Guía de Puntos de Interés de la Ciudad de Madrid

Memoria Final de Proyecto

Daniel Cortés Fernández

Proyecto Fin de Carrera - Ingeniería Informática (2011-2012)

Consultor de Proyecto: Víctor Carceler Hontoria

1. Índice

Índice de Ilustraciones

2. Introducción

El presente documento se corresponde con la Memoria Final del Proyecto de Fin de Carrera de los Estudios de Ingeniería Informática (en adelante PFC) realizados en la Universidad Oberta de Catalunya.

El objetivo de este documento es la presentación formal de la totalidad del trabajo realizado en relación con este PFC.

En primer lugar se apuntarán los objetivos que se pretenden alcanzar con la realización de este proyecto, así como el alcance que tendrá el mismo.

Una vez fijados los objetivos, definiremos un caso de estudio. Mediante la resolución de este caso alcanzaremos los objetivos planteados inicialmente. El caso de estudio elegido es un caso que podría darse en el mundo laboral: una guía turística para una ciudad.

Expondremos el caso de estudio, explicando la funcionalidad que el hipotético cliente querría obtener como resultado final del proyecto.

Una vez introducido el caso de estudio, se establecerá una planificación para abordar el proyecto. En la planificación se realizará un desglose que mostrará las tareas a realizar y un calendario de ejecución estimado para las mismas.

En el apartado de análisis se explicará el enfoque metodológico utilizado para la extracción de requisitos y ejecución del proyecto. Se realizará también un análisis de negocio (a alto nivel) que nos permitirá aislar y estudiar por separado las diferentes funcionalidades solicitadas por el cliente. También estableceremos en esta fase la interfaz gráfica de usuario que, una vez aprobada por el cliente, permitirá crear un prototipo en una fase temprana del proyecto. Junto con el prototipo, de definirá un plan de pruebas que ayudará garantizar la correspondencia entre los requisitos expresados verbalmente por el cliente y el software resultado de la realización de este PFC.

El apartado de implementación explica, en primer lugar, cómo se ha configurado el entorno de desarrollo necesario para crear el software. Una vez operativo el entorno, se explicarán las soluciones técnicas adoptadas para implementar las funcionalidades aisladas y estudiadas en el apartado de análisis.

Con el fin de facilitar la compresión del trabajo realizado y de introducir explicaciones técnicas exhaustivas, se han creado varios documentos anexos a esta memoria:

- Anexo I – Terminología aplicable
- Anexo II – Tecnologías de desarrollo
- Anexo III – Plan de Trabajo
- Anexo IV – Configuración del entorno de desarrollo
- Anexo V – Bibliografía
- Anexo VI - Código

3. Objetivos y alcance del proyecto

A continuación se presentan los objetivos que se pretenden alcanzar mediante la realización de este PFC, así como las competencias que se espera haber adquirido tras la finalización del mismo.

Los objetivos del proyecto son:

- Diseñar un Sistema de Información Geográfica destinado a funcionar en terminales móviles.
- Conocer el Sistema Operativo para móviles Android
- Manejar las características más comunes del sistema operativo Android (manejo de BBDD embebidas, sensores del terminal, GPS, cámara fotográfica y galería de imágenes)
- Conocer las funcionalidades ofrecidas por el API Google Maps

Se pretenden, además, adquirir las siguientes competencias:

- Capacidad de gestión de proyectos
- Capacidad de presentar los resultados de un proyecto de forma clara y efectiva

Como resultado de la realización de este proyecto se obtendrá:

- **Una memoria final**
- **Un prototipo** instalable, funcionando sobre la versión 2.2 de Android
- **Una presentación**

Con el fin de alcanzar estos objetivos, se ha pensado desarrollar una aplicación de guía turística para terminales móviles. Este tipo de aplicación engloba todos los conocimientos y habilidades que pretendemos adquirir con la realización de este PFC.

4. Definición del problema

4.1 Introducción

En la actualidad, los avances tecnológicos, en materia de Telecomunicaciones e Informática han posibilitado la producción de teléfonos inteligentes y tabletas digitales de altas prestaciones a precios asequibles. La combinación de tecnología y precio competitivo ha tenido como resultado que un alto porcentaje de población utilice actualmente teléfonos móviles inteligentes y demande cada vez más aplicaciones pensadas específicamente para estos terminales.

Es en este contexto dónde surge la idea de aprovechar las ventajas que ofrecen los Smart phones, no solo en cuestiones multimedia sino también sus capacidades de geolocalización, para construir una guía turística que muestre información de forma más atractiva que las guías tradicionales con el valor añadido de ayudar al usuario a llegar a los lugares de visita e incluso poder registrar la experiencia del usuario en forma de fotografías.

Para llevar a cabo esta idea se ha elegido el sistema operativo Android por las siguientes razones:

- Alta implantación en el mercado
- Expectativas de crecimiento a corto plazo
- Documentación abundante y de libre acceso
- Gran variedad de APIS para resolución de problemas comunes
- Integración con otros standards tecnológicos
- Compatibilidad con múltiples dispositivos de diversas marcas.

La funcionalidad detallada de la guía se introducirá en el siguiente apartado mediante un caso de estudio real.

4.2 Caso de estudio

Con el fin de alcanzar los objetivos propuestos en este PFC, se desea **crear una guía turística para dispositivos móviles**, que ofrezca información sobre puntos de interés cultural que pueden visitarse en la ciudad de Madrid.

La información ofrecida sobre los puntos de interés puede ser de varios tipos:

- Información general del punto (histórica, de servicio)
- Galería fotográfica
- Información geográfica del usuario

Información general

Se entenderá “información general”, como toda aquella información referente al punto de interés que el usuario pueda consultar. Esta información podrá ser de tipo histórico, arquitectónico, o bien información de servicio (horarios, tarifas, descuentos).

La guía facilitará el desplazamiento entre los diferentes puntos de interés, de forma que puedan recorrerse todos fácilmente.

Galería fotográfica

La galería fotográfica ofrecerá información visual sobre los puntos de interés, bien sean las obras más significativas de un museo, perspectivas de un monumento o alrededores de algún lugar catalogado.

Información geográfica

La guía deberá poder capturar y manejar información sobre la ubicación del usuario con dos fines:

- Mostrar la posición del usuario respecto a los puntos de interés a visitar.
- Ayudarle a llegar a estos puntos calculando la ruta que debe seguir.

Gestión de puntos de interés definidos por el usuario

Como valor añadido, un usuario podrá guardar sus propios puntos de interés, es decir, lugares que al usuario le sean significativos por algún motivo, pero que la guía no los considera de un interés turístico relevante. Para ello se ha pensado en utilizar la cámara fotográfica y la galería de fotografía del terminal; de esta forma, se podrán dar de alta puntos de interés de usuario y guardarse con una imagen asociada.

En resumen, cada punto de interés catalogado incluirá:

- Posibilidad de consulta de información general sobre el punto
- Consulta de galería fotográfica asociada al punto de interés
- Localización del punto de interés sobre mapa zonal
- Calculo de ruta de llegada a un punto de interés determinado desde

Los puntos de interés de usuario contarán con un sistema de gestión que permitirá:

- Crear, consulta o borrar un punto de interés de usuario
- Captura de imágenes mediante cámara fotográfica
- Captura de imágenes desde la galería del terminal (para puntos de usuario)
- Búsquedas autocompletadas

5. Planificación

Enunciado el problema, se impone la necesidad de definir y describir las tareas que van a llevarse a cabo, así como el establecimiento de un marco temporal para la ejecución del proyecto. La ubicación de las tareas a realizar en este marco temporal nos ayudará a evaluar el grado de avance del proyecto y la necesidad de tomar medidas correctoras en caso de que se produzcan desviaciones en las estimaciones iniciales.

En el ***Anexo III - Plan de Trabajo***, pueden consultarse las tareas que se llevarán a cabo en el proyecto, la duración estimada de las mismas, así como las fechas previstas para abordarlas. Además, se establece un calendario de entregas cuyo cumplimiento garantizará el éxito del proyecto.

Este plan de trabajo incluye igualmente:

- **Una relación de los roles intervinientes** en el proyecto y sus asignaciones
- **Una relación del material necesario** para ejecutar el proyecto
- **Un apartado de riesgos** en el que se describen los posibles problemas que podríamos encontrar durante la ejecución del proyecto.
- **Una relación de acciones mitigadoras** previstas para paliar los efectos derivados de estos problemas.

Esta información se acompaña con un diagrama de Gantt que aporta una visión clara de la planificación temporal.

6. Análisis de la aplicación

Los requisitos funcionales son una parte fundamental del proyecto, en cuanto que dirigen el desarrollo del mismo hacía un fin concreto. Una vez introducido el problema que debemos resolver es necesario proceder a realizar un análisis del mismo para extraer conclusiones y organizar las tareas que debemos ejecutar. Con el fin de servir de guía para el desarrollo hemos optado por escoger una combinación entre **SCRUM** y **eXtreme Programming**. A continuación se explicará el motivo de esta elección, así como la construcción de los requisitos, extraídos del problema planteado.

6.1 Metodología de desarrollo

A la hora de la elección de la metodología de desarrollo tuvimos en cuenta los dos grandes problemas a los que nos enfrentamos:

- Desconocimiento de todas las tecnologías implicadas
- Escasez de tiempo

Estos factores indican que lo más indicado era adoptar algún proceso de metodología ágil, como XP, sin embargo, era inviable cumplir con todas sus recomendaciones.

Finalmente, se consideró como mejor opción la utilización de una combinación entre SCRUM y eXtreme Programming.

6.2 ¿Qué propone SCRUM?

SCRUM propone un desarrollo incremental, basado en iteraciones, mejorando el producto en cada una de ellas. Estas iteraciones son conocidas como **SPRINTS** en la metodología SCRUM. Los SPRINTS tienen que ser de duración corta y fija (a ser posible). Tras cada SPRINT se obtiene un producto final que se evalúa y sobre el que sacan conclusiones para ver en qué estado se encuentra el proyecto (si los avances son los esperados, si la funcionalidad es la requerida por el cliente, etc..).

Formación de un grupo de trabajo SCRUM

Típicamente, un equipo que quiera realizar SCRUM debería estar conformado por:

- Jefe de producto
- Desarrollador/es
- Jefe de Proyecto

Es necesario comprender estos roles para comprender el proceso que va a explicarse a continuación.

El jefe de producto es el encargado de reunirse con el cliente y captar las ideas y funcionalidades que quiere para el aplicativo, su responsabilidad es que el proyecto salga adelante en coste y fechas previstos.

Los desarrolladores son las personas que van a traducir todo lo solicitado por el cliente a código informático, con un análisis mínimo previo. Igualmente, van a estimar tiempos de realización de funcionalidades.

El jefe de proyecto, actúa como moderador entre los otros dos colectivos. Vigila que el jefe de producto no exija cosas inalcanzables y que los desarrolladores no den estimaciones de

tiempo infinitas y no se pierdan en divagaciones técnicas interminables. Es un árbitro que posee conocimiento técnico y que, en caso de no lograr consenso entre el jefe de producto y desarrolladores, tiene que marcar la línea a seguir (y el resto del equipo se compromete a respetarla).

En la ilustración 1 se observa un proceso típico en SCRUM.

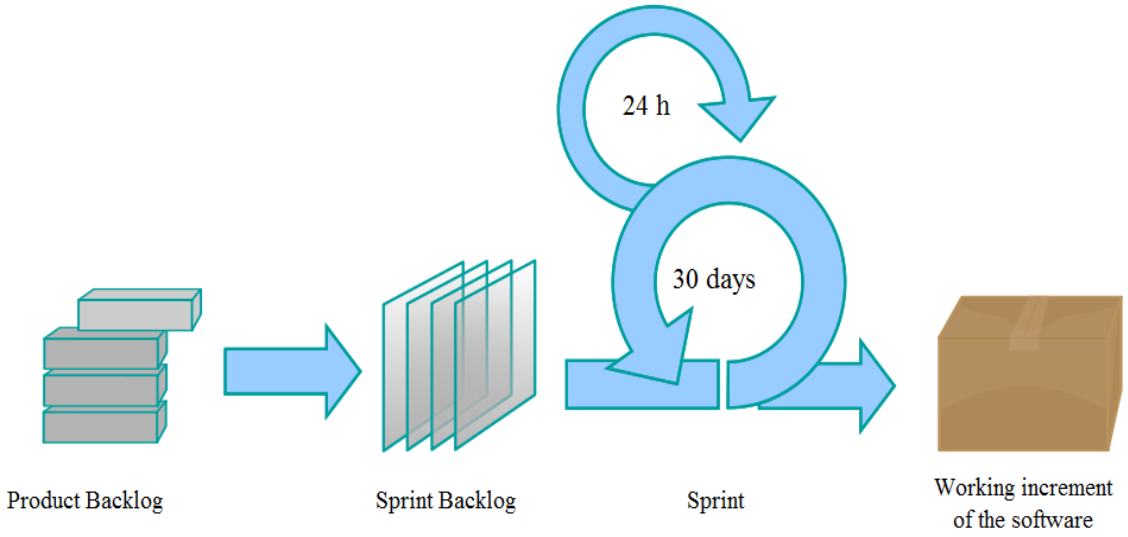


Ilustración 1 - Proceso SCRUM

SCRUM funciona recogiendo los requisitos del producto en una pila (Backlog en la ilustración). Los requisitos son capturados en forma de funcionalidad completa, y cada una de estas funcionalidades recibe el nombre de "Historia de Usuario" o, simplemente, “Historia”.

Una historia puede ser cualquier funcionalidad que el usuario ha pensado que puede hacer con la aplicación.

Previamente a la realización de un SPRINT, se consensua el número de historias que se incluirán en él. Debe existir un acuerdo entre todos los colectivos del equipo SCRUM, y en caso de no haberlo, será el jefe de proyecto quien decida. El número de historias seleccionadas serán las historias que se implementarán o corregirán en ese SPRINT, este número de historias está representado como SPRINT BACKLOG en la ilustración.

A partir de cada historia, podemos obtener una lista de tareas a realizar para implementarla. Las tareas son unidades más concretas y se centran en aspectos muy pequeños y determinados de cada historia.

En el caso de este proyecto de fin de carrera, se ha utilizado la filosofía SCRUM para:

- Capturar los requisitos de usuario
- Definir SPRINTS
- Trabajar en el producto de forma incremental, añadiendo pequeñas mejoras tras cada SPRINT

6.3 Requisitos de la aplicación: Historias de Usuario

En este apartado se configuran las historias de usuario, fruto del análisis del problema que debemos resolver, expresado en el apartado 4, ***“Definición del problema”***.

Historia de "Consultar punto de interés"

Un usuario podrá obtener información sobre un punto de interés. Esta información puede variar entre información del contexto histórico, de la arquitectura del lugar o información general (horarios, precios, etc…).

La información podrá estar completada por fotografías de interés: pictórico, arquitectónico o de situación.

Historia de "Mapa de Situación"

Un usuario que está en Madrid desea saber qué puntos de interés están cercanos al lugar dónde se encuentra en este momento y, de esta forma, poder planificar el recorrido que quiere realizar.

Historia de "Como Llegar"

Un usuario que está en Madrid quiere saber cómo llegar a un sitio de interés determinado desde la ubicación en la que se encuentra actualmente.

Historia de "Mis puntos favoritos"

Al usuario debe poder guardar en su terminal puntos de la ciudad que le han gustado, independientemente de si están catalogados como de interés o no. De esta forma podría tenerlos para visitas posteriores o compartirlos con personas de su entorno.

6.4 Identificación de Casos de Uso

Los requisitos contenidos en las historias de usuario permiten identificar los casos de uso que manejaremos en este proyecto. El siguiente diagrama pretende mostrar los casos de uso identificados, así como las relaciones existentes entre ellos:

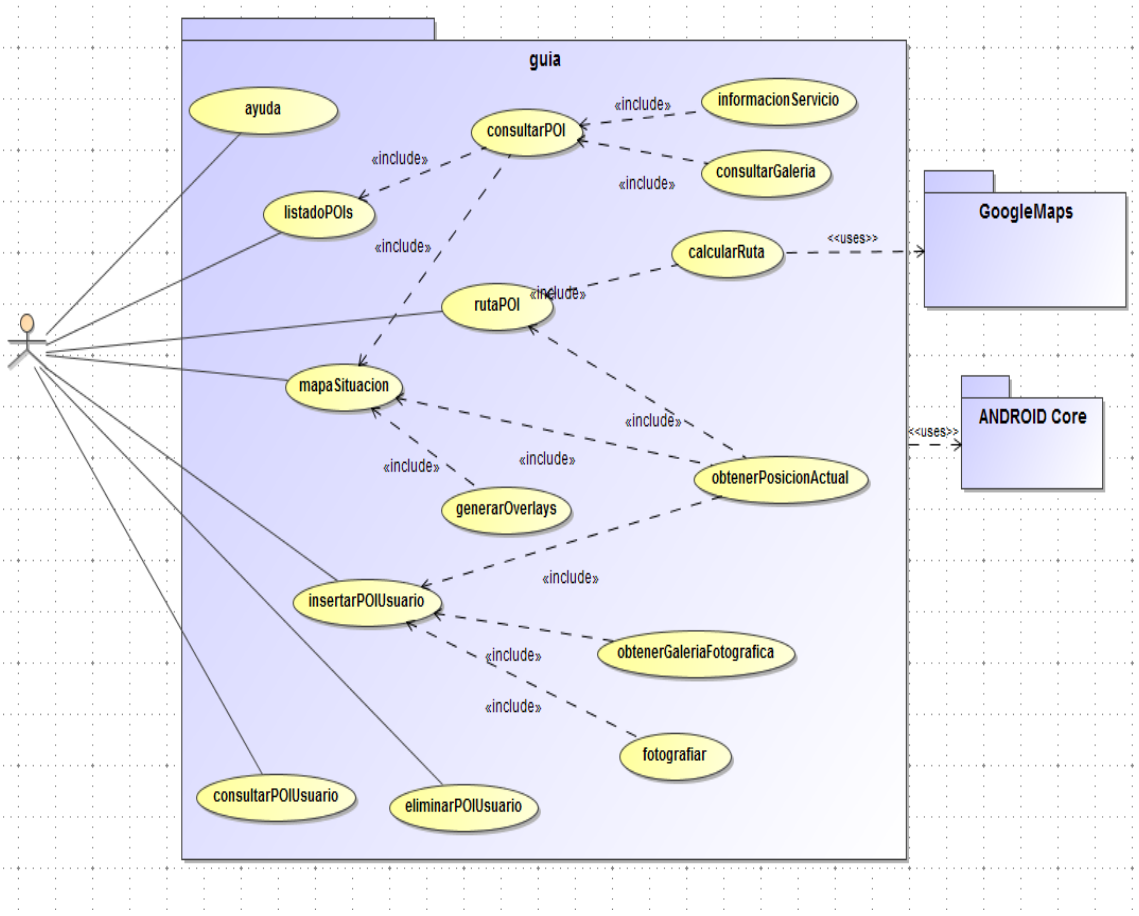


Ilustración 2 - Diagrama de casos de uso

Se ha considerado interesante incluir las llamadas a sistemas externos que sabemos que serán utilizados por nuestro aplicativo. Aunque estos productos externos serán comentados en capítulos posteriores, puede encontrarse su definición en el documento ***"Anexo II - Terminología aplicable"***.

7. Diseño del Sistema

Este capítulo se centrará en la propuesta de diseño para llevar a cabo las funcionalidades identificadas anteriormente. Para realizar los diagramas de diseño se ha utilizado la notación UML 2.0 tal y como se utiliza en el libro ***"El lenguaje Unificado de Modelado"***.

Como paso previo a la explicación del diseño de nuestro sistema es fundamental contar con algunos conceptos de la organización de Android a nivel de arquitectura. Estos conceptos nos ayudarán a comprender con mayor facilidad los diagramas de clases de nuestro proyecto y justificarán algunas de las decisiones adoptadas.

7.1 Arquitectura de clases Android

Las aplicaciones para plataforma Android giran en torno al concepto de **Actividad**. Una actividad es, según la documentación oficial de Google, ***"un componente de aplicación que provee una pantalla desde la cual el usuario puede interactuar con el aplicativo con el fin de realizar alguna opción determinada"***.

Ateniéndonos a esta definición, una aplicación no es más que un conjunto de actividades relacionadas entre sí y que cooperan para ofrecer al usuario una serie de funcionalidades también relacionadas.

El ciclo de vida de una actividad es el siguiente:

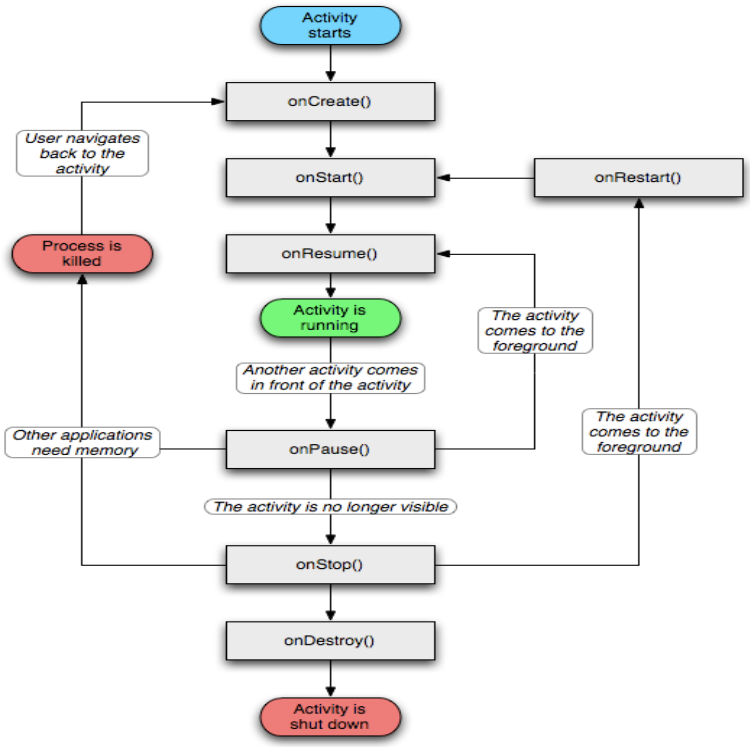


Ilustración 3 - Ciclo de vida de una Actividad Android

El núcleo del Sistema Android es el encargado de llamar a algunos de estos métodos cuando se producen eventos determinados. El desarrollador también puede llamarlos para manipular el estado de una acción en un momento dado.

7.2 Diagrama de clases del Sistema

En este apartado se realizarán los diagramas de las clases necesarias para construir la aplicación, así como las relaciones existentes a nivel estructural entre ellas.

Estableceremos una diferenciación a nivel conceptual entre las diferentes clases:

Por un lado estarán los ***beans***, entendiendo como "beans" los componentes destinados a ser reutilizados y que simplemente son estructuras de datos simples con métodos *get* y *set* (acceso y establecimiento).

Por otro lado están el resto de clases del aplicativo (Actividades, Adaptadores, Helpers, etc...) estás serán las clases encargadas de dotar de utilidad a la aplicación.

Diagrama de Beans

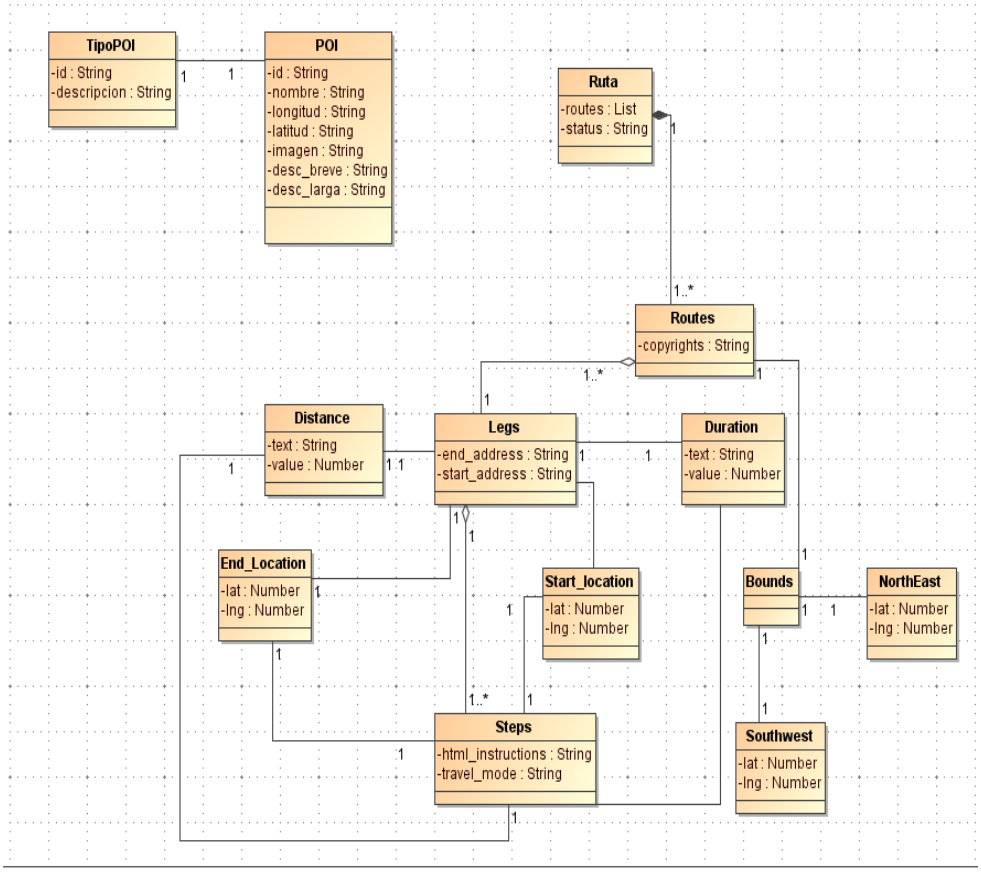


Ilustración 4 - Beans

Como se ha comentado, estas clases son simples almacenes de datos con métodos de establecimiento y acceso. No obstante, queremos destacar las clases más importantes a nivel de negocio:

- POI
- Ruta

La clase POI representa un punto de interés de la aplicación. Este punto de interés puede ser de diferentes tipos (monumento, museo, punto de usuario, etc ...).

La clase Ruta es una clase de utilidad necesaria para comunicar con el servicio de cálculo de ruta de Google Maps y su estructura está determinada por las especificaciones de este servicio de Google.

Diagrama de clases

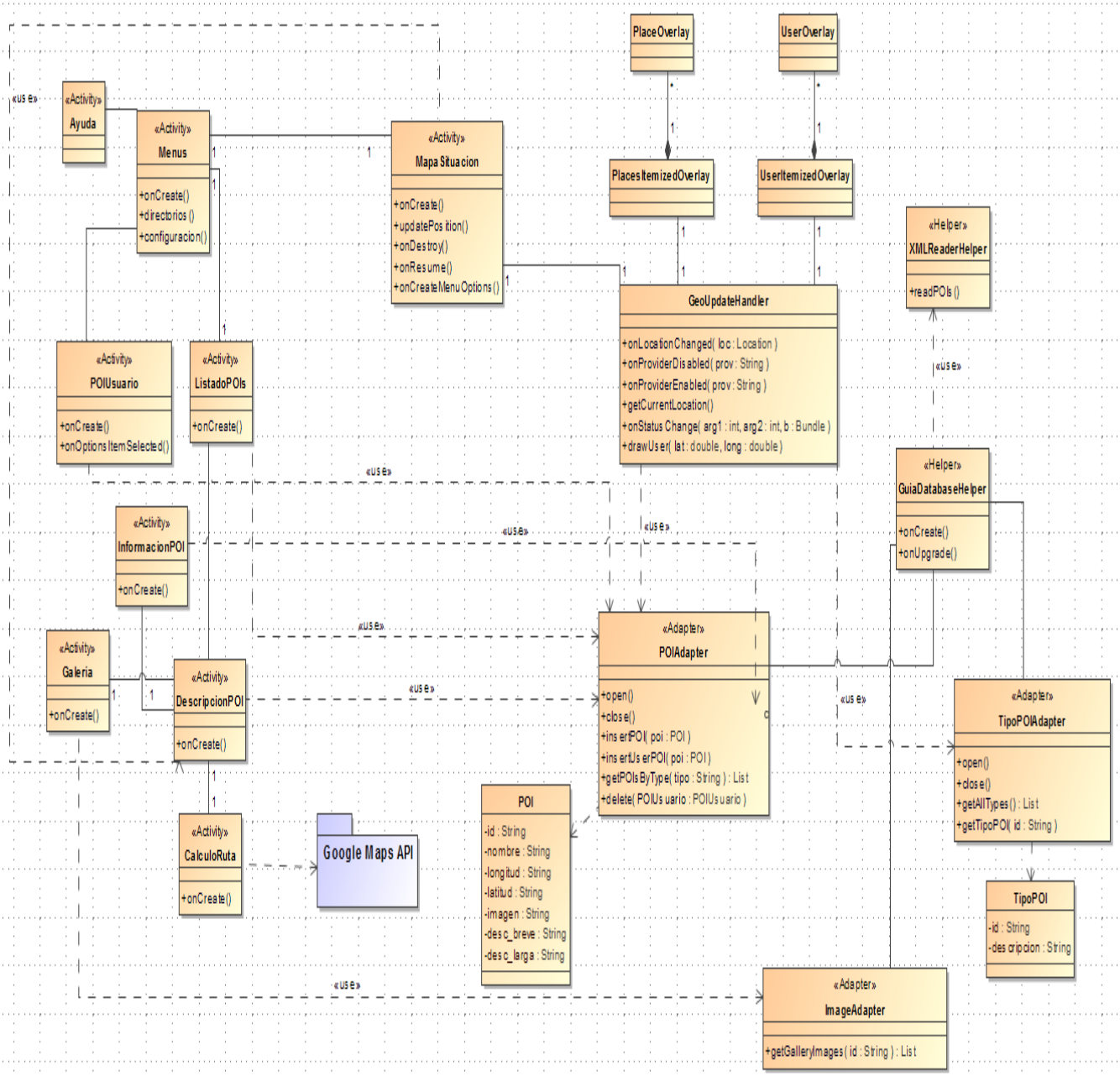


Ilustración 5 - Diagrama de clases

Podemos observar que el sistema, a nivel de clases, se constituye como un conjunto de actividades, estereotipadas como <<Activity>>, que realizan uso de ***Handlers y Helpers*** para realizar acciones tales como gestionar localizaciones o acceder a la base de datos. Cada actividad tiene un ciclo de vida análogo al de la Ilustración 3.

Los conceptos de *Handler* y *Helper* son conceptos comúnmente aceptados, procedentes de diversos *frameworks* basados en Java. *Handler* hace referencia a una clase que maneja algún tipo de dato significativo y opera con él, y *Helper* es referido a clases que ocultan alguna complejidad técnica muy recurrente, como accesos a base de datos o transformaciones XML.

7.3 Diagramas de solución técnica

En este apartado se explica el funcionamiento de las actividades más importantes mediante diagramas de secuencia. En estos diagramas se observa el desarrollo de la actividad, desde que el usuario (u otra actividad) invoca al método *onCreate* de la actividad Android, hasta que se muestran los resultados en la vista destinada a tal efecto.

Mapa de Situación

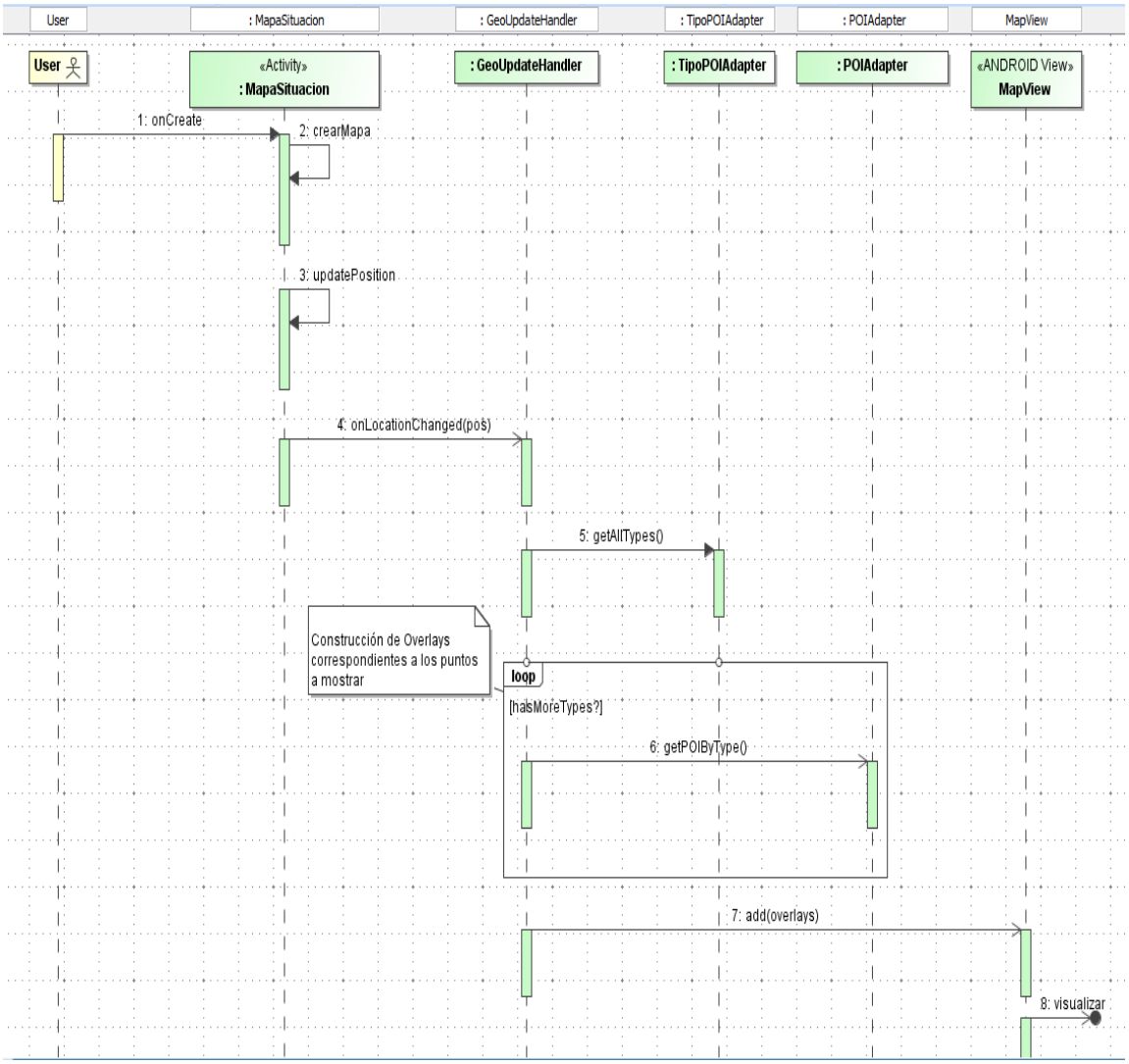


Ilustración 6 - Mapa de Situación

Esta funcionalidad es la encargada de ubicar los puntos de interés en un mapa accesible por el usuario.

Tal y como puede observarse en el diagrama, la actividad realiza una consulta de los puntos de interés catalogados en la base de datos y los superpone al plano zonal de la vista *MapView*.

El sistema Android informa al manejador de geolocalización (*GeoUpdateHandler*) de cualquier cambio producido en la localización del usuario, de tal manera que *GeoUpdateHandler* es capaz de repintar los elementos necesarios para adaptar el mapa a las nuevas circunstancias.

Calculo de ruta

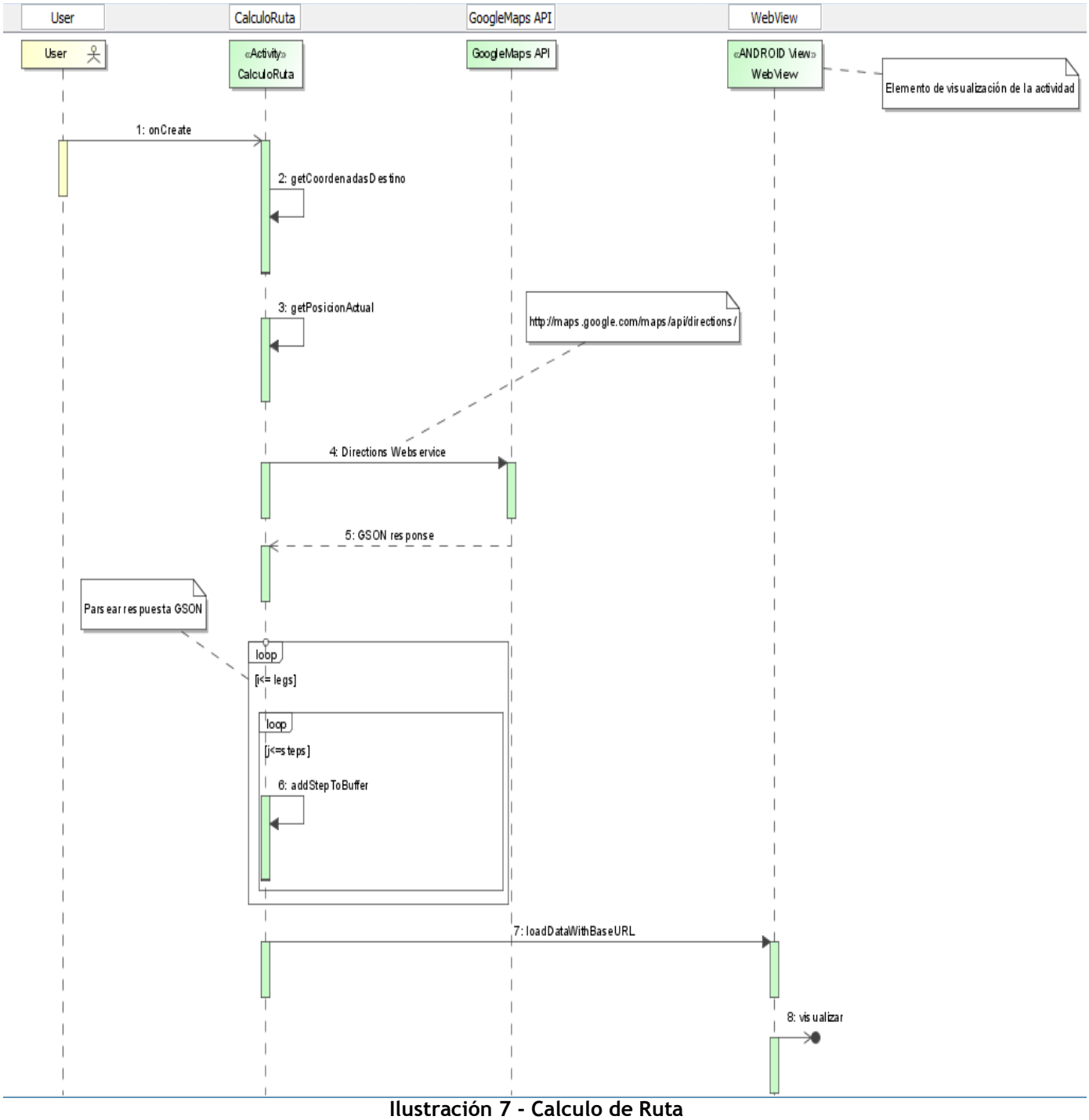


Ilustración 7 - Calculo de Ruta

Esta funcionalidad está basada en la llamada al servicio web *Directions* del API de Google Maps y en la interpretación de su respuesta.

Previamente a la llamada, necesitamos contar con las coordenadas de inicio y final del recorrido, ya que el servicio web las recibe como argumento.

Tras la llamada, se trata la respuesta, para acabar mostrándose en el marco visual de la Actividad de Android. En este caso, el marco visual lo presta la clase WebView, perteneciente al CORE de Android.

Listado de POIs

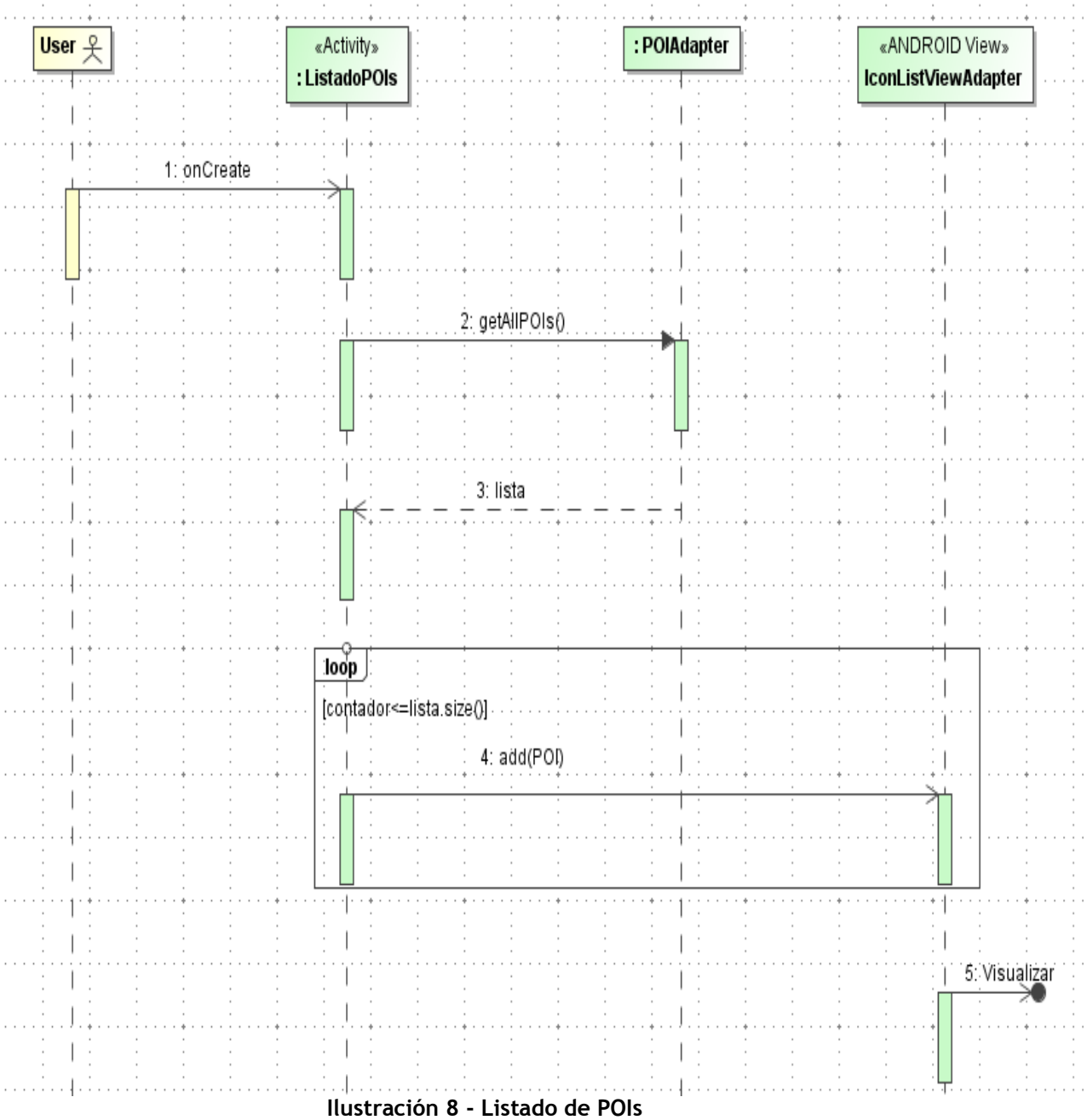


Ilustración 8 - Listado de POIs

Este diagrama pertenece a la Actividad Android que muestra al usuario el listado de POIs catalogados.

Puede observarse como se obtienen los puntos de interés que se encuentran en la BBDD por medio de su adaptador correspondiente. Una vez obtenidos se crea un adaptador de vista de lista que será el que configure el listado de puntos resultante.

Consultar un POI

Puede observarse en el diagrama que esta actividad debe ser llamada desde la actividad Listado de POIs. El funcionamiento básico consiste en la recogida del identificador de POI, pasado por ListadoPOIs y la consulta de todos los datos de este punto mediante el adaptador destinado a tal efecto.

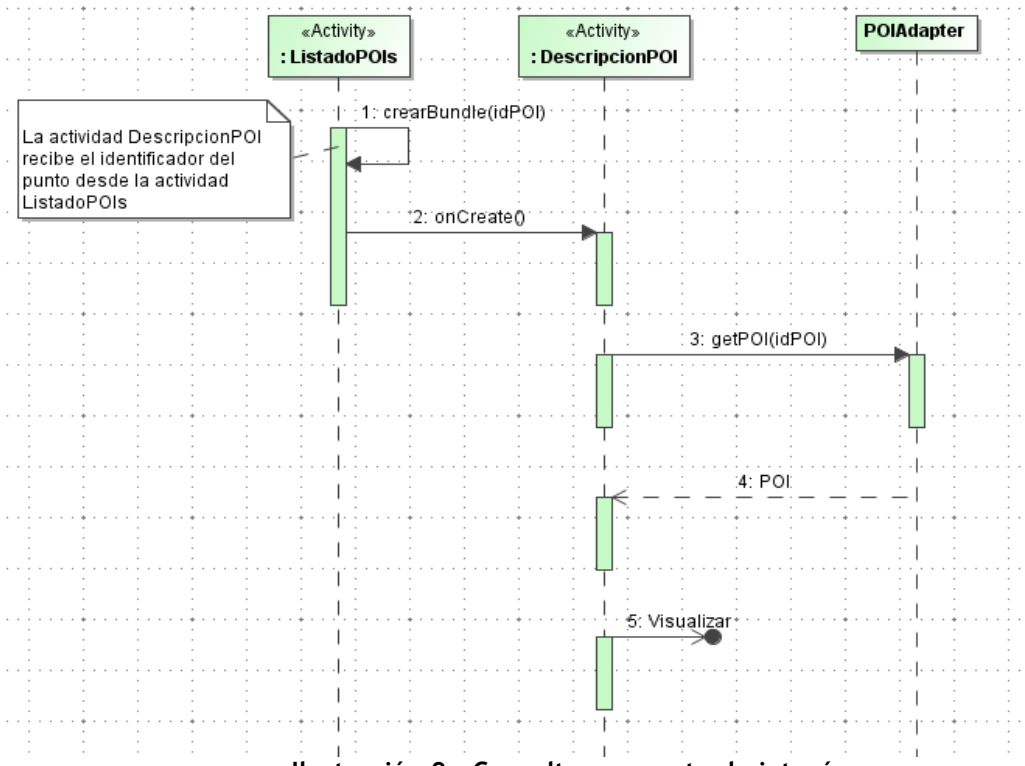


Ilustración 9 - Consultar un punto de interés

Consultar una galería

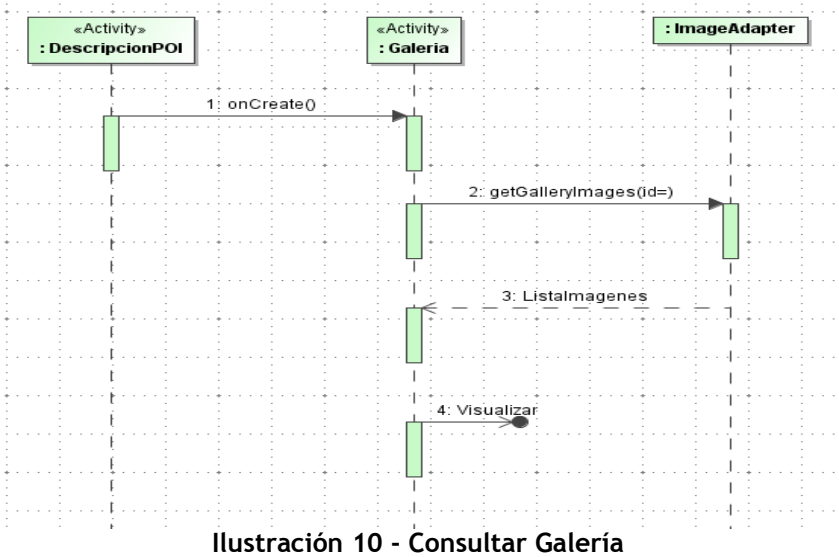


Ilustración 10 - Consultar Galería

Como muestra el diagrama, esta actividad es llamada por la actividad DescripcionPOI. La actividad Galería consulta las imágenes asociadas al POI que se estaba consultando en DescripciónPOI y las presenta utilizando su área de visualización.

Consultar información de servicio

Esta funcionalidad es llamada desde DescripcionPOI y utiliza a la clase POIAdapter para recuperar la información referente al punto indicado.

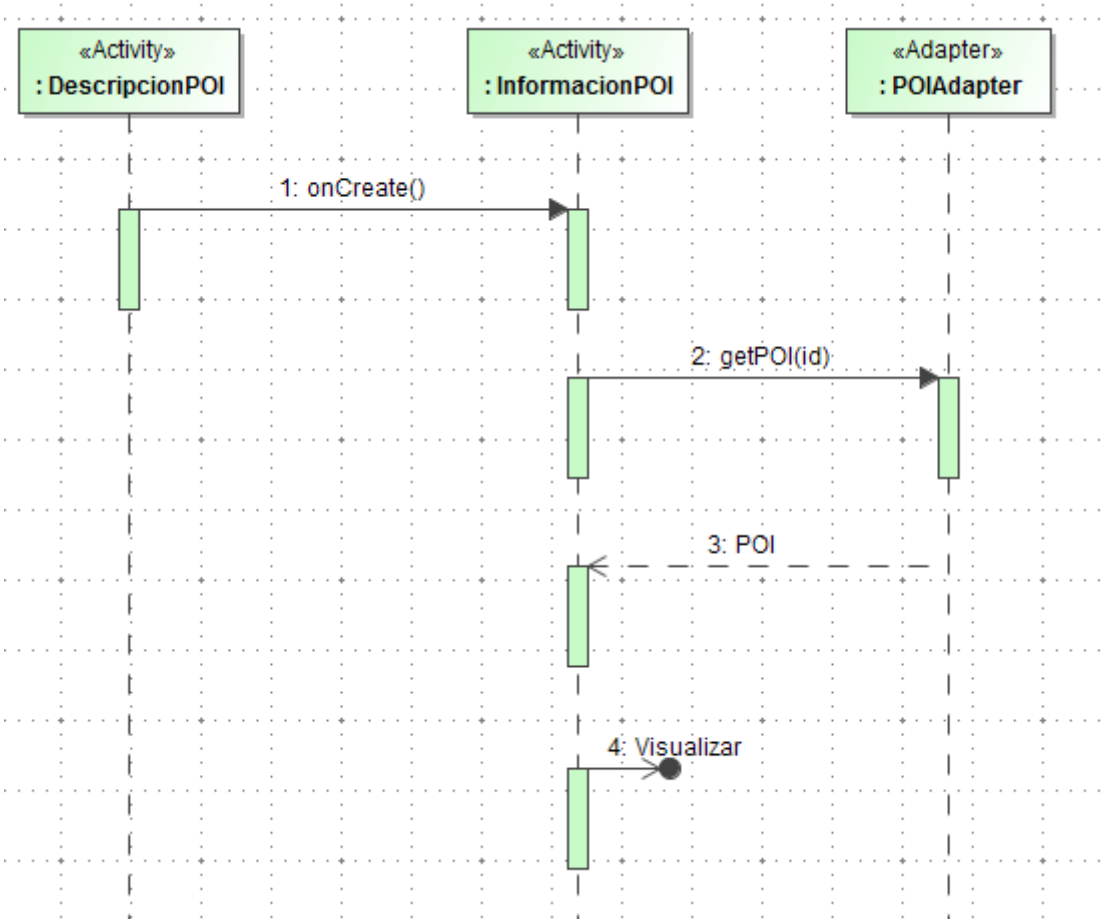


Ilustración 11 - Consulta de información de servicio

Insertar POI de Usuario

Este caso de uso nos permite almacenar un punto definido por el usuario. Tal y como se observa en la siguiente ilustración, la acción POIUsuario se encarga de obtener las coordenadas en las que se encuentra el usuario actualmente, para ello hará uso de la clase *GeoUpdateHandler*, dotada de la capacidad de manejar el sistema GPS del terminal.

Tras rellenar los datos necesarios por medio de campos de texto, el usuario tiene dos opciones para asociar una imagen al punto que está registrando. Se observa que, dependiendo de la elección, se invoca a la clase *MediaScannerConnectionClient*, si se va a tomar una fotografía del lugar o a la clase *MediaStore*, si se pretende seleccionar una fotografía procedente de la galería del terminal.

Por último, se delega la inserción del punto en el adaptador de puntos, *POIAdapter*.

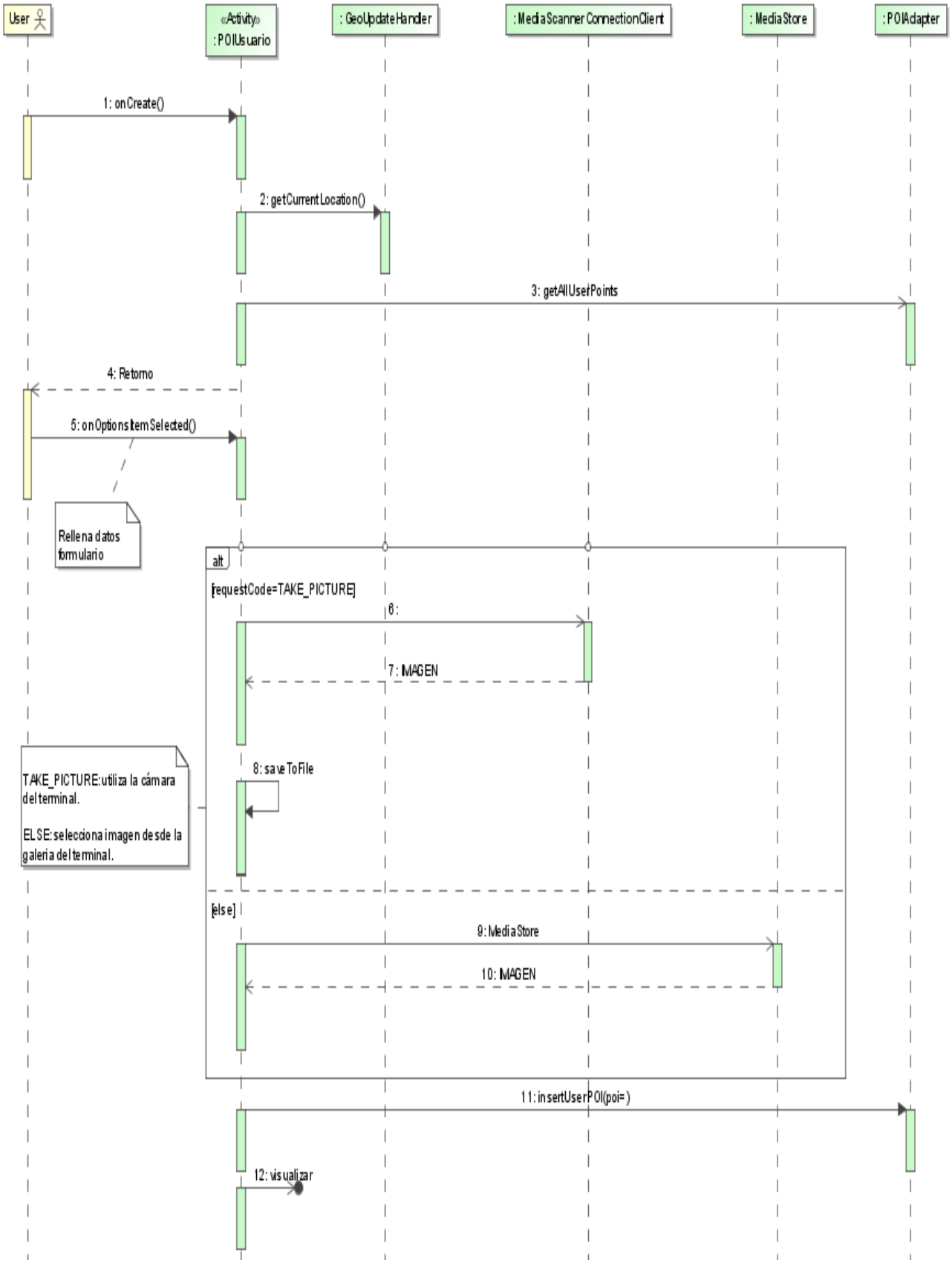


Ilustración 12 - Insertar punto de usuario

Eliminar un POI

Mediante este caso de uso podremos eliminar un punto de usuario previamente insertado.

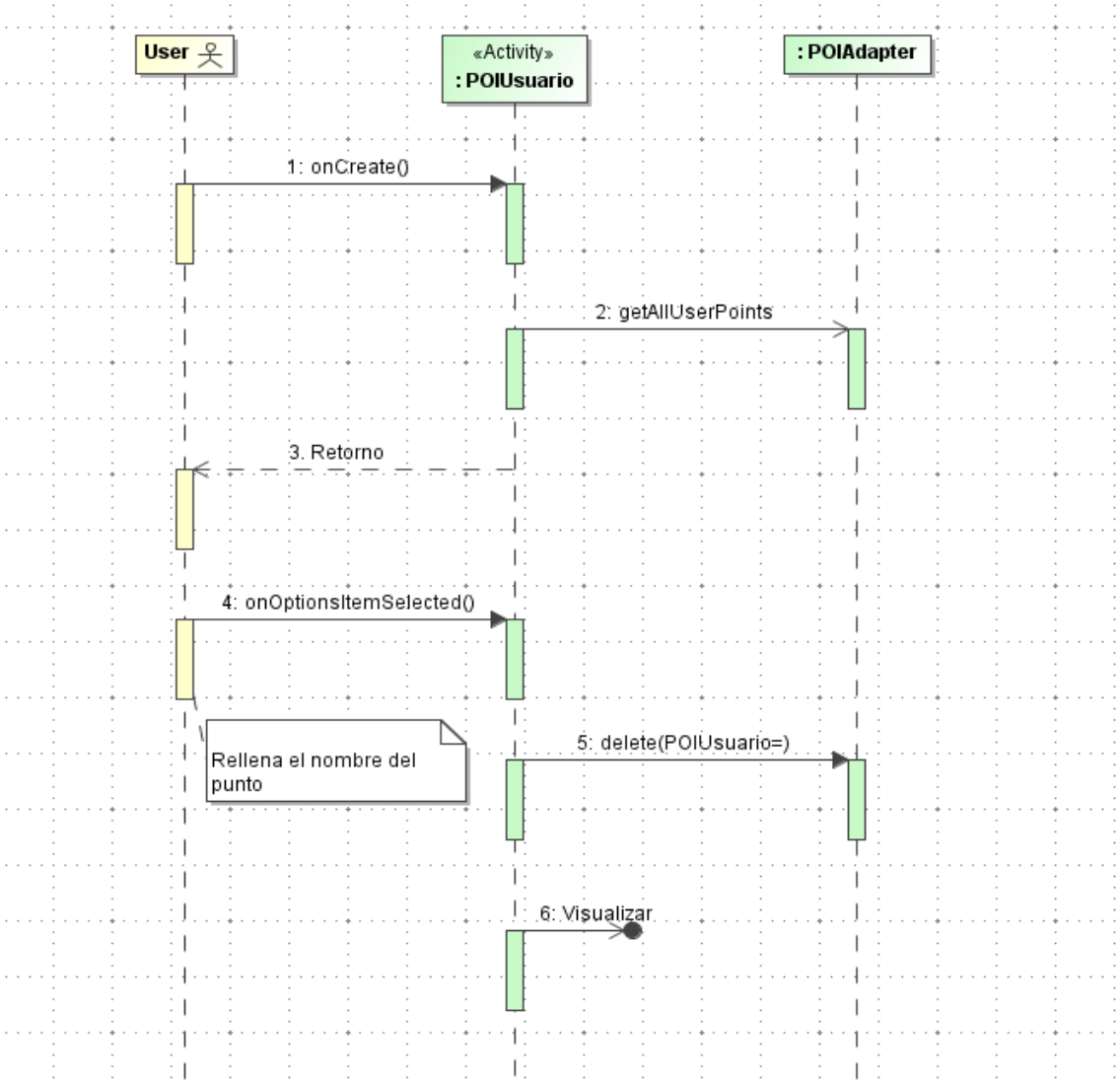


Ilustración 13 - Borrar punto de usuario

7.4 Diseño del modelo de datos

Con el fin de almacenar los datos que manejaremos en el aplicativo, se hace necesario un elemento que nos permita guardar estos datos y recuperarlos de forma ágil cuando sea necesario. Este elemento es el Sistema Gestor de Bases Datos.

A continuación, abordaremos la creación de un modelo de datos que nos permita dar respuesta a las necesidades de gestión de datos de este PFC.

Según Wikipedia, "*un modelo de datos es un lenguaje utilizado para la descripción de una base de datos. Por lo general, un modelo de datos permite describir las estructuras de datos de la base (el tipo de los datos que incluye la base y la forma en que se relacionan), las restricciones de integridad (las condiciones que los datos deben cumplir para reflejar correctamente la realidad deseada) y las operaciones de manipulación de los datos (agregado, borrado, modificación y recuperación de los datos de la base).*"

En el caso que nos ocupa se ha considerado suficiente el siguiente modelo.

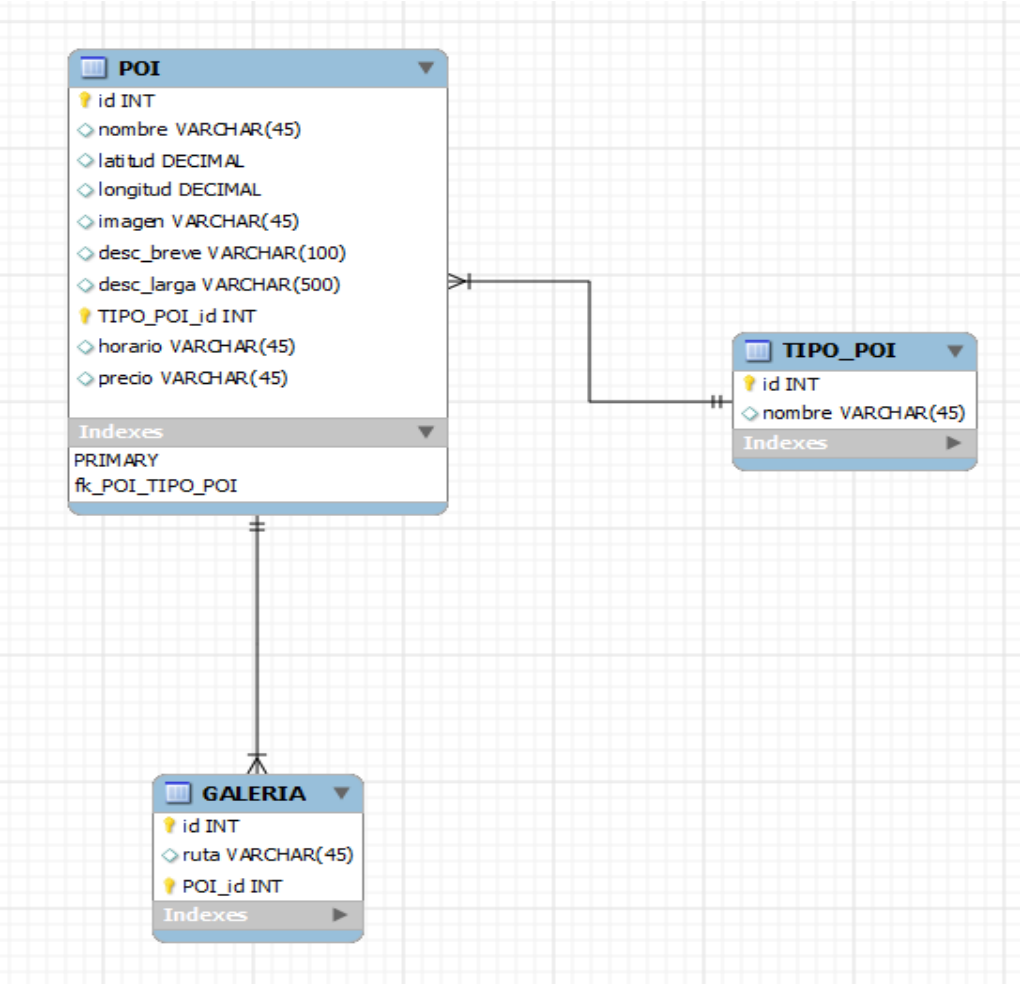


Ilustración 14 - Modelo de datos

Como puede observarse, es un modelo de datos muy simple pero que, al mismo tiempo, permite dar respuesta a nuestras necesidades.

A continuación procedemos a realizar una breve explicación de cada una de las tablas que componen el modelo.

TIPO_POI

Es una tabla maestra, compuesta por un código y una descripción. Almacena los distintos tipos de puntos que puede almacenar la aplicación (monumentos, museos, iglesias, etc). Con este mecanismo conseguimos que la aplicación esté preparada para poder albergar los diferentes tipos de puntos que se consideren necesarios.

POI

Almacena los puntos de interés de la aplicación. Contiene todos sus datos generales y está relacionada con TIPO_POI.

GALERIA

Almacena las rutas a las imágenes que pertenecen a un punto de interés. Los puntos de interés pueden tener asociada una galería de imágenes representativas. Cada imagen puede pertenecer a un único punto de interés. Se observa la relación existente en la ilustración anterior.

8. Interface de Usuario

Una vez han sido definidos y analizados los requisitos del proyecto, e introducida la arquitectura, comenzaremos con el desarrollo de la interfaz gráfica de usuario.

De acuerdo con el manifiesto ágil de desarrollo de software, es deseable "*poder ver anticipadamente cómo se comportan las funcionalidades esperadas sobre prototipos o sobre las partes ya elaboradas del sistema final ofrece una retroalimentación (feedback) muy estimulante y enriquecedor que genera ideas imposibles de concebir en un primer momento; difícilmente se podrá conseguir un documento que contenga requisitos detallados antes de comenzar el proyecto.*"

Es por esta razón que abordaremos la creación de la interfaz de usuario antes de comenzar la fase de implementación propiamente dicha.

Se comenzará el desarrollo de la interfaz explicando las líneas generales para su diseño e introduciendo conceptos sobre Android, necesarios para la compresión del proceso seguido.

8.1 Diseño de la interfaz gráfica de usuario

En el diseño de la interfaz gráfica de usuario se ha prestado especial atención a tres puntos fundamentales:

- Estética visual
- Comodidad de uso
- Independencia del dispositivo

Estética visual

En la actualidad, la estética de una aplicación resulta determinante para el éxito o fracaso de la misma. Se ha intentado dar un aspecto en los iconos del aplicativo e introducir algunos efectos visualmente atractivos.

Comodidad de uso

Una aplicación destinada a terminales móviles de estar pensada para que el usuario pueda llegar fácilmente a una funcionalidad, intentado que realice el menor número de acciones posible. Durante el desarrollo de esta guía, se ha realizado un esfuerzo para mostrar en una única vista la mayor información posible, de una forma cómoda e intuitiva.

Independencia del dispositivo

Tal y como sucede en las aplicaciones tradicionales, en el mundo de las aplicaciones para móviles no existe un estándar de pantalla de visualización. A este inconveniente hay que añadirle el reducido tamaño de las pantallas de los terminales. Por estos motivos se han utilizado técnicas de posicionamiento que se adaptan tanto como sea posible al tamaño de la pantalla de visualización del terminal utilizado.

8.2 Introducción a *Views* y *Layouts* en Android

En Android, las interfaces de usuario se construyen utilizando ***vistas, grupos de vistas y Layouts***. Existen muchos tipos de vistas, todas ellas tendrán como clase padre a *View.class*.

Los objetos *View* son las unidades básicas del interface de usuario de Android , ofrecen un marco en el que pueden "pegarse" los diferentes elementos de ordenación (*Layouts*) o visuales (*Widgets*), adicionalmente, son los encargados de realizar una primera detección sobre las acciones de usuario (pulsaciones, desplazamientos …).

Cada *Activity* (pantalla funcional) Android debe tener asociada al menos una *View*. Cuando una actividad Android recibe el foco, el sistema analiza la *View*, como si de un árbol se tratara, y dibuja todos los elementos pegados en la *View*, así como los subelementos que estos últimos puedan contener.

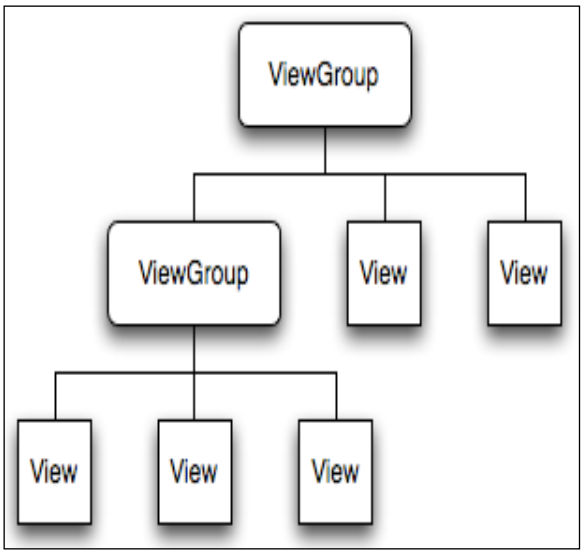


Ilustración 15 - Esquema jerárquico de View

Los *Layouts* son los elementos de posicionamiento que permiten ubicar y ordenar los diferentes elementos en pantalla. Existe diferentes tipos de *Layouts*, según la funcionalidad y el tipo de ordenación que deseemos utilizar. Los *layouts* pueden combinarse entre sí tantas veces como sea necesario, de forma absoluta o relativa a otros *layouts*.

A continuación explicaremos cómo se han configurado los *Layouts* para obtener las visualizaciones de este proyecto.

8.3 Mapa de Situación

El mapa de situación es uno de los elementos centrales de la aplicación. Como hemos explicado en apartados anteriores, es un mapa de posición dónde se ubican tanto la posición del usuario como la posición de los puntos de interés. El Mapa de Situación tendrá el aspecto que se muestra a continuación:

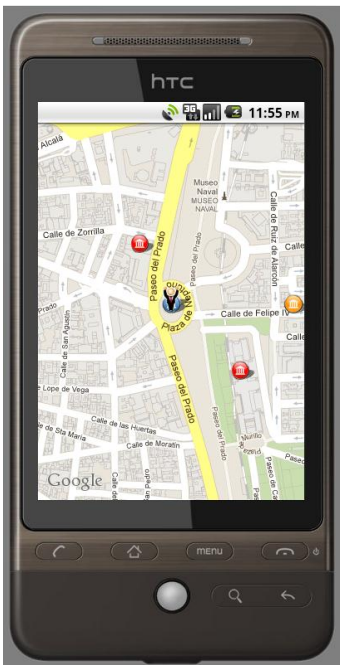


Ilustración 16 - Mapa de situación

Cómo se observa en la figura, aparecen claramente diferenciadas la posición del usuario (icono con forma humana) y la de los puntos de interés (icono circular de edificio). Cabe resaltar que, para comodidad del usuario, se representan los distintos tipos de puntos de diferentes colores.

Para esta actividad también ha sido necesario diseñar un mensaje *ad hoc* con el fin de mostrar información breve y opciones sobre un punto de interés.

Ilustración 17 - Información abreviada

Este efecto se consigue con la combinación de *Layouts* que se indica en la siguiente figura:

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:id="@+id/layout_root"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:padding="10dp"
              >
    <ImageView android:id="@+id/image"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:layout_marginRight="10dp"
               />
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:textColor="#FFF"
              />
</LinearLayout>
```

Ilustración 18 - *Layout* de ventana emergente

La botonera se añadirá mediante código situado fuera de los *layouts*.

8.4 Calculo de ruta

En el diseño del cálculo de ruta se ha tenido en cuenta el dinamismo de la operación. No sabemos de antemano qué número de elementos vamos a tener que mostrar, por lo que hemos juzgado interesante utilizar un *Layout* capaz de admitir un número indeterminado de elementos. El *Layout* utilizado para esta actividad ha sido el ***WebView***. Este *Layout* permite añadir elementos tal y como si se tratara de una página HTML.

El efecto conseguido es el mostrado en la siguiente ilustración:

Ilustración 19 - Calculo de ruta

El código utilizado para la maquetación de la actividad puede observarse en la siguiente ilustración:

PlacesItemizedOverla | MapaSituacion.java | info_previa.xml | ruta.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

    <WebView
        android:id="@+id/webview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    />
</LinearLayout>
```

Ilustración 20 - Utilización del WebView

8.5 Listado de puntos de interés

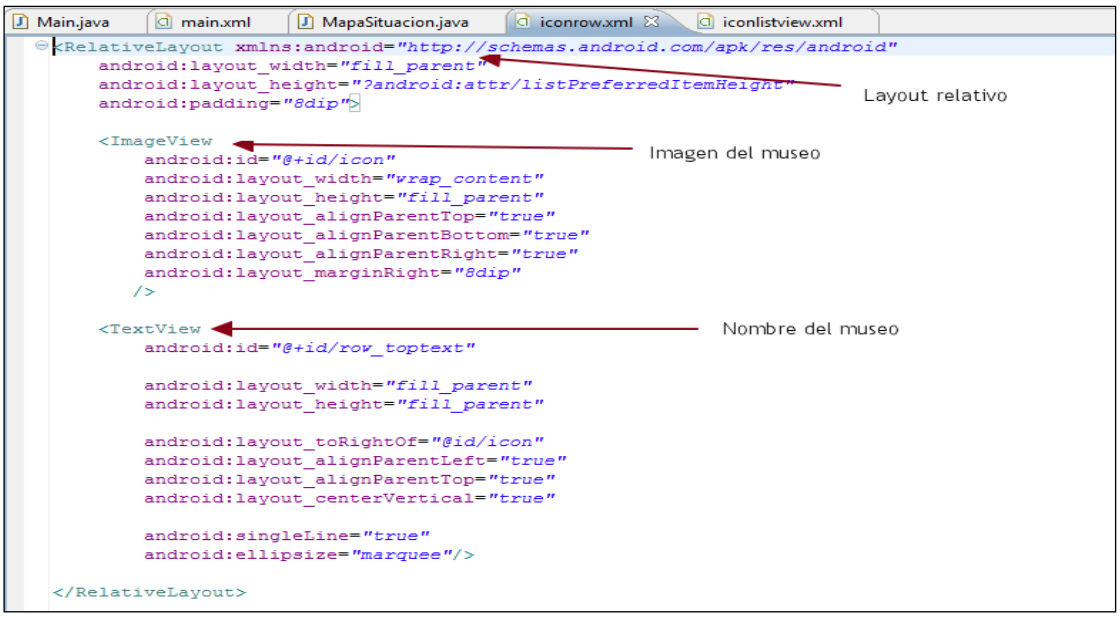
Los puntos de interés catalogados se presentan en forma de lista. Esta forma de presentación permite navegar de forma cómoda por cada uno de ellos.

Cada elemento de la lista se compone de:

- Imagen representativa del punto interés.
- Nombre del punto interés.

Ilustración 21 - Listado de puntos catalogados

Para conseguir el efecto de lista se ha utilizado la combinación de dos configuraciones de *Layout*. Cada icono de la lista se configura de la siguiente forma:




```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="8dip">

    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentTop="true"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_marginRight="8dip"
       />

    <TextView
        android:id="@+id/row_toptext"

        android:layout_width="fill_parent"
        android:layout_height="fill_parent"

        android:layout_toRightOf="@id/icon"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_centerVertical="true"

        android:singleLine="true"
        android:ellipsize="marquee"/>

</RelativeLayout>
```

Ilustración 22 - Configuración de elemento de una lista

Es preciso observar que se ha utilizado un ***RelativeLayout*** como *Layout* principal. El *RelativeLayout* nos permite definir unas reglas predeterminadas para que el elemento se auto-ubique y reordene junto los otros *layouts* que se utilizan dentro de una actividad.

Una vez definido cada elemento de la lista, utilizaremos otro *Layout* para definir cómo tienen que relacionarse los ítems entre sí. El comportamiento de la lista se define en la siguiente figura.


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

    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/settings"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <ListView
            android:id="@+id/android:list"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            />

    </LinearLayout>
```

Ilustración 23 - Configuración de la lista

Esta configuración de la lista implica que el contenedor va a orientarse de forma vertical, es decir, los elementos que se vayan añadiendo se apilarán uno sobre otro. Los elementos de la lista se ajustarán al tamaño imprescindible para poder visualizarse, pero la lista se ajustará completamente al espacio de visualización del dispositivo.

El listado incluye un *Scroll* para poder visualizar todos los elementos aunque estos excedan el área de visualización del dispositivo.

Ilustración 24 - Detalle de Scroll

8.6 Detalle de punto de interés

La actividad que muestra el detalle de los puntos de interés catalogados está configurada de la forma que se muestra a continuación:

Ilustración 25 - Descripción del punto

Puede observarse que se ha dividido el área de visualización en tres zonas dónde se ha ubicado una imagen de presentación, el punto y su nombre, y una zona central y de mayor tamaño, donde se ofrece información en profundidad sobre el punto elegido. La zona central cuenta con un *scroll* que permite visualizar la información al completo aunque el texto exceda de las dimensiones de la pantalla.

Este efecto ha sido logrado con la siguiente combinación de *layouts*:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:orientation="vertical">
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="left">
        <ImageView
            android:id="@+id/icon"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_alignParentBottom="true"
            android:layout_alignParentRight="true"
            android:layout_marginRight="8dip"
        />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@id/icon"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:layout_centerVertical="true"
            android:textStyle="bold"
            android:textSize="14dip"
            android:id="@+id/titulo"
        />
    </LinearLayout>
    <ScrollView android:scrollbars="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_marginTop="1dip">
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="left"
            android:textColor="#ffffff"
            android:isScrollContainer="true"
            android:id="@+id/cuerpo"/>
    </ScrollView>
</LinearLayout>
```

Imagen

Título

El Scroll contiene la zona principal de información

Ilustración 26 - *Layout* detalle del punto de interés

Adicionalmente, esta actividad ha sido dotada de un menú contextual que permite al usuario realizar operaciones relacionadas con el punto de interés que está visualizando.

Ilustración 27 - Descripción del punto - Menú contextual

8.7 Galería

La galería muestra imágenes relacionadas con el punto de interés. A la hora de diseñar la actividad, se ha tenido en cuenta:

- La posibilidad de mostrar varias imágenes.
- La comodidad el usuario para cambiar de una a otra.

Teniendo en cuenta estos requisitos de diseño, se ha optado por dividir la pantalla en dos partes, tal y como muestran las figuras siguientes:

Ilustración 28 - Galería de imágenes

Como puede observarse, en la parte superior aparece una barra de pre visualizaciones con todas las imágenes disponibles. El usuario puede desplazarse entre ellas utilizando un gesto de desplazamiento. Una vez se llega a la imagen que se quiere visualizar, se pulsa en ella y esta imagen se carga en la parte central de la pantalla.

Este efecto se consigue mediante la combinación de los *layouts Gallery* y *LinearLayout*, tal y como se muestra en el siguiente fragmento de código:


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LinearLayout01"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:gravity="left"
    >
    <Gallery xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/galeriaobras"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="left"
    />
        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center">
            <ImageView android:id="@+id/ImageView01"
                android:layout_width="wrap_content" android:layout_height="wrap_content" />
        </LinearLayout>

</LinearLayout>
```

Ilustración 29 - Galería de imágenes

8.8 Gestión del punto de usuario

Con el fin de facilitar la gestión de puntos de usuario, se ha integrado completamente en una única acción. La gestión de puntos de usuario se compondrá de un formulario simple, que servirá tanto para dar de alta como para dar de baja un punto definido por el usuario.

El interfaz presentará el aspecto que se muestra en la siguiente ilustración.

Ilustración 30 - Gestión de puntos de usuario

Este efecto se ha logrado mediante la combinación de *Layouts* que se muestra a continuación:

DescripcionPOI.java | POIAdapter.java | Menus.java | PasoParametro.java | ListadoPOIs.java | Galeria.java | PlacesItemizedOverlay.j | MapaSituacion.java | GeoUpdateHandler.jav

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent" android:weightSum="1">

  <TextView
     android:id="@+id/ges_poi_nombre"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="Nombre Punto:"
     />
<AutoCompleteTextView android:layout_height="wrap_content" android:id="@+id/poi_usuario_nombre_auto" android:text="" android:layout_width="match_parent">
    <requestFocus></requestFocus>
</AutoCompleteTextView>

  <TextView
     android:id="@+id/ges_poi_descripcion"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="Descripcion:"
     />
<EditText android:layout_height="wrap_content" android:layout_width="match_parent" android:id="@+id/poi_usuario_desc" android:inputType="textMultiLine">
</EditText>

  <TextView
     android:id="@+id/ges_poi_previsualizacion"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="Previsualizacion:"
     />
<ImageView android:layout_height="wrap_content" android:layout_width="wrap_content" android:id="@+id/poi_previsualizar"></ImageView>

</LinearLayout>
```

Titulo

Campo autocompletable

Descripción

Imagen asociada al punto

Ilustración 31 - Layout de punto de usuario

Además, se ha añadido un menú contextual que permite manejar cómodamente toda la funcionalidad de gestión desde una única actividad Android. Puede verse la configuración de este menú contextual en la siguiente figura:

Ilustración 32 – Menú contextual gestión de punto de usuario

A través del menú contextual podremos acceder tanto a la galería del terminal, como a la cámara fotográfica del mismo.

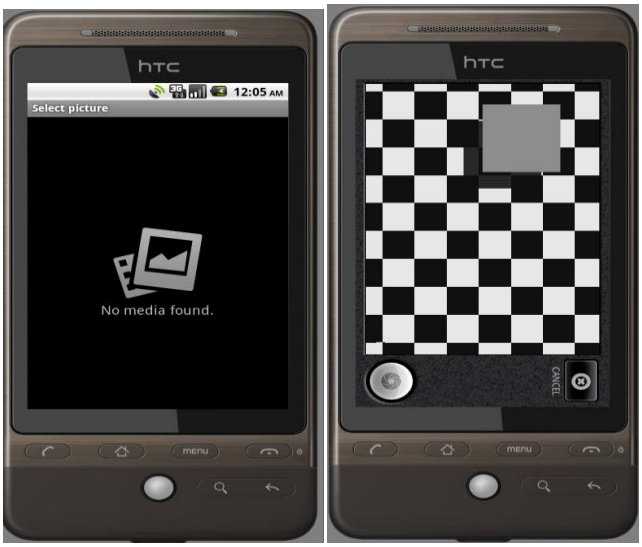


Ilustración 33 – Galería y cámara fotográfica

9. Plan de pruebas funcionales

Una vez terminado el software, surge la necesidad de probarlo. Un plan de pruebas es un documento guía que nos ayuda a garantizar el cumplimiento del software entregado con los requerimientos expresados por el cliente.

El plan de pruebas se compone de una serie de pruebas funcionales en las que, tras realizar unas acciones, se obtiene un resultado esperado. El plan debería estar compuesto, si no por todas las funcionalidades previstas en la aplicación, sí por las más importantes o críticas de cara al usuario.

El plan que se expone a continuación está compuesto por la prueba de verificación de cada una de las actividades representadas en el apartado ***7.3 Diagramas de solución técnica***.

Cada apartado contiene las acciones que deberá ir realizando el usuario para probar la funcionalidad, así como la respuesta esperada del sistema.

9.1 Emulador

Aunque sería deseable disponer de un terminal real para la ejecución del plan de pruebas, somos conscientes de la imposibilidad de contar con este terminal en algunas ocasiones. Por ello se ha previsto la ejecución del plan en un emulador. Los ***AVDs*** (Android VIRTUAL DEVICES) son emuladores de terminal Android que permiten la ejecución de aplicaciones Android sobre un PC. Es software de uso libre, pudiendo conseguirse fácilmente en internet. Además presenta múltiples *Skins*, lo que permite emular un desarrollo para un terminal concreto.

Estas pruebas se han diseñado utilizando un emulador instalado sobre el entorno de desarrollo Eclipse. La explicación detallada de la instalación y configuración del AVD está contenida en el documento ***Anexo IV – Configuración del entorno de desarrollo***.

Configuración de posición

Dado que el sistema tiene capacidades de SIG (Sistema de Información Geográfica, véase Anexo I), será necesario simular que el usuario se encuentra en una ubicación geográfica concreta con el fin de poder probar las funcionalidades de geolocalización y cálculo de rutas. Esta configuración será similar a la mostrada en la siguiente ilustración.

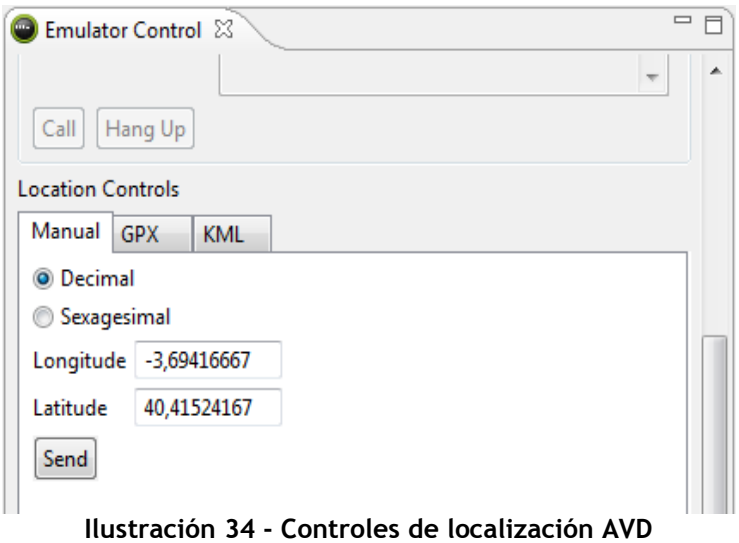


Ilustración 34 - Controles de localización AVD

9.2 Mapa de Situación

Usuario	Sistema
1. Pulsar en el menú contextual sobre “Mi Posición”. 	2. Aparecerá un mapa ubicando un icono de persona (representando al usuario) en la posición geográfica en la que se encuentra actualmente el usuario. En el mapa estarán ubicados los puntos de interés, representados en diferentes colores en función del tipo el que se trate. 
3. Pulsar sobre un punto de interés.	4. El sistema mostrará una ventana emergente, compuesta de una imagen significativa del punto y una descripción breve. En la parte inferior de la ventana aparecerá una botonera con las opciones de “llegar”, “+info” y “cancelar”. 

9.3 Listado de POIs

Usuario	Sistema
1. Pulsar en el menú contextual sobre	2. El sistema mostrará un listado de puntos

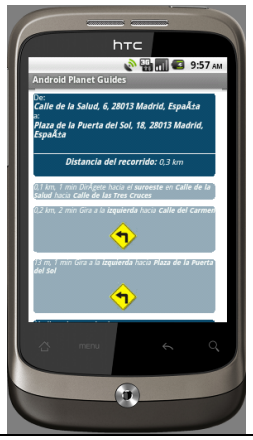
“Puntos de interés”. 	de interés. Cada elemento del listado estará compuesto por una imagen descriptiva y el nombre del punto de interés. La actividad es sensitiva a la posición del teléfono, de esta forma se repintará si giramos el terminal. 

9.4 Calculo de Ruta

Existen dos formas de ejecutar un cálculo de ruta. Es necesario realizar las dos comprobaciones para verificar que el software funciona correctamente.

Forma 1:

Usuario	Sistema
1. Pulsar en el menú contextual sobre “Mi Posición”. 	2. Aparecerá un mapa ubicando un icono de persona (representando al usuario) en la posición geográfica en la que se encuentra actualmente el usuario. En el mapa estarán ubicados los puntos de interés, representados en diferentes colores en función del tipo que se trate. 
3. Pulsar sobre un punto de interés.	4. El sistema mostrará una ventana emergente, compuesta de una imagen significativa del punto y una descripción breve. En la parte inferior de la ventana

	aparecerá una botonera con las opciones de “llegar”, “+info” y “cancelar”. 
5. Pulsar sobre “Llegar”. 	6. El sistema mostrará la ruta que debemos seguir para llegar desde la ubicación del usuario al punto seleccionado. La ruta estará compuesta de: a) Una cabecera que indicará direcciones de origen y destino, tiempo estimado de realización de la ruta y distancia del recorrido. b) Una lista de pasos a realizar para completar el recorrido. Esta lista tiene flechas indicativas de los cambios de dirección. 

Forma 2:

Usuario	Sistema
1. Pulsar en el menú contextual sobre “Puntos de interés”.	2. El sistema mostrará un listado de puntos de interés. Cada elemento del listado estará compuesto por una imagen descriptiva y el nombre del punto de interés. La actividad es sensitiva a la posición del teléfono, de esta forma se repintará si giramos el terminal.

	
3. Pulsar sobre cualquiera de los puntos de interés.	4. El sistema mostrará una pantalla con información detallada sobre el punto de interés seleccionado. 
5. En el menú contextual pulsamos la opción “Cómo llegar”. 	6. El sistema mostrará la ruta que debemos seguir para llegar desde la ubicación del usuario al punto seleccionado. La ruta estará compuesta de: c) Una cabecera que indicará direcciones de origen y destino, tiempo estimado de realización de la ruta y distancia del recorrido. d) Una lista de pasos que deberemos realizar para completar el recorrido. Esta lista tiene flechas indicativas de los cambios de dirección.

9.5 Consultar un POI

Forma 1:

Usuario	Sistema
1. Pulsar en el menú contextual sobre "Puntos de interés". 	2. El sistema mostrará un listado de puntos de interés. Cada elemento del listado estará compuesto por una imagen descriptiva y el nombre del punto de interés. La actividad es sensitiva a la posición del teléfono, de esta forma se repintará si giramos el terminal. 
3. Pulsar sobre cualquiera de los puntos de interés.	4. El sistema mostrará una pantalla con información detallada sobre el punto de interés seleccionado. 

Forma 2:

Usuario	Sistema
1. Pulsar en el menú contextual sobre "Mi Posición".	2. Aparecerá un mapa ubicando un icono de persona (representando al usuario) en la

	posición geográfica en la que se encuentra actualmente el usuario. En el mapa estarán ubicados los puntos de interés, representados en diferentes colores en función del tipo el que se trate. 
3. Pulsar sobre cualquiera de los puntos de interés.	4. Aparece una ventana de diálogo con varias opciones. 
5.Pulsamos “+Info”	6. Aparece la información completa referente al punto seleccionado. 

9.6 Consultar una Galería

Usuario	Sistema
1. Pulsar en el menú contextual sobre la opción “Puntos de interés”. 	2. El sistema mostrará un listado de puntos de interés. Cada elemento del listado estará compuesto por una imagen descriptiva y el nombre del punto de interés. La actividad es sensitiva a la posición del teléfono, de esta forma se repintará si giramos el terminal. 
3. Pulsar sobre cualquiera de los puntos de interés.	4. El sistema mostrará una pantalla con información detallada sobre el punto de interés seleccionado. 
5. Pulsar sobre “Galería” en el menú contextual. 	6. El sistema mostrará una pantalla fragmentada en dos parte: a) En la parte superior se observarán pre visualizaciones de las imágenes que componen la galería. Puede desplazarse esta barra de pre visualización realizando el gestor correspondiente. b) Si se pulsa sobre una imagen, aparecerá cargada en la parte inferior de la pantalla.

9.7 Consultar información de servicio

Usuario	Sistema
1. Pulsar en el menú contextual sobre la opción “Puntos de interés”. 	2. El sistema mostrará un listado de puntos de interés. Cada elemento del listado estará compuesto por una imagen descriptiva y el nombre del punto de interés. La actividad es sensitiva a la posición del teléfono, de esta forma se repintará si giramos el terminal. 
3. Pulsar sobre cualquiera de los puntos de interés.	4. El sistema mostrará una pantalla con información detallada sobre el punto de interés seleccionado. 
5. Pulsar sobre “Información” en el menú contextual.	6.El sistema mostrará una pantalla con información de servicio del punto de interés

	(en caso de tenerla). 

9.8 Insertar POI de Usuario

Debido a limitaciones en el emulador (uso de cámara fotográfica y galería), se recomienda que este puntos se verifique en un terminal Android real.

Usuario	**Sistema**
1. Pulsar en el menú contextual sobre la opción “Mis puntos favoritos”. 	2. El sistema mostrará un formulario compuesto de 2 campos. Un nombre para el punto y una descripción. 
3. El usuario rellena el nombre del punto y la descripción. El campo “Nombre” es un campo autocompletado y se completa con los nombres de puntos de usuario existentes en BBDD. El nombre insertado no podrá coincidir con uno ya existente. La asociación de una imagen a un punto de usuario puede realizar se dos formas: a) Fotografiando el punto. b) Seleccionando una imagen de la galería.	4. Forma 1: El sistema mostrará la cámara fotográfica del terminal. El usuario tomará la fotografía del punto y esta se mostrará en la zona de pre visualización del formulario.

Forma 1: Tomar fotografía

Forma 2: Seleccionar desde galería

5. El usuario pulsa el botón "Guardar punto" del menú contextual.

Forma 2:

El sistema mostrará la galería fotográfica del terminal, permitiendo al usuario seleccionar una imagen. La imagen seleccionada se cargará en la zona de pre visualización del formulario.

6. El sistema da de alta el punto de usuario con su imagen asociada.

9.9 Eliminar POI de Usuario

Debido a limitaciones en el emulador (uso de cámara fotográfica y galería), se recomienda que este puntos se verifique en un terminal Android real.

Usuario	**Sistema**
1. Pulsar en el menú contextual sobre la opción “Mis puntos favoritos”. 	2. El sistema mostrará un formulario compuesto de 2 campos: Un nombre para el punto y una descripción. 
3. El usuario introducirá el nombre (o parte de él) del punto que desea eliminar.	4. El sistema autocompletará el nombre. Aparecerá cargada la descripción y la imagen relacionada con el punto.

9.10 Ayuda

Usuario	Sistema
1. Pulsar en el menú contextual sobre la opción “Ayudas”. 	2. El sistema mostrará una pantalla con un texto de ayuda. 

10. Implementación

En este apartado procederemos a explicar todos los detalles concernientes a la implementación del software.

En primer lugar explicaremos cómo se ha configurado el entorno de desarrollo para llevar a cabo el proyecto. Nos detendremos especialmente en los detalles de integración entre las diferentes herramientas utilizadas.

Una vez dispuesto el entorno de desarrollo, procederemos a explicar las soluciones técnicas más significativas. Para el correcto seguimiento de las mismas es recomendable consultar los documentos ***Anexo I*** y ***Anexo II***, en los que se resolverán dudas terminológicas y se explicarán las tecnologías intervinientes en la implementación.

El código fuente mostrado en este capítulo puede ser consultado en su totalidad en el ***Anexo VI – Código***.

10.1 Entorno de desarrollo

Como entorno base de desarrollo se ha utilizado Eclipse. Este IDE (Integrated Development Environment) es una herramienta de uso libre, comúnmente utilizada para el desarrollo en varios lenguajes, entre ellos Java. Además, esta herramienta cuenta con plugins especialmente pensados para el desarrollo en Android.

Todos los detalles de la preparación del entorno se han incluido en el ***Anexo IV - Configuración del entorno de Desarrollo***.

10.2 Soluciones técnicas

Manejo del GPS del terminal

En la actualidad, gran parte de los terminales móviles disponibles en el mercado traen incorporado un sistema GPS que puede ser utilizado desde el Sistema Operativo del terminal.

En este proyecto se utiliza el GPS del terminal para obtener la localización de usuario de cara a la implementación de dos funcionalidades:

- Mostrar la posición actual del usuario
- Calcular la ruta desde la posición actual a un punto de destino

Para implementar esta funcionalidad, Android nos ofrece el interfaz ***LocationListener*** y la clase ***LocationManager***. Implementando ***LocationListener***, Android notificará las actualizaciones de posición, mientras que por medio de ***LocationManager*** podremos recuperar las coordenadas actuales.

Para gestionar estas funcionalidades hemos creado en nuestro proyecto la clase ***GeoUpdateHandler***. Esta clase implementa ***LocationListener*** y es notificada por Android ante cambios en el estado del GPS, además, la hemos dotado del método ***getCurrentLocation***, que nos devuelve las últimas coordenadas registradas por el terminal.

```
public void onProviderDisabled(String arg0) {
    Toast.makeText(mContext,"GPS Desactivado", Toast.LENGTH_SHORT).show();
}

public void onProviderEnabled(String arg0) {
    Toast.makeText(mContext,"GPS Activado", Toast.LENGTH_SHORT).show();

}

public void onStatusChanged(String arg0, int arg1, Bundle arg2) {
    Toast.makeText(mContext,"GPS Cambio de estado", Toast.LENGTH_SHORT).show();

}

/**
 * Devuelve la localización actual en coordenadas geográficas
 * @return Location
 */
public Location getCurrentLocation(){
     //Calculo posición actual
    LocationManager locationManager = (LocationManager) actividad.getSystemService(Context.LOCATION_SERVICE);

  //Posición actual
    Criteria criteria=new Criteria();
    String bestProvider=locationManager.getBestProvider(criteria, true);
    Location loc=locationManager.getLastKnownLocation(bestProvider);

    return loc;
}
```

Ilustración 35 - Métodos de GeoUpdateHandler

El método más relevante de ***GeoUpdateHandler*** es el método ***onLocationChanged***. Este es el método que reescribiremos para repintar los ítems del mapa en función de los cambios de posición del usuario. En el siguiente apartado, *Manejo de Map Views*, se mostrará parte del código necesario para el repintado. Puede consultar el código íntegramente en el ***Anexo VI***.

Manejo de MapViews

Un ***MapView*** es una clase que representa un mapa al que pueden añadírsele diferentes controles, que nos permiten ejercer acciones sobre él, así como diferentes capas, que nos permiten representar elementos en el mapa.

Ilustración 36 - Mapa con elementos añadidos

La funcionalidad (***MapaSituacion***) que localiza al usuario y a los puntos de interés de la aplicación sobre un mapa, está implementada utilizando una actividad Android que extiende la clase ***MapView***. La implementación consta de dos partes, por un lado es necesario crear el Layout de la actividad y situar en el ***MapView***.

GeoUpdateHandler.java | mapa_situacion.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mapView"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <com.google.android.maps.MapView
        android:id="@+id/mapview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"
        android:apiKey="0-cw0fiGv77pTNEIRPg_3wDQX1MnzZhjgQ8mgFA"
    />

</LinearLayout>
```

Ilustración 37 - MapView con API Key de Google Maps

Nótese que una de las propiedades del ***MapView*** es el Maps API Key (véase ***Anexo IV - Configuración del Entorno de Desarrollo***), sin esta clave los mapas no estarán operativos.

Por otro lado está la clase asociada, ***MapaSituacion***, que define el ciclo de vida del ***MapView***, así como algunas de sus características.

```
public void onCreate(Bundle savedInstanceState){
    Log.i("Guia","Creando MapaSituacion");
    super.onCreate(savedInstanceState);
    setContentView(R.layout.mapa_situacion);

    mapView=(MapView)findViewById(R.id.mapview);
    mapView.displayZoomControls(true);
    mapOverlays=mapView.getOverlays();
    mapController = mapView.getController();
    mapController.setZoom(17);

    updatePosition();

}
```

Ilustración 38 - Establecimiento de atributos de MapView

Los diferentes puntos que ubicaremos en el mapa están representados por las clases ***OverlayItem e ItemizedOverlay***. Estas clases nos permiten crear puntos personalizados a nivel de imagen y comportamiento. En la aplicación existen, básicamente, dos tipos de puntos: el punto de localización de usuario y el punto de interés.

Por un lado, el punto de localización de usuario muestra un icono de figura humana que representa la posición en la que se encuentra actualmente el usuario (o la última lectura del terminal GPS). Para implementar este tipo de puntos se ha creado la clase ***UserItemizedOverlay***. Esta clase extiende de ***ItemizedOverlay*** y, por un lado, nos permite establecer qué icono queremos mostrar asociado al punto y por otro, la acción que se desea realizar al pulsar sobre el punto. A continuación se muestra el método ***onTap***, encargado de la acción a realizar si se pulsa en el icono que representa al punto.

```
@Override
protected boolean onTap(int index) {
    OverlayItem item = mOverlays.get(index);
    AlertDialog.Builder dialog = new AlertDialog.Builder(mContext);
    dialog.setTitle(item.getTitle());
    dialog.setMessage(calle);
    dialog.setIcon(es.guia.interfaz.R.drawable.icon);
    dialog.show();


  return true;
}
```

Ilustración 39 - Método onTap de UserItemizedOverlay

Por otro lado, el punto de interés presenta varios tipos de comportamiento: en función del tipo de punto del que se trate, el icono del punto se mostrará en un color u otro, esto permitirá al usuario distinguir visualmente la temática del punto antes de pulsar sobre él. Además, al pulsarlo, se mostrará una ventana emergente con una fotografía del punto, una descripción breve y varias posibilidades (Llegar, +info, Cancelar), este comportamiento se obtiene con el código que se muestra en la siguiente ilustración.

Ilustración 40 - Detalle del comportamiento de punto de interés

Para implementar este comportamiento se han utilizado las clases ***PlacesOverlay***, ***PlacesItemizedOverlay*** y, como ***inner class*** en esta última, la clase ***OpcionesHandler***.

La clase principal de este conjunto es ***PlacesItemizedOverlay***, en ella vamos a definir el icono representativo del punto y el comportamiento en pulsación. A continuación comentaremos el método ***onTap*** y como se ha implementado el comportamiento mostrado en la ilustración 40.

```
@Override
protected boolean onTap(int index) {
    PlacesOverlay item =(PlacesOverlay) mOverlays.get(index);
    POI poi=item.getPOI();

    AlertDialog.Builder builder;
    AlertDialog alertDialog;


    LayoutInflater inflater = (LayoutInflater) mContext.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);
    View layout = inflater.inflate(R.layout.info_previa,
                                   (ViewGroup) actividad.findViewById(R.id.layout_root));


    TextView text = (TextView) layout.findViewById(R.id.text);
    text.setText(poi.getDesc_breve());

    ImageView image = (ImageView) layout.findViewById(R.id.image);
    image.setImageResource(Menus.hImagenes.get(poi.getImagen()));

    builder = new AlertDialog.Builder(mContext);
    builder.setView(layout);

    //Manejador de opciones

    OpcionesHandler oHandler=new OpcionesHandler(poi,actividad);

    builder.setPositiveButton("Llegar",oHandler);
    builder.setNeutralButton("+Info",oHandler);
    builder.setNegativeButton("Cancelar",oHandler);

    alertDialog = builder.create();
    alertDialog.setTitle(poi.getNombre());
    alertDialog.show();


  return true;
}
```

Ilustración 41 - Método onTap de PlacesItemizedOverlay

En primer lugar se recupera el punto que el usuario ha pulsado del conjunto de puntos distribuidos en el mapa. Android trata los puntos distribuidos en el mapa como una lista

numerada de puntos, de esta forma, podemos recuperar las características de un punto en base a la posición que ocupa en esta lista.

Una vez recuperado el punto pulsado, pedimos al sistema el ***LAYOUT_INFLATER_SERVICE***. Este ***Layout***, predefinido en el sistema, nos va a permitir mostrar la ventana emergente con una imagen y un texto descriptivo. Puede observarse en el código como establecemos mediante los métodos ***setText*** y ***setImageResource*** el texto que queremos mostrar y su imagen asociada.

El efecto de los 3 botones se logra mediante la ***inner class OpcionesHandler***, cuyo código comentamos a continuación.

```
class OpcionesHandler implements DialogInterface.OnClickListener{

    public static final int LLEGAR=-1;
    public static final int CANCEL=-2;
    public static final int INFO=-3;


    private POI poi;
    private Activity actividad;

    public OpcionesHandler(POI poi, Activity actividad){
        super();
        this.poi=poi;
        this.actividad=actividad;

    }

    public void onClick(DialogInterface dialog, int id) {

        Log.d("Id: ",String.valueOf(id));

        switch(id){
            case LLEGAR:
                Log.d("Longitud: ",poi.getLongitud());
                Log.d("Latitud: ",poi.getLatitud());

                Intent intento=new Intent(actividad, CalculoRuta.class);

                intento.putExtra("longitudDestino", poi.getLongitud());
                intento.putExtra("latitudDestino", poi.getLatitud());

                actividad.startActivity(intento);

                break;
            case INFO:
                Log.d("+Info","Mas informacion");
                break;
            default:
                dialog.cancel();
        }

    }
```

Ilustración 42 - Código de la clase OpcionesHandler

Puede observarse fácilmente cómo se definen 3 botones y, mediante la interface ***Dialogo.onClickListener***, se controla el comportamiento de cada uno de ellos.

La creación y ubicación de los overlays corresponde al método ***onLocationChanged*** de la clase ***GeoUpdateHandler***, como se comentó anteriormente, es necesario refrescar el mapa con cada cambio de posición del usuario ya que la vista ***MapView*** queda invalidada. A continuación mostramos un fragmento del código de ***onLocationChanged***.

```
for(int k=0;k<listaTipos.size();k++){
    Log.d("GeoUpdateHandler","=========================");
    TipoPOI tipoPOI=listaTipos.get(k);

    Log.d("GeoUpdateHandler",tipoPOI.getId());
    Log.d("GeoUpdateHandler",tipoPOI.getNombre());



    poiAdapter=new POIAdapter(actividad);
    listaPOI=poiAdapter.getPOIsByType(tipoPOI.getId());
    Log.d("GeoUpdateHandler","Consultar por tipo: "+tipoPOI.getId());

    if(listaPOI.size()==0) continue;

    //Generación de items de punto de interés
    String nombreTipo=tipoPOI.getNombre().toLowerCase();
    PlacesItemizedOverlay poiOverlay= new PlacesItemizedOverlay(
            mContext.getResources().getDrawable(Menus.hImagenes.get(nombreTipo)),mContext,actividad);

    for(int i=0;i<listaPOI.size();i++){
        POI p=listaPOI.get(i);
        Log.d("GeoUpdateHandler",p.getNombre());
        PlacesOverlay nuevoPunto=new PlacesOverlay(
                new GeoPoint(
                        (int)(Double.parseDouble(p.getLatitud())*1E6),(int)(Double.parseDouble(p.getLongitud())*1E6)),p.getTipoPOI(),p.getNombre(),p);
        poiOverlay.addOverlay(nuevoPunto);
    }

    mapOverlays.add(poiOverlay);
}

tipoAdapter.close();
poiAdapter.close();

//Añadir al mapa las distintas listas de puntos
mapOverlays.add(itemizedoverlay);
```

Ilustración 43 - Construcción de los puntos de interés procedentes de BBDD

En este fragmento de código de observan 2 bucles. El primero de ellos recorre los diferentes tipos de puntos que pueden existir (Museos, Monumentos …), esto permite pintar el icono del POI de un color diferente en base al tipo de punto al que pertenece. Es especialmente interesante el bucle interno ya que es donde se crea el punto. Se observa cómo se crea un nuevo ***PlacesOverlay*** que recibe como parámetro las coordenadas geográficas en el que será ubicado. Cada punto creado se añade a un objeto ***PlacesItemizedOverlay***, que actúa a modo

de lista contenedora de los puntos construidos. Para finalizar, se pasan al mapa todos los ***PlacesItemizedOverlay*** generados.

Manejo de API Google Maps

Además de mostrar mapas de situación, ubicando al usuario y los puntos de interés, el sistema facilitará la ruta para llegar a pie a cada uno de los puntos. Esta funcionalidad ha sido implementada mediante la actividad ***CalculoRuta***, que se apoya a su vez en 2 elementos: Google Maps API y JSON (Véase ***Anexo I - Terminología aplicable***).

La actividad ***CalculoRuta*** obtiene la posición actual del usuario, recupera la posición del punto al que se quiere llegar y realiza una llamada a uno de los servicios Web que componen el API de Google Maps.

La construcción de la URL de la llamada al servicio Web se muestra en la siguiente ilustración.

```
try{

    Log.d("INFORMACION","Preparando conexion ...");
    URL url= new URL("http://maps.google.com/maps/api/directions/json?origin="
            +loc.getLatitude()+","+loc.getLongitude()
            +"&destination="+latitudDestino+","+longitudDestino
            +"&sensor=true&mode=walking&language=es");

    URLConnection urlCon=url.openConnection();
    Log.d("INFORMACION","Conexion abierta ...");
    InputStream in=urlCon.getInputStream();

    StringBuffer store=new StringBuffer();
    DataInputStream dataIn=new DataInputStream(in);

    int b=in.read();
    while(b!=-1){
        store.append((char)b);
        b=in.read();
    }

    dataIn.close();

    Toast.makeText(getBaseContext(), "Finalizado calculo de ruta" , Toast.LENGTH_LONG).show();
```

Ilustración 44 - Proceso de llamada al servicio de cálculo de ruta

La contestación a la petición, por parte de Google Maps, se realiza utilizando JSON. Esta respuesta será tratada por la actividad ***CalculoRuta*** para eliminar información innecesaria y construir la visualización apropiada para el usuario.

```
//Parseo de la respuesta JSon
Gson gson=new Gson();
Ruta ruta=gson.fromJson(store.toString(), Ruta.class);

Routes route=(Routes)(ruta.getRoutes()).get(0);
List<Legs> legs=route.getLegs();

String stDest=null;

for(int i=0;i<legs.size();i++){
    Legs leg=(Legs)legs.get(i);

    stDest=leg.getEnd_address();
    txt.append("<div class=\"cabecera\">");
    txt.append("De: <br/><b><i>"+leg.getStart_address()+"</i></b><br/> a:<br/> <b><i>"+leg.getEnd_address()+"<i></b><br/><br/>");
    txt.append("<hr/>");
    txt.append("<center><b>Distancia del recorrido:</b> "+leg.getDistance().getText()+"</center><br/></div>");

    List<Steps> pasos=leg.getSteps();

    for(int k=0;k<pasos.size();k++){
        Steps step=(Steps)pasos.get(k);
        //System.out.println(step.getDistance().getText()+", "+step.getDuration().getText()+" "+step.getHtml_instructions());
        txt.append("<div class=\"paso\">");
        txt.append("<p>");
        txt.append(step.getDistance().getText()+", "+step.getDuration().getText()+" "+step.getHtml_instructions()+"<br/>");
        if(step.getHtml_instructions().indexOf("derecha")!=-1){
            txt.append("<center><img src=\"file:///android_asset/right-arrow.png\" /></center>");
        }else if(step.getHtml_instructions().indexOf("izquierda")!=-1){
            txt.append("<center><img src=\"file:///android_asset/left-arrow.png\" /></center>");
        }else if(step.getHtml_instructions().indexOf("Cont")!=-1 || step.getHtml_instructions().indexOf("recto")!=-1){
            txt.append("<center><img src=\"file:///android_asset/go-arrow.png\" /></center>");

        }
        txt.append("</p>");
        txt.append("</div>");

    }


}
```

Ilustración 45 - Parseo de JSON de respuesta utilizando GSON

Para mostrar la información al usuario se ha elegido el contenedor WebView. Este contenedor admite código estándar HTML y lo muestra tal y como lo mostraría un navegador en un PC.

Ilustración 46 - Imagen de la ruta calculada

Operaciones con Base de Datos

El sistema gestor de BBDD que utilizaremos en este proyecto es SQLite (véase ***Anexo II - Tecnologías de Desarrollo***). La plataforma Android trae consigo clases destinadas al manejo de SQLite.

La solución de implementación que hemos escogido para operar con BBDD está basada en la utilización de clases de tipo ***Helper*** y clases de tipo ***Adapter***.

La clase principal de conexión a BBDD proporcionada por Android es la clase ***SQLiteOpenHelper***. Se ha creado la clase ***GuiaDatabaseHelper***, extendiendo ***SQLiteOpenHelper***, para tratar nuestro problema concreto. Reescribiendo los métodos de ***SQLiteOpenHelper*** en ***GuiaDatabaseHelper*** hemos realizado las siguientes funciones:

- Crear una BBDD nueva, si no existe previamente
- Actualizar la BBDD si existía previamente y ha cambiado de versión.

La primera vez que se instancia ***GuiaDatabaseHelper***, la clase comprueba la existencia de la BBDD utilizada por la guía. Si esta BBDD no existe en el terminal, se procede a su creación, para ello se llama automáticamente al método ***onCreate***. Hemos reescrito este método, perteneciente a la clase padre (***SQLiteOpenHelper***), para adaptarlo a las necesidades específicas de nuestro aplicativo.

```
@Override
public void onCreate(SQLiteDatabase db) {


    Log.d("Guia","Ejecutando el script de creacion");

    //Script completo de creación de BBDD.
    db.execSQL("CREATE  TABLE \""+TIPO_POI+"\" (\"ID\" INTEGER PRIMARY KEY  AUTOINCREMENT NOT NULL , \"NOMBRE\" VARCHAR NOT NULL  UNIQUE );");
    db.execSQL("CREATE  TABLE \""+POI+"\" (\"id\" INTEGER PRIMARY KEY  AUTOINCREMENT NOT NULL , \"nombre\" VARCHAR NOT NULL , \"latitud\" DOUBLE, \"longitud
    db.execSQL("CREATE  TABLE \""+GALERIA+"\" (\"ID\" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL , \"IMAGEN\" VARCHAR NOT NULL, id_poi INTEGER, FOREIGN KEY(


    //Datos pre-almacenados
    db.execSQL("insert into "+TIPO_POI+" (nombre) values ('Museo')");
    db.execSQL("insert into "+TIPO_POI+" (nombre) values ('Monumento')");
    db.execSQL("insert into "+TIPO_POI+" (nombre) values ('Restaurante')");
    db.execSQL("insert into "+TIPO_POI+" (nombre) values ('Usuario')");

    //Puntos de interés catalogados
    if(context.getResources().getXml(R.xml.lista_pois)==null) Log.i("Guia", "El xmlresources es nulo");
    XMLReaderHelper xmlHelper=new XMLReaderHelper(context.getResources().getXml(R.xml.lista_pois));
    Vector<POI> listaPOIs=xmlHelper.getPOIs();

    for(int i=0;i<listaPOIs.size();i++){
        POI p=listaPOIs.elementAt(i);
        String sql="insert into "+POI+" (nombre,longitud,latitud,imagen,desc_breve,desc_larga,tipo_poi) values "
            +"('"+p.getNombre()+"'"
            +","+p.getLongitud()
            +","+p.getLatitud()
            +",'"+p.getImagen()+"'"
            +",'"+p.getDesc_breve()+"'"
            +",'"+p.getDesc_larga()+"'"
            +","+p.getTipoPOI()+")";

        db.execSQL(sql);

        //Inserción de imágenes asociadas
        Vector<String> galeria=p.getImagenesGaleria();
```

Ilustración 47 - Método onCreate de GuiaDatabaseHelper

En el caso de que la BBDD ya exista, el sistema comprueba automáticamente su versión, en caso de que esta difiera de la existente en el terminal, se llama automáticamente al método ***onUpgrade()***. El método ***onUpgrade***, al igual que ***onCreate***, pertenece a la clase ***SQLOpenHelper*** y ha sido sobreescrito para adaptarlo a nuestras necesidades.

```
@Override
public void onUpgrade(SQLiteDatabase db, int version, int newVersion) {
    Log.d("Guia","Existe un cambio de versión en BBDD. Preparado para actualizar.");

    // TODO Auto-generated method stub
     db.execSQL("DROP TABLE IF EXISTS "+TIPO_POI);
     db.execSQL("DROP TABLE IF EXISTS "+POI);
     db.execSQL("DROP TABLE IF EXISTS "+GALERIA);

     Log.d("DEBUG","Base de datos borrada");

     onCreate(db);
}
```

Ilustración 48 - Método onUpgrade de GuiaDatabaseHelper

Para realizar las operaciones ***CRUD*** de las entidades del modelo se ha creado una clase ***Adapter*** por cada una de ellas. Las clases ***Adapter*** pueden ser llamadas desde cualquier actividad y la aíslan completamente de la BBDD, de esta forma reutilizamos código, ganamos en orden y en facilidad de mantenimiento (en caso de modificación, habría que modificar en un único lugar).

Carga de datos

Para realizar la carga inicial de datos hemos buscado un método flexible, que facilite la gestión de puntos con el menor número de modificaciones posibles en el código de la aplicación.

Con esta premisa hemos creado una estructura XML que contiene un listado de puntos de interés con sus correspondientes atributos. Cuando se detecta que la BBDD debe ser creada o actualizada, la aplicación busca el fichero XML con la lista de puntos de interés, lo trata y lo inserta en la BBDD por medio de la clase ***GuiaDatabaseHelper***.

De esta forma se pueden catalogar fácilmente nuevos puntos de interés o eliminar alguno de los existentes sin realizar grandes modificaciones en el código de la aplicación.

En la ilustración 49 puede observarse la estructura que presenta un punto de interés.

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<listaPOI>
   <poi nombre="El Prado" latitud="40.413858" longitud="-3.692386" tipo="1">
      <imagen>pre_elprado</imagen>
      <descBreve>
      <![CDATA[
      Considerado el noveno museo del mundo por importancia. Reune una amplia colección del realismo español con Velazquez, Goya, Rubens o Murillo a la cabeza
      ]]>
      </descBreve>
      <descLarga>
      <![CDATA[
      El Museo Nacional del Prado, ubicado en Madrid, España, es uno de los más importantes y más visitados del mundo (el noveno en 2009). Singularmente rico
      que posee las mejores colecciones que existen a nivel mundial, a lo que hay que sumar destacados conjuntos de autores tan importantes como Murillo, Ribe
      Al igual que otros grandes museos europeos, como el Louvre de París y los Uffizi de Florencia, el Prado debe su origen a la afición coleccionista de las
      El Prado no es un museo enciclopédico al estilo del Museo del Louvre, la National Gallery de Londres, o incluso (a una escala mucho más reducida) el vec
      Las escuelas pictóricas de España, Flandes e Italia (sobre todo Venecia) ostentan el protagonismo en el Prado, seguidas por el fondo francés, más limita
      Aunque sean aspectos menos conocidos, cuenta también con una importante sección de Artes decorativas (Tesoro del Delfín) y con una destacada colección d
      ]]>
      </descLarga>
      <listadoImagenesGaleria>
         <imagenGaleria>fraguavulcano_velazquez_b</imagenGaleria>
         <imagenGaleria>hilanderas_velazquez_b</imagenGaleria>
         <imagenGaleria>meninas_velazquez_b</imagenGaleria>
         <imagenGaleria>rendicionbreda_velazquez_b</imagenGaleria>
         <imagenGaleria>venusespejo_velazquez_b</imagenGaleria>
         <imagenGaleria>esopo_velazquez_b</imagenGaleria>
      </listadoImagenesGaleria>
   </poi>
```

Ilustración 49 - XML de puntos de interés

La lectura del XML la realiza la clase ***XMLReaderHelper***, de forma que el resto de clases quedan aislados de los detalles de esta lectura y tratamiento de información.

A continuación mostramos el código de la clase ***XMLReaderHelper***.

```
try{
    xmlFile.next();
    int eventType = xmlFile.getEventType();

    POI poi=null;
    while (eventType != XmlPullParser.END_DOCUMENT){

        if(eventType == XmlPullParser.START_DOCUMENT){
            Log.i("Guia","Comienza el catalogo de puntos de interés");
        }else if(eventType == XmlPullParser.START_TAG){
            String nameTag=xmlFile.getName();
            if(nameTag.equals("poi")){
                String nombre=xmlFile.getAttributeValue(0);
                String latitud=xmlFile.getAttributeValue(1);
                String longitud=xmlFile.getAttributeValue(2);
                String tipo=xmlFile.getAttributeValue(3);

                poi=new POI();

                poi.setNombre(nombre);
                poi.setLongitud(longitud);
                poi.setLatitud(latitud);
                poi.setTipoPOI(tipo);
            }else if(nameTag.equals("imagen")){
                xmlFile.next();
                String imagen=xmlFile.getText();
                Log.d("Guia","Imagen: "+imagen);
                poi.setImagen(imagen);
            }else if(nameTag.equals("descBreve")){
                xmlFile.next();
                String descBreve=xmlFile.getText();
                Log.d("Guia","DescBreve: "+descBreve);
                poi.setDesc_breve(descBreve);
            }else if(nameTag.equals("descLarga")){
                xmlFile.next();
                String descLarga=xmlFile.getText();
                Log.d("Guia","DescLarga: "+descLarga);
                poi.setDesc_larga(descLarga);
            }else if(nameTag.equals("imagenGaleria")){
                Log.i("Guia", "PASO POR LA GALERIA DE IMAGENES");
                xmlFile.next();
                String imagenGaleria=xmlFile.getText();
                Log.i("Guia",imagenGaleria);
                poi.addImagenGaleria(imagenGaleria);
            }
        }else if(eventType==XmlPullParser.TEXT){
            Log.d("Guia",xmlFile.getText());
        }else if(eventType==XmlPullParser.END_TAG){
            if(xmlFile.getName().equals("poi")){
```

Ilustración 50 - Tratamiento del listado de puntos de interés

Para realizar la lectura y tratamiento del fichero se ha utilizado la clase ***XmlResourceParser***, esta clase es parte del API de Android.

Manejo de la cámara fotográfica

En la operativa de gestión de puntos de interés de usuario, existe la posibilidad de asociar una fotografía al punto de usuario que va a ser dado de alta. Esta fotografía puede ser tomada en el momento haciendo uso de la cámara que incorporan todos los Smarthphones.

Para utilizar la funcionalidad de la cámara se ha utilizado la clase MediaStore, perteneciente al API de Android, tal y como se muestra en el siguiente fragmento de código.

```
Intent intento =  null;

switch (item.getItemId()) {
      case R.id.usuario_poi_camara:
          intento=new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

          //Configuración de ruta
          name=Environment.getExternalStorageDirectory()+"/test.jpg";

          Uri output = Uri.fromFile(new File(name));
          intento.putExtra(MediaStore.EXTRA_OUTPUT,output);
          startActivityForResult(intento,TAKE_PICTURE);
          return true;
```

Ilustración 51 - Llamada a la cámara fotográfica del terminal

Podemos observar cómo se crea un intento y se inicia la actividad indicando que se queda a la espera del resultado de la misma.

Una vez finalizada la actividad, se recoge la imagen proporcionada por la cámara y se muestra en el área de visualización.

```
if(savedInstanceState!=null){
     name=savedInstanceState.getString("name");
     foto=BitmapFactory.decodeFile(name);
     int w = foto.getWidth();
     int h = foto.getHeight();
     // Matriz contenedora de imagen
     Matrix mtx = new Matrix();
     mtx.postRotate(90);
     // Rotación de Bitmap
     foto = Bitmap.createBitmap(foto, 0, 0, w, h, mtx, true);
     BitmapDrawable bmd = new BitmapDrawable(foto);

     previsualizar.setImageDrawable(bmd);
 }
```

Ilustración 52 - Previsualización de la imagen

Cuando el usuario decide guardar el punto que ha fotografiado, el sistema se encarga de guardar la fotografía en un directorio dentro de la tarjeta de memoria del usuario.

```
private String saveToFile(Bitmap b) throws IOException{
      String fileName=String.valueOf(Calendar.getInstance().getTimeInMillis())+".jpg";
      Log.i("Guia","path: "+Menus.hResourcePath.get(Menus.FOTOGRAFIA));
      OutputStream fOut = null;
      File file = new File(Menus.hResourcePath.get(Menus.FOTOGRAFIA), fileName);
      fOut = new FileOutputStream(file);


      b.compress(Bitmap.CompressFormat.JPEG, 70, fOut);
      fOut.flush();
      fOut.close();

      MediaStore.Images.Media.insertImage(getContentResolver(),file.getAbsolutePath(),file.getName(),file.getName());

      return fileName;
}
```

Ilustración 53 - Salvar imagen

Puede observarse en el código de la ilustración 52 que el mapa de bits se transforma en una imagen con formato jpeg, para ello reducimos la calidad de la imagen al 70%, ahorrando espacio en el almacenamiento.

Para evitar colisiones entre nombres de imágenes, se genera un nombre aleatorio basado en la fecha actual.

Por último, haciendo uso de la clase Media, introducimos la imagen capturada en la galería de imágenes del terminal, pudiendo así acceder a ella de forma externa a la aplicación.

Manejo de galería

En la operativa de gestión de puntos de interés de usuario, existe la posibilidad de asociar una fotografía al punto de usuario que va a ser dado de alta. Se ha contemplado la posibilidad de asociar esta imagen desde la galería de fotografías disponibles en el terminal, de esta forma el usuario podrá asociar al punto de interés cualquier fotografía almacenada en el teléfono.

El manejo de la galería se realiza mediante un intento, tal y como se muestra en el siguiente fragmento de código.

```
case R.id.usuario_poi_galeria:
    intento = new Intent(Intent.ACTION_PICK, android.provider.MediaStore.Images.Media.INTERNAL_CONTENT_URI);
    //Configuración de ruta
    name=Environment.getExternalStorageDirectory()+"/test.jpg";
    startActivityForResult(intento,SELECT_PICTURE);
```

Ilustración 54 - Llamada a la galería del terminal

Una vez seleccionada la imagen, se sigue el proceso mostrado en la ilustración 52 para previsualizar.

Manejo de autocompletar

Una de las mayores dificultades en el uso de aplicaciones para móviles es la ausencia de un teclado físico, a la manera de un portátil o de un PC de sobremesa, que permita teclear de forma ágil. Es por ello que se ha incorporado a la aplicación una funcionalidad de autocompletado que permite localizar un punto de interés de usuario tecleando únicamente parte de su nombre.

Esta funcionalidad se consigue realizando, en el inicio de la actividad POIUsuario, una consulta de los puntos de interés dados de alta por el usuario.

Recordamos que el layout de la actividad ***POIUsuario*** presenta un campo de tipo ***AutoCompleteTextView***, a este campo le asociaremos una instancia de la clase ***ArrayAdapter*** que contendrá los nombres de los puntos de interés. De esta forma, el elemento ***AutoCompleteTextView*** comprobará automáticamente los elementos de la lista de nombres que coincidan en parte con el texto que está tecleando el usuario.

El proceso es muestra en el siguiente fragmento de código.

```
private void autocomplete(){
    Log.d("Guia","Construyendo estructura para autocompletar");

    lista=poiAdapter.getAllUserPOIs();


    SITIOS=new String[lista.size()];
    for(int i=0;i<lista.size();i++) SITIOS[i] = (lista.get(i)).getNombre();

    //  for(int i=0;i<SITIOS.length;i++) Log.d("Guia", SITIOS[i]);


    //Opciones de autocompletar
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_dropdown_item_1line, SITIOS);
    nombre = (AutoCompleteTextView)findViewById(R.id.poi_usuario_nombre_auto);
    nombre.setAdapter(adapter);

    nombre.setOnFocusChangeListener(this);
    Log.d("Guia","Construcción de autocompletar finalizada");

    descripcion=(TextView)findViewById(R.id.poi_usuario_desc);
}
```

Ilustración 55 - Método autocomplete

10.3 Especificación de características y permisos

Para el correcto funcionamiento del aplicativo se requerirá que terminal tenga unas características definidas, además será necesario que el usuario autorice a la guía a utilizar diferentes elementos del terminal.

La especificación de los elementos y permisos necesarios para la correcta ejecución se recoge en el fichero AndroidManifest.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="es.guia.interfaz"
      android:versionCode="1"
      android:versionName="0.27"
      android:installLocation="preferExternal">
     <uses-permission android:name="android.permission.INTERNET" />
     <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Menus"
                  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=".PasoParametro"></activity>
        <activity android:name=".CalculoRuta"></activity>
        <activity android:name=".ListadoPOIs" android:label="Lista de puntos de interes"></activity>
        <activity android:name=".DescripcionPOI" android:label="Punto de Interes"></activity>
        <activity android:name=".Ayuda" android:label="Ayuda"></activity>
        <activity android:name=".InformacionPOI" android:label="Informacion de servicio"></activity>
        <activity android:name=".Galeria" android:label="Galeria de Imagenes"></activity>
        <activity android:name=".POIUsuario" android:label="Gestión de puntos de usuario" android:screenOrientation="nosensor"></activity>

        <activity android:name=".MapaSituacion" android:label="Mapa de Situacion"
            android:theme="@android:style/Theme.NoTitleBar" android:noHistory="true">
         </activity>
          <uses-library android:name="com.google.android.maps" />
</application>
    <uses-sdk android:minSdkVersion="8"></uses-sdk>
</manifest>
```

Ilustración 56 - Detalle del fichero AndroidManifest

A continuación se exponen los puntos más relevantes utilizados en la guía.

Ubicación del Software

Mediante el elemento ***android:installLocation*** permitimos al usuario instalar la guía tanto en la memoria interna del teléfono como en la memoria externa. De esta forma situar el software donde más le convenga en cada momento.

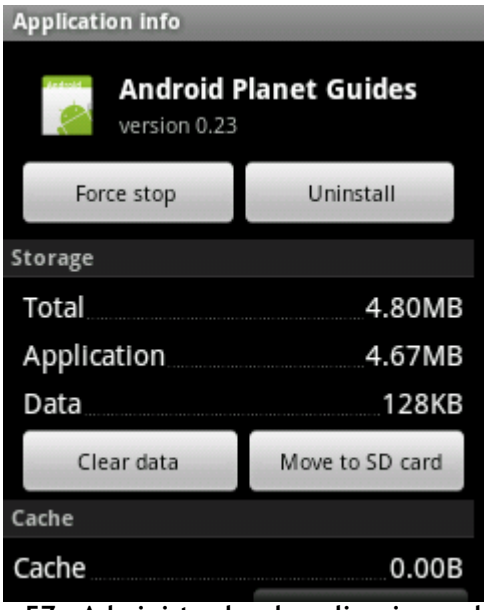


Ilustración 57 - Administrador de aplicaciones de Android

Permisos

Será necesario solicitar permisos para utilizar los siguientes elementos del terminal:

- Conexión a Internet (***android.permission.INTERNET***)
- Acceso a GPS (***android.permission.ACESS_FINE_LOCATION***)
- Permiso de escritura en la tarjeta de memoria externa (***android.permission.WRITE_EXTERNAL_STORAGE***)

Se solicitan mediante el elemento ***uses-permission***, tal y como se observa en la ilustración 56.

Versión de Android necesaria

Para que la aplicación funcione adecuadamente, necesitaremos una versión mínima del SDK instalado en el sistema operativo. En este caso será la versión 8 y se indicará mediante el elemento ***uses-sdk***.