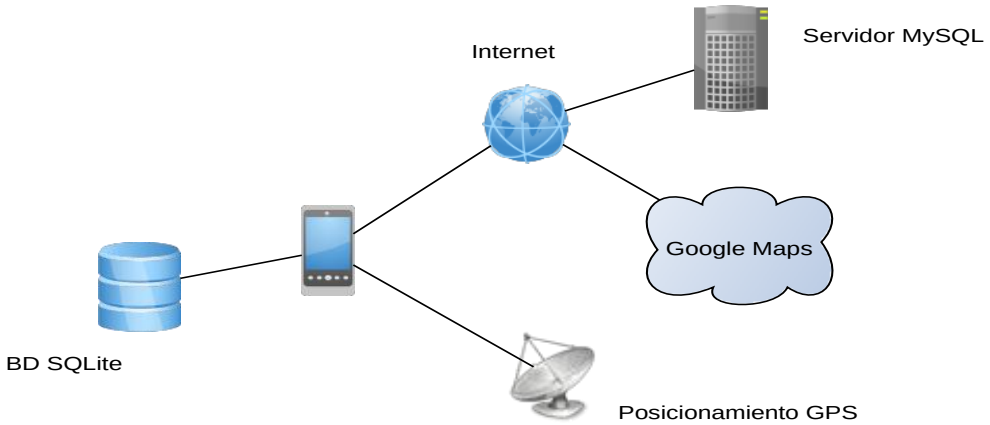

Fig. 1.1. Esquema general del Sistema

1.4. Modelo Cliente Servidor

Como se observa en la figura anterior en el sistema intervienen dos partes el cliente y el servidor.

El servidor es una máquina física conectada a la red en la que se aloja la base de datos.

En la base de datos se almacena toda la información de los puntos guardados en el servidor, la herramienta utilizada para implementar la base de datos es MySQL, un sistema gestor de bases de datos.

El cliente, tendrá instalada la aplicación que le permitirá guardar los puntos y gestionar toda la información guardada, al mismo tiempo podrá interactuar con el servidor en caso necesario.

La base de datos del dispositivo móvil será implementada con SQLite, proporcionada por el Sistema Operativo Android.

La comunicación entre cliente y servidor se realiza con el protocolo TCP.

CAPÍTULO 2. TEGNOLOGÍAS UTILIZADAS

En éste apartado se explican las tecnologías utilizadas en la aplicación.

2.1. Android

Android es un sistema operativo diseñado para ser utilizado en dispositivos móviles basado en Linux, por lo que todos los servicios base se rigen por un sistema operativo de código abierto.

Características más importantes del S.O:

- Framework de la aplicación: la cual nos permite la reutilización de componentes para diferentes aplicaciones.
- Máquina virtual Dalvik: máquina virtual Java optimizada para los dispositivos móviles
- Navegador Web integrado: el motor de navegación es el WebKit, de código abierto y utilizado como base para varias aplicaciones.
- Gráficos
- SQLite: para el almacenamiento de datos estructurados
- Soporte audiovisual: soporta la mayoría de formatos de vídeo, audio e imágenes.

Dependiendo del Terminal:

- Telefonía GSM.
- Bluetooth, EDGE, 3G y Wifi.
- Cámara, GPS, brújula y acelerómetro.

El entorno de desarrollo utilizado es Eclipse usando el plugin de herramientas de desarrollo de Android.

Las aplicaciones se desarrollan habitualmente en lenguaje Java con Android Software Development Kit (Android SDK), por lo que para poder desarrollar una aplicación se necesita un conocimiento aceptable de programación en Java y el SDK.

2.2. SQLite y MySQL

Tanto SQLite y MySQL serán utilizados para el almacenamiento de datos en el terminal y el servidor respectivamente.

- SQLite

  Android proporciona las librerías de SQLite para la creación y gestión de bases de datos.

  El motor SQLite no es un proceso independiente con el que el programa se comunica, en lugar de eso, pasa a ser parte del propio programa. Esto conlleva a que las llamadas sean más eficientes y el tiempo de respuesta sea menor que en una solicitud de cliente - servidor.

- MySQL

  MySQL es un sistema de administración relacional de bases de datos, archiva datos en tablas separadas en vez de colocar todos los datos en un archivo.

  MySQL permite su uso de forma libre.

2.3. GPS i Wifi

Se utiliza la señal GPS para poder determinar la posición del terminal, en el caso de que la señal GPS no se esté recibiendo, utilizará las redes móviles y Wifi para saber la posición actual.

Android proporciona el servicio de localización para poder obtener la posición en la que se encuentra el usuario, utilizando los diferentes proveedores mencionados anteriormente.

CAPÍTULO 3. INTRODUCCIÓN A LA PROGRAMACIÓN EN ANDROID

3.1. Componentes de una aplicación Android

Android proporciona diferentes componentes para poder desarrollar la aplicación según el objetivo de ésta.

No todas las aplicaciones tienen que tener todos los componentes pero cualquier aplicación será una combinación de éstos.

- Activity

  Es el componente más usado en las aplicaciones Android. Una *actividad* representa una pantalla individual en el Terminal y presenta una interfaz gráfica al usuario.
  La navegación entre las pantallas se realiza iniciando nuevas actividades. Cuando iniciamos una nueva actividad, la anterior se queda en pausa hasta que vuelve a ser invocada.
  El componente se implementa mediante la clase Activity.

- Intent

  Clase que se utiliza para moverse de una pantalla a otra. Un *Intent* describe lo que una aplicación desea hacer.

- Broadcast Intent Receiver

  Utilizado para lanzar una ejecución dentro de la aplicación cuando un determinado evento se produce.
  Este componente no tiene interfaz de usuario pero puede utilizar la API de *Notification Manager* para avisar al usuario, a través de la barra de estado, de que se ha producido un evento.

- Service

  Un servicio no tiene interfaz gráfica, pero puede ejecutarse en "background" por un tiempo indefinido. Por ejemplo, podemos utilizar un servicio que vaya capturando la posición GPS y nos avise cuando estemos cerca de un punto de interés.

- Content Provider

  Con este componente, cualquier aplicación puede almacenar datos (en un fichero, en base de datos SQLite o cualquier otro formato), y que éstos sean compartidos entre distintas aplicaciones.

3.2. Cambiar de Activity

En Android, cada pantalla que tenga la aplicación es una *activity* y para poder pasar de una a otra utilizaremos los *Intents.*

Para iniciar una nueva actividad podemos utilizar estos dos métodos:

- startActivity(), se le pasa el *Intent* con los datos necesarios.
- startActivityForResult(), iniciamos la actividad esperando un resultado de ella, se le pasa el *Intent* y un código para identificar el resultado posteriormente.

El método getIntent() se utilizará para gestionar el Intent que ha llamado a la actividad.

Para recibir el resultado de una actividad, tendremos que implementar el método onActivityResult() en la actividad que hemos llamado startActivityForResult().

También podemos pasar información entre *Activities*, crearemos un objeto Bundle donde introduciremos los datos, y otro en la *Activity* que los recoja.

En el caso de que queramos devolver un valor al finalizar la *Activity*, se utiliza el método setResult((RESULT_OK o RESULT_CANCELED), intent).

En el anexo podemos ver la creación de objetos *intents* para comunicarnos con las demás *activities* de la aplicación y objetos *Bundle* para pasar datos entre ellas. También se introduce el código de la notificación tipo Toast, con la que se muestra por pantalla un mensaje al usuario.

3.3. Android Manifest

El archivo *AndroidManifest.xml* está presente en todas las aplicaciones Android.

Su contenido especifica todos los componentes de la aplicación, la configuración global de la misma (librerías que necesita la aplicación y permisos que necesita para funcionar).

Al generar un nuevo proyecto, el archivo es creado automáticamente, pero en cuanto creemos más actividades o utilicemos diferentes servicios, tendremos que declararlo en el fichero *Manifest*.

En el anexo podemos ver un ejemplo del archivo Manifest.xml

CAPÍTULO 4. REQUISITOS NECESARIOS PARA PROGRAMAR APLICACIONES ANDROID

4.1. Eclipse

Para desarrollar la aplicación Android se ha utilizado el entorno de programación Eclipse.

En este caso se ha instalado Eclipse 3.7.

4.2. SDK de Android

Para poder comenzar a desarrollar la aplicación, tenemos que descargarnos la versión correspondiente del SDK (Sotware Development Kit) de Android, teniendo en cuenta el S.O. en el que se está trabajando.

4.3. ADT (Android Developement Tools)

Google pone a disposición el plug-in para Eclipse *Android Development Tools* (ADT) que facilita el desarrollo de aplicaciones.

Al instalarlo y crear un proyecto comprobamos que se genera:

- Directorio /src/ donde está todo el código fuente de la aplicación. Eclipse creará el código de la pantalla principal de la aplicación.

- Directorio /res/ contiene todos los ficheros de recursos necesarios para el proyecto (imágenes, videos, etc...), las carpetas a destacar son:

    - ✓ /res/drawable/. Las imágenes de la aplicación.
    - ✓ /res/layout/. Ficheros de definición de la interfaz gráfica.
    - ✓ /res/menú/. Definición de los menús de la aplicación

- Directorio /gen/ contiene elementos de código generados automáticamente al compilar el proyecto.

    - ✓ Fichero R.java y la clase R, es el más importante. La clase R tendrá unas constantes con sus respectivos ID de todos los recursos de la aplicación incluidos en la carpeta /res/ para poder acceder a ellos fácilmente desde nuestro código.

- Directorio /assets/. Contiene todos los demás ficheros auxiliares para la aplicación.

- Fichero AndroidManifest.xml
-

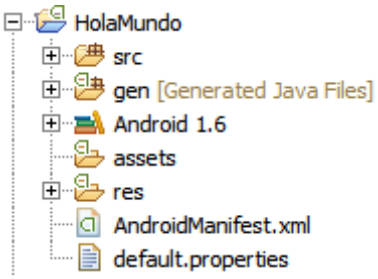

Fig. 4.1. Elementos generados inicialmente

En el anexo podemos ver un ejemplo del archivo R.java.

4.4. Configuración ADT y Emulador

- Configuración ADT

  En la ventana de configuración de Eclipse – Android indicaremos la ruta donde se ha instalado el SDK.

- SDK Targets

  Para poder desarrollar las aplicaciones en versiones concretas de Android, tendremos que descargar los targets correspondientes a éstas.

  Podemos hacerlo desde el AVD Manager en la sección *Avaiable Packages.*

- Emulador

  Las aplicaciones podemos probarlas y depurarlas en el emulador o AVD (Android Virtual Device). Desde el AVD Manager podremos crear los AVD que necesitemos, configurándolos con las versiones y características específicas para nuestra aplicación de Android.

CAPÍTULO 5. IMPLEMENTACIÓN DE LA APLICACIÓN

A continuación explicaremos las partes importantes que hemos implementado para que la aplicación realice todas las funcionalidades descritas en el apartado anterior.

5.1. Interfaz de usuario

Para poder diseñar todas las pantallas de nuestra aplicación existirán dos ficheros distintos: por una parte, el diseño visual de las pantallas que se define como un fichero XML y por otro lado, el código java que determina la lógica de las pantallas.

- /res/layout/nombre_pantalla.xml
- /src/paquetejava/nombre_clase.java

5.1.1. Diseño visual

El fichero XML define todos los elementos visuales que componen las pantallas, especificando así todas sus propiedades.

Lo primero que definimos son los *layout*, elementos no visibles que determinan la distribución, posición y dimensiones de los controles que se incluyan.

Android nos proporciona los siguientes *layouts*:

- FrameLayout

  Coloca todos los controles alineados en la esquina superior izquierda. Los controles quedarán ocultos por el siguiente, a no ser que el último tenga transparencia.

- LinearLayout

  Coloca todos los elementos de forma horizontal o vertical según se establezca la propiedad de *android:orientation*.

- TableLayout

  Distribuye los elementos definiendo las filas, columnas y la posición del componente dentro de la tabla.

La estructura la definimos indicando las filas <*TableRow*> y el total de columnas de la tabla se determinará con los componentes que contenga la fila.

- RelativeLayout

  Permite especificar la posición de cada elemento de forma relativa a otro elemento incluido en el *layout.*

En la aplicación, el *layout* más utilizado ha sido el *RelativeLayout* ya que podemos poner todos nuestros controles tal y como lo deseemos. El *LinearLayout* también se ha utilizado para las *Layouts* que contienen controles del tipo *MapView*.

Todos los componentes podrán establecer sus propiedades *android:layout_width* y *android:layout_height* que tendrán los valores:

- *fill_parent*: el control tendrá la dimensión del *Layout* que lo contiene.
- *wrap_content*: el control tendrá la dimensión de su contenido.

A continuación se pueden incluir tantos controles como necesitemos para la interacción con el usuario. Los utilizados en la aplicación son los siguientes:

- Button: botón básico, definimos el texto del botón con la propiedad *android:text*, también se puede definir el estilo, color, tamaño de la fuente, etc .

- ImageButton: definimos una imagen a mostrar en vez de texto con la propiedad *android:src*, asignándole alguna imagen que tengamos en la carpeta */res/drawable*.

- TextView: Se utiliza para mostrar un determinado texto al usuario. Igual que en el caso del botón, podemos definir las propiedades del texto.

- EditText: Se utiliza para la introducción de texto por parte del usuario.

- CheckBox: Utilizado para marcar y desmarcar opciones de la aplicación.

- RadioButton: Tiene la misma propiedad que el checkbox pero perteneciendo a un grupo, sólo se podrá marcar una opción y tiene que estar marcada obligatoriamente.

- ListView: muestra al usuario una lista de opciones seleccionables.

- MapView: Utilizado para incluir un mapa de *Google Maps*.

Todos los controles tendrán la propiedad *android:id* para poder identificarlo en el código de la aplicación. El id se generará automáticamente como constante en la clase *R*.

5.1.2. Implementación

Cada pantalla se definirá en ficheros java independientes.
Las clases java que implementen la lógica de las pantallas extenderán de la clase *Activity*. De este modo sobrescribiremos el método *OnCreate()* para llamar a la Actividad y establecer como interfaz gráfica la que hayamos definido.

Cada elemento XML tendrá su propia clase java, con la que crearemos objetos haciendo referencia a los diferentes controles de la interfaz que vayamos a manipular indicando el ID de cada control.

Una vez ya podamos manipular los diferentes controles, tendremos que implementarlos.

En el anexo podemos ver ejemplos de implementación de los diferentes controles que hemos utilizado.

Una vez acabada la parte lógica y visual de la aplicación se han de colocar todas las actividades que creemos en el fichero *AndroidManifest.xml*, exceptuando la principal, ya que se ha generado por defecto al crear el nuevo proyecto.

5.2. Localización Geográfica

Con la localización geográfica, podremos saber cuál es nuestra ubicación y así utilizar las diferentes opciones que nos da la aplicación.

La ubicación se determina a través de la señal GPS. En caso de que no obtenga la señal utilizará las redes móviles o Wi-fi, determinando así la ubicación e intentando obtener la señal GPS.

Dependiendo de la *activity* en la que nos encontremos (guardar, mostrar mapa o mostrar ruta), al pulsar el botón menú podremos escoger las siguientes opciones:

- Guardar
- Ir hasta el punto
- Vista mapa o satélite
- Información del tráfico
- Posición Actual

5.2.1. Mapas

Android proporciona el servicio de localización, obteniendo la latitud y longitud de la posición en la que se encuentra el usuario.

Se utilizará la librería que proporciona *Google* para tener acceso a los mapas controles de *Google Maps,* por lo que la clase java que implementa esta pantalla extiende de la clase *MapActivity* utilizando el método *isRoutedDisplayed(),* que deberá devolver *true* en caso de que se muestre información de ruta sobre el mapa.

Para mostrar el mapa en el layout, añadiremos el control *MapView* (encargado de descargar y procesar el mapa para visualizarlo) teniendo en cuenta la propiedad *android:apiKey,* donde indicaremos la clave de uso de *Google Maps (Map API Key). E*n la bibliografía se ha adjuntado un link donde se especifican todos los pasos a seguir para la obtención de dicha clave.

Para poder ejecutar la aplicación, tendremos que modificar el fichero *AndroidManifest.xml* en el que se especificará la utilización de la librería de *Google Maps*, y se habilitarán los permisos para acceder a Internet y obtener la ubicación del dispositivo.

```
<uses-library android:name="com.google.android.maps" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

- **Localización**

Para la localización, utilizaremos la clase *LocationManager*, con la que se activa el proveedor de localización (GPS). Para poder obtener la posición actual se han de implementar los siguientes métodos y objetos:

- ✓ *getLastKnownLocation(String provider),* método que devuelve la última posición conocida.

- ✓ *LocationListener*, objeto implementado para definir las acciones que se han de realizar cuando se reciban las actualizaciones. Los diferentes métodos son:

  - onLocationChanged(location), cada vez que se recibe una actualización de la posición.
  - onProviderDisabled(provider), el proveedor se deshabilita.
  - onProviderEnabled(provider), el proveedor se habilita.
  - onStatusChanged(provider, status, extras), el proveedor cambia su estado (OUT_OF_SERVICE, TEMPORARILY_UNAVAILABLE y AVAILABLE).

- ✓ *requestLocationUpdates()* método al cual nos subscribimos para recibir notificaciones de cambio de la posición. Le pasaremos 4

parámetros (Nombre del proveedor, tiempo actualizaciones (ms), distancia entre actualizaciones (m), *LocationListener*).

- **Control MapView**

A continuación se especifican los diferentes métodos que se han utilizado para poder manipular el mapa:

- ✓ setSatellite(true) cambio a vista satellite.
- ✓ setTraffic(true) muestra la información del tráfico.
- ✓ setBuiltInZoomControls(true) muestra los controles de zoom sobre el mapa.
- ✓ getController() utilizado para acceder al controlador del mapa, para poder modificar los datos de localización.
- ✓ animateTo(GeoPoint) desplaza el mapa hasta un punto determinado de forma progresiva indicando el centro del mapa con el objeto GeoPoint que tendrá los valores de latitud y longitud en microgrados (*1E6).
- ✓ setZoom(n) se indica el nivel de zoom deseado, siendo n un valor entero entre 1 y 21.

- **Overlays (Capas)**

Las capas las necesitamos para poder dibujar sobre el mapa, por ejemplo, las marcas de posición o dibujar la ruta para llegar al destino.
Para poder definir una capa se ha de crear una clase java que extienda de la clase *Overlay* y sobrescribir el método *draw()*, donde dibujaremos todo lo que queramos incluir en el mapa.
Cuando utilicemos la clase Overlay, además de pasarle los datos de localización para dibujar en la ubicación determinada, también le pasaremos mediante un indicador qué es lo que queremos que dibuje:

- ✓ Modo 0: Localización actual GPS (Menu_guardar)
- ✓ Modo 1: Origen Ruta
- ✓ Modo 2: Trazo ruta
- ✓ Modo 3: Destino Ruta
- ✓ Modo 4: Punto que Buscamos (Menu_buscar-Mostrar Mapa)

5.3. Obtención de la ruta

Para la obtención de la ruta de punto a punto *Google* proporcionaba el package *DrivingDirection*, pero éste fue eliminado a partir de la versión SDK de *Android 1.1*, por lo que la obtención de la ruta será a través de una petición http a *Google maps* en la que la *URL* que se envía contendrá el punto de origen y destino, el formato del archivo y el modo en el que queremos que nos muestre la ruta, a pie o caminando.

El archivo lo solicitaremos en KML, ya que está basado en XML para representar los datos geográficos, mostrando la latitud, longitud y altitud de la ruta solicitada.

Una vez realizada la petición, el programa utiliza el *parser DOM* para el tratamiento de los datos recibidos en un *Stream*.

Con *DOM* el documento *XML* se lee completamente, guardándose en un objeto *Document*.

El parser devuelve todo el contenido en forma de estructura de árbol, donde los elementos *XML* se representan en forma de nodos y su jerarquía padre-hijo se establece con la relación entre dichos nodos.

El motivo de la elección de *DOM* es que nos permite localizar y tratar un elemento concreto del documento *XML* sin tener que recorrer todo su contenido de principio a fin como en *SAX* el cual se lee secuencialmente.

En éste caso buscaremos la etiqueta <*GeometryCollection*>*,* siendo sus hijos todos los datos necesarios para la obtención de la ruta.

A continuación mostramos un ejemplo de petición *http* enviada por el usuario a *Google Maps* para la obtención de la ruta en *KML*.

http://maps.google.com/maps?f=d&hl=en&saddr=41.3600964, 2.1024115&daddr=41.3605321, 2.1087186&ie=UTF8&0&om=0&output=kml

5.4. Guardar y Buscar punto

La aplicación permite al usuario guardar la ubicación en la que se encuentra.

Cuando se seleccione la opción de guardar aparecerá una pantalla en la que pondremos el nombre del punto, la descripción y el número de teléfono de contacto. También tendremos la opción de poder compartir el punto en el servidor si se desea.

Una vez que se hayan introducido todos los datos, le daremos al botón OK para poder guardar el punto en la base de datos local, antes de guardar el punto se comprobará que el nombre no existe, en el caso de que exista informará al usuario para que éste cambie el nombre del punto.

En el caso de que el usuario quiera compartir la ubicación se realizará el mismo proceso que en local, ya que el punto también será guardado en la base de datos del Terminal.

Una vez guardado en local, se enviará al servidor para que sea guardado. Este proceso será explicado más adelante.

Para poder almacenar los datos y consultarlos, Android proporciona la base de datos SQLite sin tener que solicitar la información a través de Internet, por lo que será guardada localmente.

5.4.1. Base de datos SQLite

En primer lugar se ha de crear la base de datos a través de una clase que extienda de SQLiteOpenHelper.

SQLiteOnpenHelper tiene un constructor y dos métodos, onCreate() y onUpgrade(), utilizados para crear y actualizar la base de datos, éstos serán sobrescritos para adaptarlos a la aplicación. OnCreate() será ejecutado automáticamente en caso de que la BD no exista.

Para la comunicación con la base de datos utilizaremos sentencias SQL y éstas las ejecutaremos con el método execSQL().

Para poder ver la BD con todos los datos introducidos se puede utilizar cualquier administrador SQLite.

La base de datos local será guardada como *BDpuntos* y la tabla que contendrá los puntos *Puntos* tendrá la siguiente estructura:

Tabla 5.1. Tabla puntos de la BD Local

id_punto	latitud	longitud	desc_punto	tel_punto	usuario_comp	compartido

Donde:

- ✓ id_punto: nombre del punto.
- ✓ latitud y longitud: coordenadas GPS.
- ✓ Desc_punto. Descripción del punto.
- ✓ tel_punto: teléfono de contacto.
- ✓ usuario_comp: usuario que comparte la ubicación, en el caso de que no haya sido compartida el campo estará vacio.
- ✓ compartido: especifica si el punto está compartido (SI o NO).

Con SQLite a parte de crear la base de datos podremos insertar, actualizar, eliminar y consultar los puntos.

En el anexo se pueden ver las diferentes sentencias SQL utilizadas para realizar las acciones anteriormente mencionadas.

La opción Buscar de la aplicación nos muestra un listado con todos los puntos guardados en la BD, dicho listado será representado con el control ListView.

Con cada punto tendremos las siguientes opciones:

- Mostrar detalle del punto
- Compartir punto
- Mostrar Mapa
- Mostrar Ruta
- Renombrar Punto
- Borrar Punto

Utilizaremos un menú contextual para que aparezcan las opciones mencionadas cuando el usuario pulse sobre el punto.

Las opciones Mostrar detalle, Renombrar y Borrar las realizaremos con la comunicación de la BD local o la BD del Servidor, dependiendo de si el punto esta compartido o no. En el caso de que el punto no haya sido compartido por el propio usuario, Borrar y Renombrar estarán deshabilitados.

Las opciones Mostrar mapa y Mostrar ruta mostrarán las actividades pertenecientes a cada función.

La opción Compartir punto estará deshabilitada en caso de que ya lo esté, en caso de que se quiera compartir, la aplicación se conectará al servidor.

5.5. Conexión con el Servidor

La conexión con el servidor la haremos con sockets TCP, utilizando las clases ServerSocket y Socket. El servidor dará soporte a todos lo clientes que quieran conectarse a él, mediante la utilización de threads, éstos serán creados cada vez que el servidor acepte la conexión del cliente.

Tendremos cuatro clases para poder implementarlo,

- Servidor_localizador: es la clase principal, estará escuchando siempre por el puerto para poder aceptar todas las peticiones de conexión.

- Servidor_thread: la clase extiende de la clase thread y será llamada desde la clase principal para aceptar la conexión, de ésta manera el servidor sigue escuchando por el mismo puerto.

- Protocolo_servidor: en esta clase implementaremos todo el protocolo del servidor.

- MySQL: Implementará todas las funciones que nos pida el cliente en la base de datos. Creará las tablas en caso de que no existan y añadirá o modificará datos en ellas.

En el servidor instalaremos MySQL que gestionará todas las bases de datos que creemos y utilicemos en nuestro servidor.

5.5.1. Servidor MySQL

Para poder utilizar la base de datos MySQL la instalaremos en el servidor y descargaremos el driver para poder conectarnos a la base de datos.

La comunicación con la base de datos será igual que con SQLite a través sentencias SQL, para poder enviarlos utilizaremos la clase statement.

Statement tiene dos métodos executeUpdate() y executeQuery(), para modificar la base de datos y realizar consultas respectivamente.

La base de datos será creada con el nombre LOCALIZADOR y tendrá dos tablas:

- Usuarios

  Esta tabla contendrá los todos los usuarios de la aplicación.

  En caso de que el usuario quiera registrarse se añadirá a la tabla y si quiere acceder a la base de datos, el servidor le dará permiso en caso de que usuario y contraseña sean correctos.

  Tabla 5.2. Tabla usuarios BD servidor

Usuario	Contrasena

- Puntos

  Esta tabla tendrá la misma estructura que la tabla de la base de datos del cliente.

  Tabla 5.3. Tabla puntos BD servidor

id_punto	latitud	longitud	desc_punto	tel_punto	usuario_comp	compartido

  Las sentencias SQL utilizadas para realizar las acciones anteriormente mencionadas, son las mismas que en SQLite.

5.5.2. Funciones del servidor

El servidor nos permite hacer las siguientes funciones:

- Registrarse como usuario

  En el caso que el nombre de usuario se repita el registro no se llevará a cabo e informará al usuario para que éste cambie su identificación.

- Acceder a la base de datos

  Si el nombre de usuario o contraseña son erróneos, informará al usuario.

- Compartir

  Si se quiere compartir el punto, primero se recorrerá la tabla para que el punto no tenga el mismo nombre que otro compartido anteriormente, en el caso de que esto suceda informaremos al usuario para que cambie el nombre del punto.

  Si se guarda correctamente la aplicación guardará el punto en su base de datos local.

- Borrar o renombrar punto.

  Se corroborará que el usuario que solicita la acción sea el mismo que lo ha añadido en la tabla, en caso contrario informará al usuario.

  Si los cambios son realizados correctamente enviaremos una notificación para que el cliente haga los cambios en su base de datos local y así se corresponda con la del servidor.

- Actualizar base de datos

  Desde el cliente tendremos la opción de actualizar nuestra base de datos.

  El servidor recorrerá toda la tabla Puntos, buscando los puntos que no haya compartido el propio usuario y enviándole un array con los identificadores (siendo el mismo nombre con el que se ha guardado en la BD), desde la aplicación se compararán los identificadores y en el caso de que le falte algún punto solicitará al servidor que le envié toda la información para poder actualizarlo.

  Si el punto ya no existe lo eliminará de su base de datos.

En el siguiente diagrama se puede observar la comunicación Cliente – Servidor del sistema en el que el cliente solicita una actualización de la base de datos.

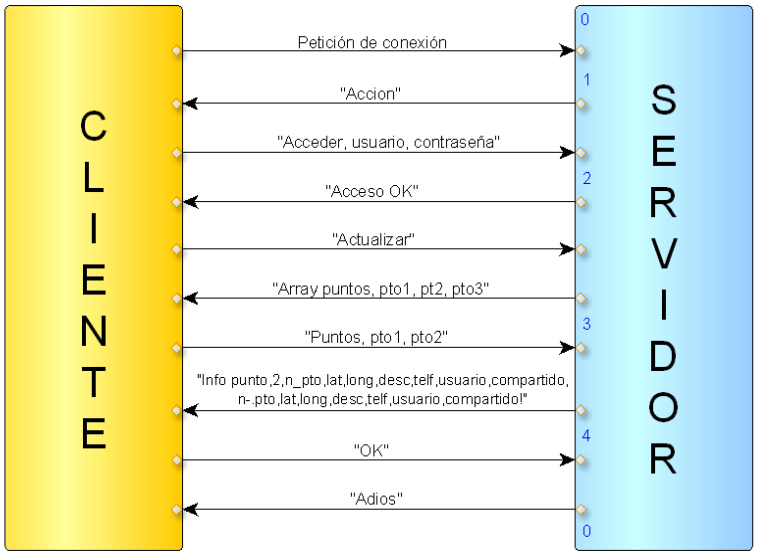

Fig. 5.1. Dialogo entre cliente y servidor

Podemos observar en el dialogo que tenemos diferentes estados en el servidor:

- Estado 0: Espera conexión
- Estado 1: Solicitud de acceso o registro
- Estado 2: Solicitud de acción, borrar, compartir o actualizar
- Estado 3: Actualización
- Estado 4: Salir

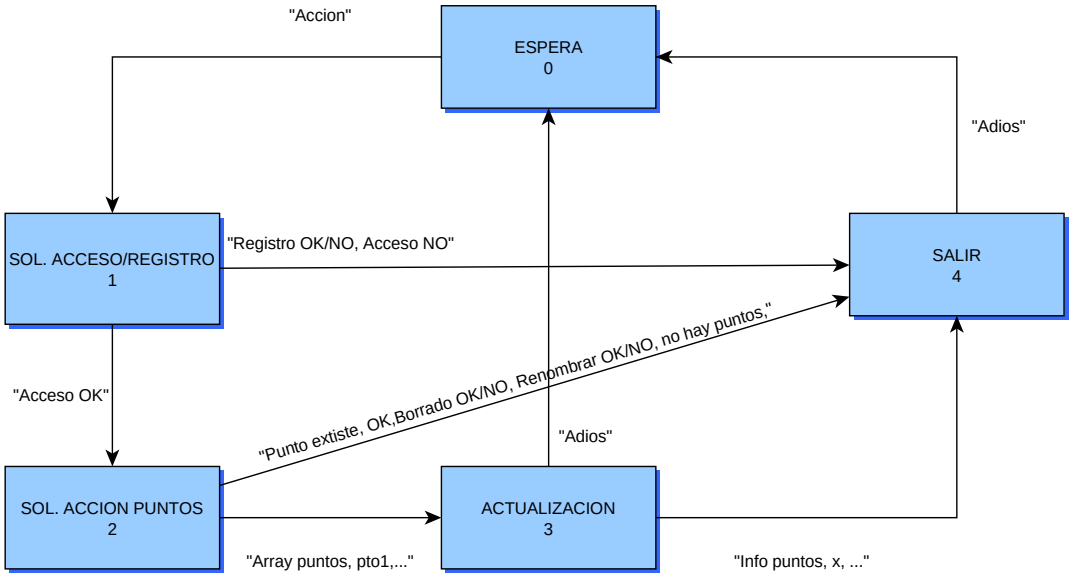

Fig. 5.2. Diagrama de estados del Servidor

CAPÍTULO 6. GUÍA DE USO DE LA APLICACIÓN G-DROID

Cuando se abra la pantalla principal de la aplicación, si la señal GPS está inhabilitada saldrá un mensaje de información para mostrar la pantalla de configuración de ubicación del Terminal para activar la recepción de la señal GPS, ya que es necesaria para la aplicación.

Fig. 6.1. Diálogo GPS inhabilitado

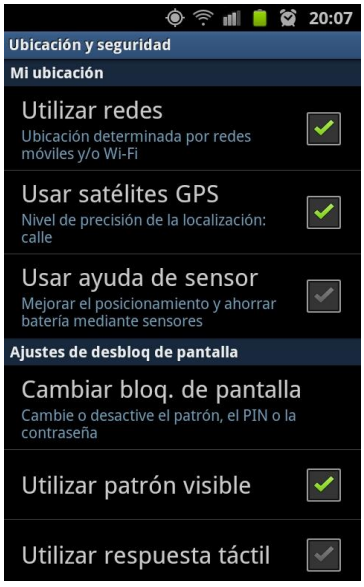

Fig. 6.2. Pantalla configuración

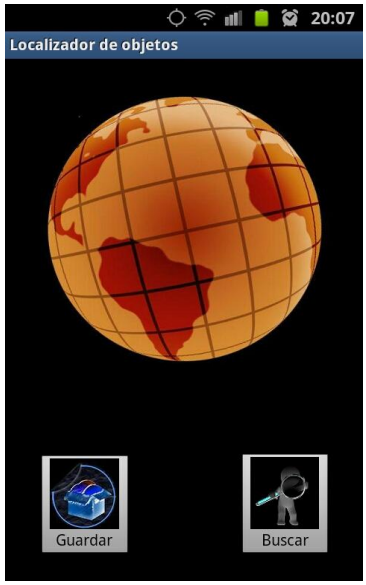

Fig. 6.3. Pantalla principal

G-DROID tiene dos funcionalidades básicas: guardar punto y buscar punto.

6.1. Guardar punto

Para poder acceder a esta funcionalidad, se ha de hacer click en el botón Guardar del menú principal. La señal GPS tiene que estar activada. En caso de que no encuentre la señal obtendrá la localización a través de la red Wifi, que también tendrá que estar activada.

Cumpliendo los requisitos anteriormente mencionados, nos aparecerá una pantalla con un mapa de Google Maps donde aparecerá la ubicación actual, el cual nos permitirá desplazarnos en el mapa y utilizar los controles de zoom.

Para poder ver todas las opciones que nos permite esta pantalla se ha de hacer click en el botón menú del Terminal:

- Guardar Posición: muestra la pantalla para guardar la posición actual.
- Mi posición: desplaza el mapa a la ubicación actual del usuario.
- Configuración Vista: nos permite elegir entre vista satélite y vista mapa.
- Menú: nos permite volver a la pantalla principal.

Si escogemos la opción de Guardar nos mostrará un formulario donde tendremos que introducir los campos: ID del punto, Descripción y teléfono de contacto.

El usuario podrá elegir si desea que la ubicación sea compartida en el servidor, en caso de que quiera compartirla, el usuario tendrá la opción de acceder o registrarse en el servidor (en caso de que no se haya registrado anteriormente).

Esta pantalla nos permite:

- Cancelar: cancela la acción de guardar y vuelve a la pantalla anterior.
- OK: guarda el punto, una vez guardado la aplicación volverá de nuevo a la pantalla principal.
- Menú: volver a la pantalla principal.

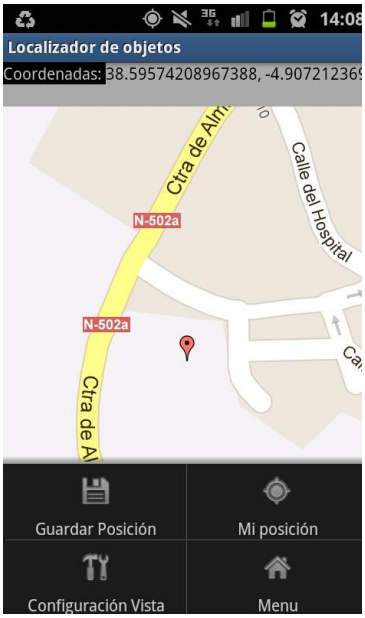

Fig. 6.4. Pantalla con la ubicación actual

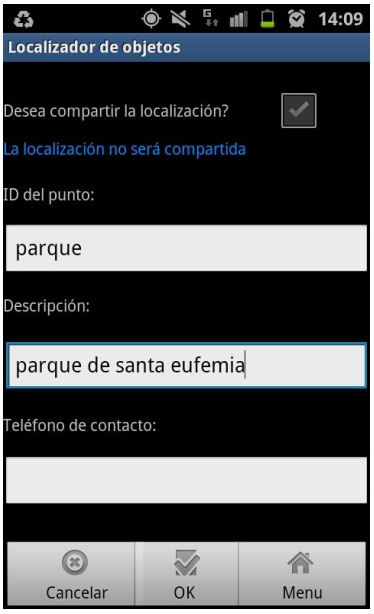

Fig. 6.5. Pantalla guardar punto

6.2. Buscar punto

Para acceder a esta funcionalidad, se ha de hacer click en el botón Buscar de la pantalla principal. Al darle al botón nos mostrará un listado con todos los puntos guardados en el Terminal y en el servidor, en el caso de que anteriormente el usuario haya actualizado la base de datos.

Cada punto mostrará el nombre del punto, la descripción, y en caso de que el punto sea compartido por otro usuario, el nombre de quién lo ha compartido.

Esta pantalla nos permite:

- Actualizar: actualiza la base de datos con el servidor, solicitando usuario y contraseña para poder acceder a éste o registrarse en caso de que no lo esté. Una vez la aplicación compare la base de datos del Terminal con la del servidor informará al usuario de los cambios que se hayan producido.
- Menú: volver a la pantalla principal.

Si se pulsa cualquier punto de la lista, se abrirá un menú contextual con las siguientes opciones:

- Mostrar detalles del punto
- Compartir punto
- Mostrar mapa
- Mostrar ruta
- Renombrar punto
- Borrar punto
- Salir

Las opciones no siempre pueden estar habilitadas, depende de los permisos que tenga el usuario.

Fig. 6.6. Listado de puntos

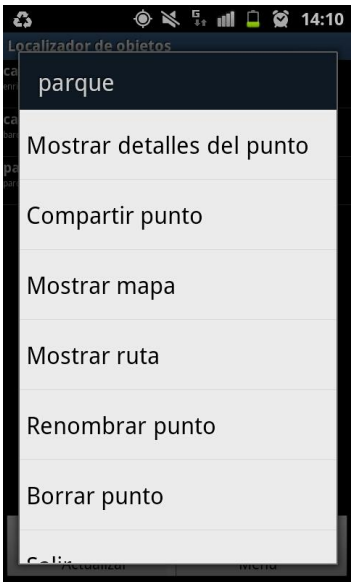

Fig. 6.7. Menú contextual

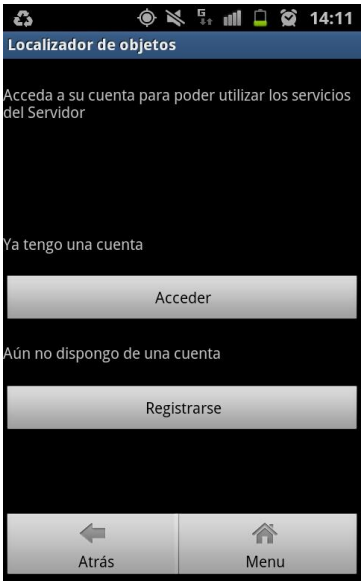

Fig. 6.8. Pantalla para acceder o registrarse

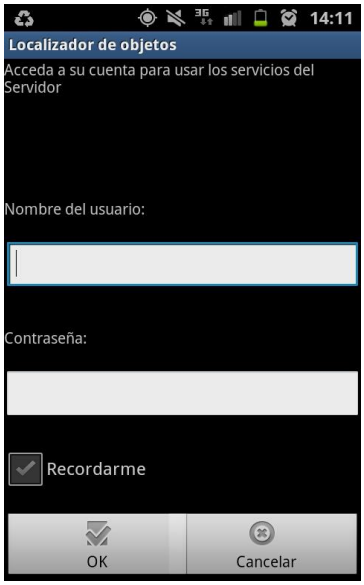

Fig. 6.9. Pantalla acceder

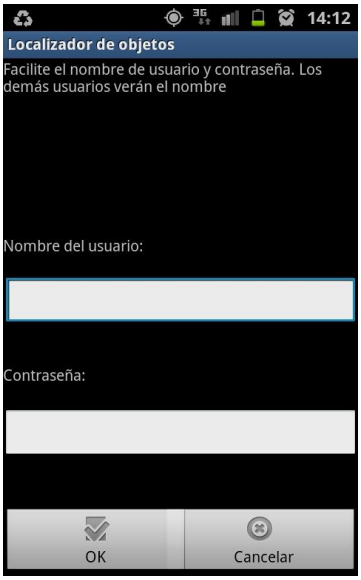

Fig. 6.10. Pantalla registrars

6.2.1. Mostrar mapa

Al pulsar la opción Mostrar mapa nos aparecerá una pantalla con el mapa de Google Maps con la ubicación del punto y las mismas opciones de desplazamiento y zoom explicadas en el primer apartado.

Al pulsar el botón menú del Terminal tendremos las siguientes opciones:

- Ir hasta aquí: nos mostrará la misma pantalla que pulsando el botón Mostrar ruta
- Mi posición
- Configuración Vista
- Menú

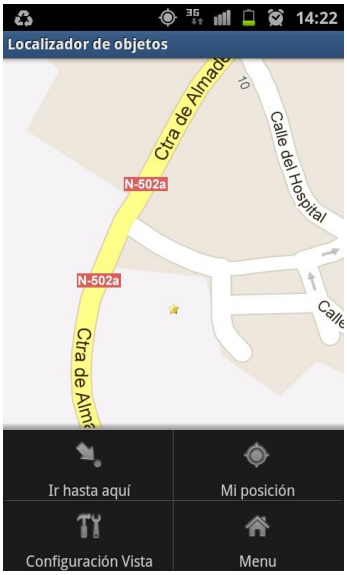

Fig. 6.11. Pantalla mostrar mapa

6.2.2. Mostrar Ruta

Al pulsar la opción Mostrar ruta nos aparecerá un diálogo en que el se pregunta al usuario si desea la ruta a pie o a coche. Una vez seleccionada la opción, mostrará una pantalla con el mapa Google maps y la ruta desde la posición en la que se encuentra el usuario hasta el punto seleccionado con las opciones de desplazamiento y zoom en el mapa.

Al pulsar el botón menú del Terminal tendremos:

- Información del tráfico
- Mi posición
- Configuración Vista
- Menú

Fig. 6.12. Opciones de ruta

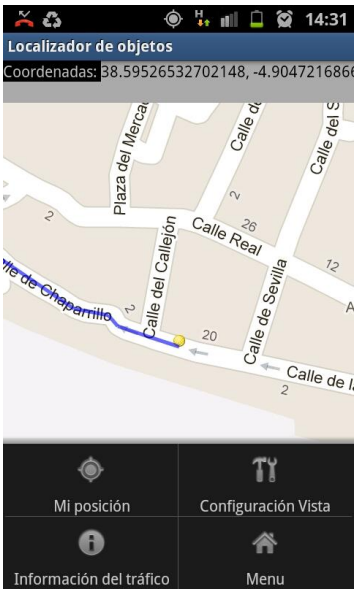

Fig. 6.13. Pantalla ruta

CAPÍTULO 7. MEJORAS EN LA APLICACIÓN

Una vez probada la aplicación en el Terminal se ha detectado una serie de puntos que podrían mejorar el funcionamiento de la aplicación. Se listan a continuación.

- Localización Wifi

  Inicialmente la aplicación sólo obtenía la localización a través de la señal GPS. Una vez probada en el Terminal, si la señal GPS no se recibe hace imposible el funcionamiento de la aplicación.
  Para garantizar su funcionamiento, se desarrolla la opción de que, en caso de que la señal GPS no se encuentre, muestre la localización mediante redes móviles o Wifi temporalmente hasta que la señal GPS esté disponible.

- Ruta a pie

  La aplicación sólo mostraba la ruta hacia el punto de destino en coche. Durante la realización de la memoria, se encontró la manera de solicitar la ruta a pie.

CAPÍTULO 8. PLANIFICACIÓN DEL PROYECTO

En este apartado se especifican los pasos que se han seguido para la realización del proyecto y las horas dedicadas.

Acción	**Horas**
Repaso de programación en Java.	5h.
Formación de Android y preparación del equipo de trabajo.	20h.
Aprendizaje de programación en Android y desarrollo de la aplicación en local.	102h.
Desarrollo del Servidor y funciones del Terminal relacionadas con el Servidor	39h.
Cambios en la aplicación para mejorar el funcionamiento de ésta una vez probada en el Terminal.	35h.
Escritura de la memoria y revisión de la aplicación.	53h.
TOTAL	**255 h.**

CAPÍTULO 9. CONCLUSIONES

El proyecto tenía como objetivo la realización de una aplicación Android que fuera útil para los usuarios y como consecuencia el estudio de dicho sistema operativo.

Los dos objetivos han sido completados ya que hemos conseguido llevar a cabo la aplicación con las funcionalidades deseadas y se ha adquirido un nivel alto de conocimiento de la plataforma Android.

Para aprender a desarrollar aplicaciones tenemos una gran cantidad de información y tutoriales disponibles, aparte de todos los tutoriales que ha publicado *Google* para el desarrollo de aplicaciones.
Java es el lenguaje de programación utilizado para programar todo el código, tanto para la aplicación como para la comunicación cliente-servidor. Esto ha facilitado las cosas, ya que se ha realizado un recordatorio tanto teórico como práctico de asignaturas de la universidad y no un aprendizaje de un nuevo lenguaje de programación.

La realización de este proyecto me ha servido, personalmente, para desarrollar una buena metodología de trabajo y unos conocimientos que mejoran mi nivel curricular.

9.1. Posibles mejoras

A continuación se ofrece un listado de mejoras que darán un mejor funcionamiento a la aplicación y no se han implementado:

- La interfaz gráfica puede mejorarse, de esta manera conseguiríamos que la aplicación fuera más atractiva para el usuario.

- Añadir una fotografía del punto en caso de que el usuario lo desee y guardarla en la base de datos.

- Mejora de la implementación de layouts, la orientación tiene que ser vertical. En caso de que sea horizontal no se ven correctamente algunas de las layouts creadas.

- Ordenar puntos, ordenarlos por nombre o por el usuario que lo haya compartido, ya que los puntos se visualizan en el orden que se hayan guardado en la base de datos.

- Búsqueda de puntos, buscar un punto en concreto o los puntos de un usuario: la base de datos puede llegar a ser extensa y dificultaría encontrar el punto deseado.

- Puntos temporales, eliminar los puntos después de pasar un determinado tiempo en el servidor para que sea una base de datos más dinámica ya que la aplicación suele utilizarse para encontrar lugares que el usuario no sabe donde están ubicados.

- Mostrar más información al usuario, el archivo solicitado en KML proporciona mas información a parte de la ruta con los datos de latitud y longitud para poder mostrarla en el mapa, del fichero se puede extraer la información de tiempo y distancia al punto de destino, y las indicaciones para llegar al punto como información adicional para el usuario.

- Haciendo referencia al punto anterior, con la *API* de *Google Maps* también se puede extraer la dirección de la posición donde se encuentra el usuario, guardando la dirección como otro campo en la base de datos, y así tener información más detallada del punto.

BIBLIOGRAFÍA

Webs

- Tutorial Oracle

  http://docs.oracle.com/javase/tutorial/index.html

- Tutorial eclipse

  http://dis.um.es/~bmoros/privado/bibliografia/tutorial%20eclipse%20para%20novatos%20java%20%28Pollino%29.pdf

- Tutorial Android

  http://www.sgoliver.net/blog/?p=1313

- Tutorial para la instalación del ADT para eclipse

  http://developer.android.com/sdk/eclipse-adt.html

- Web para solicitar la clave de uso de la API de Google Maps

  http://code.google.com/intl/es-ES/android/maps-api-signup.html

- Tutorial SQL

  http://cursos.atica.um.es/DBA9i1/sqlplus.html

- Tutorial para la instalación de MySQL y utilización de de MySQL con Java

  http://www.chuidiang.com/java/mysql/InstalacionMySQL.php

- Traductor para obtener latitud y longitud a través de una dirección

  http://translate.google.es/translate?hl=es&langpair=en|es&u=http://www.getlatlon.com/

Anexo I: Objetos *Intent*, *Bundle* y notificaciones *Toast*.

```
// Iniciamos otra Actividad desde la actual, llamando a la
clase deseada
Intent intent = new Intent(Main.this,Guardar.class);

// Objeto Bundle, introducimos datos necesarios para la
siguiente actividad.

Bundle bundle=new Bundle();
bundle.putString("Accion", "Menu");

intent.putExtras(bundle);

//llamamos a la siguiente actividad solicitándole un resultado
startActivityForResult(intent,0);

//En el caso de que no queramos que nos devuelva un resultado
startActivity(intent);
```

Para poder devolver un resultado a la Actividad anterior utilizaremos el siguiente método.

```
Intent data = new Intent();
setResult(RESULT_OK, data)
```

El siguiente método se ejecutará cuando reciba el resultado de la actividad llamada, en él insertamos el código de notificación tipo Toast y el objeto Bundle del cual se extraen los datos que nos pasa la Activitie anterior.

```
protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(resultCode == RESULT_OK){

      Bundle l= data.getExtras();

      if(l.getString("Accion").equals("Menu")){

      //Informaremos al usuario mediante un mensaje de
      notificación Toast.

      Toast T= Toast.makeText(getApplicationContext(), "El
      proceso de conexión con el Servidor y creación del
      punto se ha cancelado",Toast.LENGTH_LONG);

      T.show();

      }
    }
}
```

Anexo II: Manifest.xml y R.java

Manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>

<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
      package="lidia.ibern"
      android:versionCode="1"
      android:versionName="1.0">

    <uses-sdk android:minSdkVersion="7" />


    <application                  android:icon="@drawable/icon"
android:label="@string/app_name">

      <uses-library  android:name="com.google.android.maps"/>
        <activity android:name=".Localizadordeobjetos"
                  android:label="@string/app_name">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
         </activity>
         <activity android:name=".guardar"></activity>
         <activity android:name=".buscar"></activity>

    </application>

   <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"/>

</manifest>
```

R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */
package lidia.ibern;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_menu_back=0x7f020001;
        public static final int ic_menu_home=0x7f020004;
    }
    public static final class id {
        public static final int BtnBuscar=0x7f060026;
        public static final int BtnCancelar=0x7f06001c;
    }
    public static final class layout {
        public static final int main=0x7f030005;
        public static final int buscar=0x7f030006;
        public static final int guardar=0x7f030007;
    }
    public static final class menu {
        public static final int menu_listview=0x7f050000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Anexo III: Método *OnCreate()* e implementación de controles

Método OnCreate(), utilizado para llamar a la Actividad y establecer la interfaz gráfica.

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.pantalla_principal);
//Aquí crearemos todos los objetos que hacen referencia a los
elementos de la layout que tengamos que utilizar para la
implementación de la pantalla.
      Button btnGuardar = (Button)findViewById(R.id.BtnGuardar);
}
```

Establecemos como interfaz el xml creado como pantalla_principal
Implementación de los diferentes controles utilizados:

Button

```
btnGuardar.setOnClickListener(new OnClickListener(){

      @Override
      public void onClick(View arg0) {

                        //Código
      }
 });
```

Check Box

```
CheckBox.setOnCheckedChangeListener(new
CheckBox.OnCheckedChangeListener(){

      public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked){

            if(isChecked){
            //Código
            }

            else{
            //Código
            }
          }
});
```

Anexo IV: Sentencias SQL

A continuación mostramos las sentencias SQL utilizadas para la comunicación con la base de datos SQLite y MySQL.

```
//Creación de la tabla SQLite
CREATE TABLE nombre_tabla (valor1 INTEGER, valor2 TEXT);

//Creación de la tabla en MySQL
CREATE TABLE IF NOT EXISTS nombre_tabla (valor1 INTEGER,valor2
VARCHAR(20);
```

En SQLite cuando la base de datos es llamada, en caso de que no exista se crea por defecto con el método OnCreate(). Por lo contrario en MySQL se ha de crear la base de datos directamente por consola, si no es creada dará error.

```
//Creación de la Base de datos en MySQL
create database nombre_base_datos;
```

```
//Insertar un registro
INSERT INTO nombre_tabla (campo1,campo2) VALUES
('valor1','valor2')

//Eliminar un registro
DELETE FROM nombre_tabla WHERE campo='valor'

//Actualizar un registro
UPDATE nombre_tabla SET campo_a_cambiar='valor' WHERE
campo='valor'

// Consulta de la tabla
SELECT * FROM nombre_tabla
```

En caso de que queramos consultar determinados campos se sustituye * por el nombre de los campos deseados separándolos con comas.
Si lo queremos de un determinado registro añadiremos la cláusula WHERE indicando el registro que estamos buscando como se muestra en actualizar o eliminar registro.