Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación de Android para geo-localización de tareas pendientes. Gestión e implementación de BD y aplicación móvil

Trabajo Fin de Grado

Grado en Ingeniería Informática

Curso: 2013 - 2014

Autor: Jessica Carpintero Macián

Tutor: Joan Josep Fons Cors

Resumen

Cada día que pasa la gente tiene menos tiempo para organizarse, para recordar que tenía que hacer, para buscar el sitio adecuado. ¿Cuántas veces la gente pasa por delante de un sitio el cual tenía algo que comprar o hacer y por no recordarlo le toca volver otro día?

Para ello este proyecto va explicar una aplicación móvil basada en un gestor de tareas, donde al usuario se le permitirá gestionar todas sus tareas pendientes, así como añadir sus puntos de interés favoritos donde poder realizarlas. A parte, la aplicación mostrará en todo momento su posición y las tareas que tenga más próximas, igual que notificaciones de cercanía y tiempo que permitirán al usuario recordar si tenía que realizar alguna tarea. Además, el usuario podrá sincronizar sus tareas contenidas con otros gestores, permitiéndole tener en una única aplicación, todo unificado. Toda la información de la aplicación estará almacenada en un servidor que será el que haga de conexión con el resto de gestores y donde estará almacenado todos los datos del usuario.

Para conseguir el objetivo explicado en el párrafo anterior, se ha realizado una aplicación móvil en la plataforma Android, la cual hace uso de muchas librerías propias del entorno para proporcionar al usuario una aplicación fácil de usar y que se acomode a sus necesidades. Por otro lado, la aplicación ha utilizado las APIs de Google Maps y Google Task para poder realizar el mapa y la sincronización de tareas. Para la creación del servidor y su comunicación con la aplicación y con otros gestores de tareas se ha utilizado PHP y una capa RESTful.

Una vez finalizada la aplicación y el servidor se ha conseguido que el usuario pueda tener agrupadas todas las tareas que tenga creadas en la propia aplicación, a través de un sistema unificado y él cual le notifica en todo momento si existe alguna tarea cercana. Esto ayuda mucho a las personas olvidadizas y desordenadas a mantener sus tareas al día, permitiéndoles vivir con mayor calidad de vida.

Palabras clave: gestor, tareas, aplicación, móvil, sincronización, geolocalización, servidor.

Abstract

Every day that passes people have less time to organize, to remember what they should do, to find the right place. How many times people pass in front of a site which had something to do or buy but do not remember it, and therefore must come again another day?

For this reasons, this project will explain a mobile application based on a task manager where the user will be allowed to manage all your pending tasks and add your favorite points of interest where they can perform them. In addition, the application displays your position all the time and nearest tasks, plus notifications of closeness and countdown that will remind the user to perform some task. Furthermore, users can synchronize their tasks with other managers, allowing their to have in a single application, unified whole. All application information will be stored on a server that will synchronize with other managers and stored all user data.

To achieve the objective described in the previous paragraph, we have made a mobile application on the Android platform, which makes use of many own environment libraries to provide the user an easy to use application that fulfil your needs. On the other hand, the application uses Google Maps API to perform the map and Google Task API for tasks synchronization. To develop the server and its communication interface with the application and other task managers we use PHP and a RESTful layer.

Upon completion the application and the server, the user can have grouped all the tasks created in both applications through a unified system that notifies you all the time if there is a near task. This helps a lot to people forgetful and disorganized to maintain their daily tasks, allowing them to live with greater quality of life.

Keywords: manager, tasks, application, mobile, synchronization, geo-localization, server.

Tabla de contenidos

1. Siglas y abreviaturas

HTML (HyperText Markup Language): es un lenguaje de marcado para la desarrollo de páginas web.

PHP (Hypertext Pre-Processor): es un lenguaje de programación del lado del servidor diseñado para webs de contenidos dinámicos.

HTTP (Hypertext Transfer Protocol): Es el un protocolo sin estado utilizado para la transacción web.

REST (Representational State Transfer): es una técnica de arquitectura software sobre capa HTTP que establece la idea de tratamiento del contenido como recursos con operaciones aplicables.

IDE (Integrated Development Environment): Entorno de desarrollo que ofrece frameworks y herramientas para proyectos sobre distintos lenguajes.

GPS (Global Positioning System*):* sistema de navegación espacial basado en geo-posicionamiento por satélite. Proporciona al usuario información precisa sobre su posición y hora actual.

SQL (Structured Query Language): Lenguaje estructuras de consultas de base de datos relacionales.

MySQL: Sistema gestor relacional de bases de datos, multiusuario, rápido y robusto, que utiliza el lenguaje SQL.

SqLite: es un sistema de gestión de base de datos relacional compacto, comprimido en un único fichero y muy ligero.

XML (eXtensible Markup Language): lenguaje de marcas utilizado para el almacenamiento de datos sin codificación.

SOAP (*Simple Object Access Protocol*): protocolo estándar que define la forma de que dos objetos en diferentes procesos puedan comunicarse con datos XML.

XML-RPC (eXtensible Markup Language – Remote Procedure Call): protocolo de transmisión de mensajes basado en XML.

Java: Lenguaje de programación orientado a objetos y clases, concurrente y ejecutable en cualquier dispositivo que tenga la máquina virtual de java.

API (Application Programming Interface): conjunto de funciones y procedimientos que proporciona una librería para poder ser utilizado por otros software como una capa de abstracción.

2. Tabla de ilustraciones

3. Introducción

Se ha diseñado un gestor de tareas que incorpora una capa de interoperabilidad para poder comunicarse con diferentes plataformas de tareas. Además, tiene acceso a una base de datos con geo-localización de los puntos de interés donde se pueden realizar las tareas.

El uso de este gestor se realiza a través de aplicaciones móviles, en este caso se ha decidido desarrollar una aplicación implementada en Android. La aplicación permite al usuario gestionar sus tareas, tanto de este gestor como de otros (Google Tasks,…), y geo-localizarlas en un mapa.

3.1 Historia y motivación

Actualmente el desarrollo de aplicaciones está en alza con previsión de crecimiento en el futuro. Estos datos se pueden confirmar viendo en la Ilustración las estadísticas que demuestran que los Smartphone son los dispositivos más utilizados hoy en día. Las personas utilizan su Smartphone para realizar cualquier actividad, todo esto es gracias a la gran variedad de apps que pueden contener. Estas apps permiten a las personas entretenerse, comunicarse, gestionarse,.... Y si a esto se le suma que cada día que pasa la gente tiene menos tiempo para sus tareas y vive estresada hace pensar que hacía falta una aplicación que les ayude a gestionar sus tareas cotidianas. Por ello se ha decidido realizar un gestor de tareas con su aplicación móvil para brindar a las personas el poder gestionar sus tareas sin necesidad de perder mucho tiempo. Ya no tendrán que preocuparse en recordar que tarea tenían pendiente y en donde tenían que realizarla, la aplicación se encargara de almacenar toda esa información y suministrarla cuando sea necesario.

3.2 Objetivos

En este apartado se definen los objetivos personales para este proyecto en grupo, los cuales consisten en diseñar e implementar gran parte de la aplicación móvil, así como el modelo de datos del sistema y gestión de la base de datos. A continuación se definen a nivel más profundo los objetivos que se quiere alcanzar:

- Representar la posición geográfica del usuario.
- Consultar el listado de tareas pendientes.

- Gestionar una base de datos con las tareas pendientes y sus puntos de interés donde se puedan realizar.
- Geo-localizar en un mapa la posición donde se oferta la localización de una tarea, y los puntos de interés donde se pueda realizar.
- Notificar las localizaciones donde se presten las tareas más cercanas al usuario.

3.3 Antecedentes

Este apartado se centra en comparar los objetivos de la aplicación con otras aplicaciones que están en el mercado e intentar suplir sus deficiencias dando mayores funcionalidades al usuario. En este caso se han elegido dos bastante usadas que son explicadas a continuación:

1. ePythia :
    - 1.1 Tareas geolocalizadas
        - 1.1.1 Añadir tareas desde su localización concreta
        - 1.1.2 Asignar en el mapa localización de la tarea
        - 1.1.3 Introducción de tareas manual como de voz
        - 1.1.4 Notificaciones
        - 1.1.5 Definir radio para las notificaciones según prioridad de la tarea
        - 1.1.6 Radar para revisar lugares cercanos para realizar tareas pendientes
    - 1.2 Sincronización en la nube

2 Wunderlist:
- 2.1 Planifica cualquier tarea
- 2.2 Comparte con cualquiera las tareas
- 2.3 Accede desde cualquier lugar
- 2.4 Creación de tareas concurrentes
- 2.5 Subtareas (Las tareas grandes pueden subdividirse en pequeñas para facilitar su realización)
- 2.6 Notas
- 2.7 Fecha de vencimiento en las tareas
- 2.8 Recordatorios
- 2.9 Notificaciones
- 2.10 Posibilidad de imprimir
- 2.11 Sincronización con la nube
- 2.12 Utilizable desde cualquier dispositivo

Estas aplicaciones tienen extras que se quiere incluir en la aplicación (tareas geo-localizadas, notificaciones, fecha de vencimiento de las tareas…), pero aparte de todas estas funcionalidades nuestra aplicación también puede

sincronizar tareas con otros servidores, dando al usuario la oportunidad de tener toda sus tareas contenidas en un único lugar.

Seguidamente, se explica cómo se estructura la memoria con sus diferentes apartados.

3.4 Estructura de la memoria

Este proyecto está compuesto por el grupo de alumnos de la UPV ETSINF Emanuel Alloza Álvarez y Jessica Carpintero Macián. En esta memoria se define la parte correspondiente a Jessica Carpintero Macian.

En primer lugar se han definido las diferentes siglas y abreviaturas, para que el lector tenga referencia de todas ellas según lea el contenido de la memoria. Seguidamente se encuentra el índice de la Ilustraciones. Después, está la introducción la cual pone en contexto al lector sobre lo que va a tratar el proyecto, en este punto se encuentran también definidos los objetivos y su estructura.

A continuación se detallan los diferentes apartados que han ayudado a definir el proyecto. Primero de todo se explicará una breve comparativa de las tecnologías conocidas para implementarlo y la explicación de que tipo de tecnologías se han escogido. En segundo lugar se ha definido la especificación de requisitos donde se recoge toda la información a través de la identificación de los requisitos funcionales y no funcionales. Justo después, se habla de la planificación donde está representada todas las funcionalidades que tiene que contener el proyecto, como un listado de prioridades de implementación y un esquema cronológico con la representación del tiempo estimado que debería costar realizar el proyecto.

En el siguiente apartado se define el modelo de datos que sigue la aplicación, el diseño está definido en la memoria de mi compañero donde se encuentran los ejemplos de cada pantalla, divididas en función de las funcionalidades descritas. Seguidamente esta explicada la implementación de la aplicación, en ellas se describen y visualizan los diferentes métodos que han definido las funcionalidades. Seguidamente, hay una descripción de las pruebas que se han realizado de que problemas han conllevado y su solución.

Y por último, se encuentra las conclusiones que se han llegado después de realizar todo el proyecto, como sus mejoras a largo plazo para dar mayores servicios al usuario y la forma de poder monetizar la aplicación y así sacar rentabilidad al trabajo realizado y mejores versiones para que los usuarios tengan la aplicación que ellos deseen.

4. Contexto tecnológico

4.1 Comparativa de tecnologías, frameworks e IDEs

Se han evaluado los requisitos de la aplicación y del servidor, y tras realizar un pequeño estudio de las tecnologías disponibles se han tomado las siguientes resoluciones:

La aplicación móvil se desarrollará para la plataforma Android, debido al último estudio de Worldpanel Comtech, el 88.6% de los usuarios Españoles de dispositivos móviles utiliza dicha plataforma, sumado a que incluye una gran compatibilidad con Google Maps y otras herramientas para notificaciones. Esto complementado a la experiencia y soporte del lenguaje forman la mejor opción.

	ESPAÑA	EUROPA*	EEUU	CHINA	AUSTRALIA	JAPÓN
Android	88,6	70,7	57,6	80,0	57,3	41,5
BlackBerry	0,0	1,1	0,7	0,1	1,0	0,0
iOS	7,6	19,2	35,9	17,9	33,1	57,6
Windows	3,0	8,1	5,3	1,0	6,9	0,9
Other	0,8	0,9	0,4	1,0	1,7	0,0

Fuente: Worldpanel Comtech
*Alemania, Reino Unido, Francia, Italia, España

Ilustración 1 - Estadísticas del uso de Android

Como complemento se utilizará un sistema de base de datos de sqlite por su ligereza en la plataforma de Android, y Google Maps por ser la mejor opción en cuanto a calidad de mapas y ligereza de datos.

La aplicación será realizada con Android Studio, el IDE propio de Google para programar en Android. La elección de este IDE es que aporta la posibilidad de ver cómo queda la aplicación en diferentes versiones de forma automática, sin necesidad de compilar ni probar en diferentes dispositivos. Ayuda a la localización de las clases gracias a que mantiene una estructura jerárquica. Además, contiene gradle que realiza las tareas de compilación, testing, empaquetado y el despliegue de los proyectos automáticamente.

4.2 Tecnologías elegidas

4.2.1 Android

Android es un sistema operativo orientado a dispositivos móviles, basado en una versión modificada del núcleo Linux. Se trata de un sistema abierto, multitarea, que permite a los desarrolladores acceder a las funcionalidades principales del dispositivo mediante aplicaciones. Cualquier aplicación puede ser reemplazada libremente o ser desarrollada por terceros, a través de herramientas proporcionadas por Google, mediante los lenguajes de programación Java y C.

El código fuente de Android está disponible bajo diversas licencias de software libre y código abierto, Google liberó la mayoría del código de Android bajo la licencia Apache. Todo esto permite que un desarrollador no solo pueda modificar su código sino también mejorarlo. A través de esas mejoras puede publicar el nuevo código y con el ayudar a mejorar el sistema operativo para futuras versiones.

4.2.2 SQLite

SQLite es un motor de base de datos SQL embebido. A diferencia de otros gestores de base de datos no tiene un proceso de servidor independiente permitiéndole ser autónomo. Contiene una completa base de datos SQL, con varias tablas, índices, triggers, etc y todo ello está contenido en un único archivo de disco único. Su formato es multiplataforma por tanto, le permite ser portable para cualquier sistema. Además, no es necesario instalarlo ni configurarlo, esto hace que su uso sea más sencillo y por tanto cada día que pasa es más utilizada en cualquier tipo de sistema.

4.2.3 Google MAPS API

La API de Google Maps es un conjunto de APIs que permite sobreponer datos sobre un mapa de Google Maps personalizado. Con ello, se puede crear atractivas aplicaciones web y móviles, donde poder agregar imágenes de satélite, Street View, perfiles de elevación, indicaciones sobre cómo llegar, mapas con estilos, análisis y una amplia base de datos de ubicaciones.

Además, gracias a su cobertura global y sus actualizaciones diarias permite conceder un servicio en mejora constante permitiendo que las aplicaciones estén actualizadas en cualquier momento.

4.2.4 REST

Rest es tipo de arquitectura de desarrollo web que utiliza el estándar HTTP. Permite a servicios y aplicaciones ofrecer su contenido en forma de recursos, accesibles por cualquier cliente o dispositivo que utilice HTTP. Es mucho más simple y ligero que SOAP y XML-RPC.

5. Especificación de requisitos

Lo primero que se realiza siempre si se quiere obtener un resultado satisfactorio en la aplicación es tener una buena planificación y especificación en este apartado realizado por los dos integrantes del equipo se define los pasos que se han llevado acabo para tomarlos. Casi todos los pasos han sido planteados en común, pero cada integrante ha redactado sus casos de uso individuales que estarán definidos en la memoria complementaria del otro integrante.

5.1 Toma de requisitos

El primer paso para desarrollar el gestor de tareas y la aplicación móvil es la toma de requisitos funcionales y no funcionales:

- Funcionales: Describen la interacción entre el sistema y su ambiente independientemente de su implementación. El ambiente incluye al usuario y cualquier otro sistema externo que interactúa con el sistema.
- No funcionales: Describen las restricciones y obligaciones que el sistema debe contener.

Complementario a esto, se ha realizado un análisis de las tecnologías existentes para evaluar su funcionalidad y rendimiento, para tomar las decisiones de implementación de este proyecto más adecuadas.

5.2 Requisitos funcionales

En este apartado se describe la interacción entre el sistema y su ambiente (usuario y cualquier sistema externo) independiente de su implementación.

5.2.1Actores

Los actores son los que representan un conjunto de roles, que son jugados por una persona, dispositivo u otros sistemas que interactúan con la aplicación. En este caso, el sistema contiene cuatro roles diferentes definidos a continuación:

- ***Usuario no registrado:*** Este actor solo tiene acceso a registrarse, para después poder pasar a ser un usuario registrado.

- ***Usuario registrado:*** Es el actor que tiene acceso a todas las funcionalidades de la aplicación. Toda la aplicación está orientada para dar servicios a este usuario con la idea de mantener todas sus tareas pendientes unidas y geo-localizadas para que sepa en cada momento que tiene pendiente cerca de él.

- ***Servicio de autentificación de Google Tasks:*** Permite al usuario conectarse a su lista de tareas contenida en Google Tasks para sincronizarlas con las que tendrá en la aplicación.

- ***Administrador del servidor:*** Actor encargado de gestionar y mantener el servidor.

5.2.2 Persona

El diseño de esta aplicación es utilizable por la mayoría de usuarios habituales de dispositivos móviles, si bien es cierto que un gestor de tareas es una herramienta usable por un amplio rango de usuarios, su uso habitual y cotidiano se enfoca entre personas de entre 16 y 65 años. No se han hecho excesivas distinciones en cuanto a sexo, religión o situación económica, pues se considera que por el formato de aplicación no existen diferencias significativas.

En cuanto a la orientación hacia profesiones de los usuarios, sí que se enfoca la balanza hacia gente que trabaja en empleos de amplia jornada laboral, estresantes y lo más posible pertenecientes al sector servicios, por lo que se ha realizado el diseño visual e interactivo pensando en la rapidez y facilidad de uso.

Pensando en el estilo de vida estresante de los usuarios se ha diseñado un sistema personalizable y no intrusivo de notificaciones para ayudar por motivos de cercanía y cuenta atrás de plazos para una tarea. Dichas notificaciones se pueden configurar por tipo de alerta, prioridad de tareas, sonido, vibración y color de led de parpadeo.

Para cubrir el mayor rango de usuarios se ha diseñado e implementado un sistema de servidor y aplicación multilenguaje con dos idiomas de serie: castellano e inglés. Con los que se cubre un gran porcentaje de la población, y es ampliable.

El usuario ejemplo o " Persona" a utilizar en el escenario siguiente es Paco, un varón de 30 años oficinista de una mediana empresa, casado y con un hijo. A continuación se describe un escenario ejemplo de su interacción con la aplicación.

5.2.3 Escenario

El escenario representa un claro uso del sistema en términos de una serie de interacciones entre la aplicación y el usuario. A continuación se detalla el escenario:

El usuario Paco, es una persona algo olvidadiza. Después de desayunar coge su móvil y hace a la aplicación WhereIsMyTask, donde ya tiene su usuario y contraseña guardados por haber entrado otras veces, por lo que pulsa directamente iniciar sesión.

Una vez dentro de la aplicación, visualizando el mapa, pulsa el botón de agregar tarea y crea una indicando que tiene que comprar patatas con prioridad alta, pues su novia le ha indicado que las quiere para la cena. Como valores para esta tarea establece en el campo dirección unos de sus puntos de interés favoritos (el Mercadona de la esquina) y la categoría (de alimentación). Tras guardar la tarea, vuelve a la página principal y comprueba que hay un marcador rojo en la calle del Mercadona.

Acto seguido pulsa el botón de sincronizar tareas, y tras acceder al listado de tareas, comprueba que existe la tarea "Comida de empresa a las 14:00", que proviene de Google Tasks. Al medio día, tras haber ido a la comida y gustarle el restaurante, accede a la aplicación y en el gestor de puntos de interés agrega su nuevo restaurante favorito pulsando su ubicación en el mapa.

Por la tarde volviendo a casa, cansado de todo el día y olvidado del encargo de su novia, al pasar a cien metros del Mercadona de la esquina le salta una notificación de la aplicación advirtiéndole de la tarea y cercanía de dicho lugar, por lo que Paco ira a comprar patatas y tendrá contenta a su novia.

5.2.4 Casos de uso

Los casos de uso son los encargados de la abstracción que describe una clase de escenarios. A continuación se detallaran los diferentes casos de uso del sistema:

- ***Ajustes del mapa***

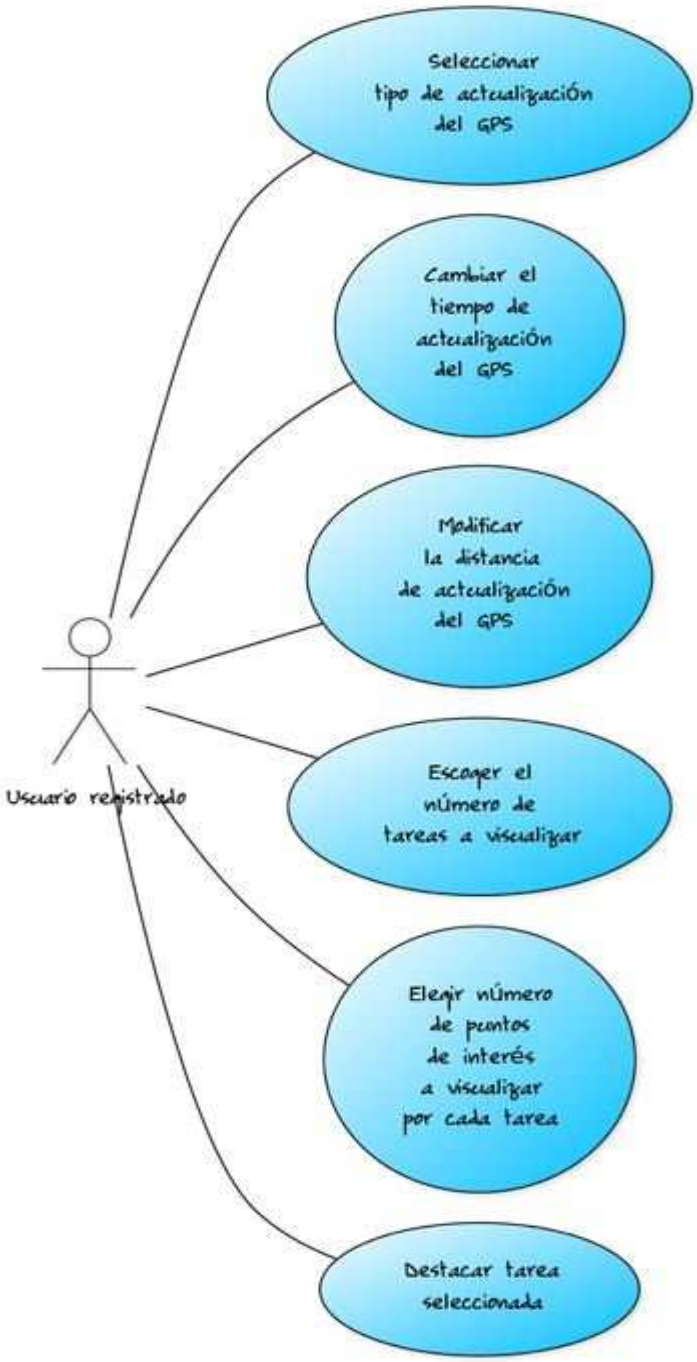


Ilustración 2 - Ajustes del mapa

  - *Seleccionar tipo de actualización del GPS*

Titulo:	Seleccionar tipo de actualización del GPS
Descripción	El usuario puede elegir el tipo de actualización que desee para el GPS. Tendrá tres opciones por tiempo, distancia o modo automático
Actor	Usuario registrado
Relaciones	
Precondición	GPS activo
Comentarios	La opción por tiempo como por distancia permitirá al GPS refrescarse según los parámetros que le indique el usuario. En el acaso automático se utilizara un algoritmo propio en base a posiciones anteriores del usuario, optimizando así el uso de la batería del dispositivo

- *Cambiar el tiempo de actualización del GPS*

Titulo:	Cambiar la distancia de actualización del GPS
Descripción	El usuario puede elegir cada cuanta tiempo quiere que se actualice el GPS
Actor	Usuario registrado
Relaciones	
Precondición	GPS activo y la opción de tipo de actualización por tiempo tiene que estar seleccionada
Comentarios	Opciones: 30 s, 1 min, 2 min, 5 min

- *Modificar la distancia de actualización del GPS*

Titulo:	Modificar la distancia de actualización del GPS
Descripción	El usuario puede elegir cada cuanta distancia quiere que se actualice el GPS
Actor	Usuario registrado
Relaciones	
Precondición	GPS activo y la opción de tipo de actualización por distancia tiene que estar seleccionada
Comentarios	Opciones: 100 m, 200 m, 300 m, 500 m, 1 Km, 5 Km

- *Escoger el número de tareas a visualizar*

Título:	Escoger el número de tareas a visualizar
Descripción	El usuario puede escoger el número de tareas a visualizar en el mapa
Actor	Usuario registrado
Relaciones	
Precondición	
Comentarios	Opciones: 1 tarea, 2 tareas, 5 tareas, 10 tareas, Todas las tareas

- *Elegir número de puntos de interés a visualizar por cada tarea*

Título:	Elegir número de puntos de interés a visualizar por cada tarea
Descripción	El usuario puede elegir el número de puntos de interés por cada tarea que quiere visualizar en el mapa
Actor	Usuario registrado
Relaciones	
Precondición	
Comentarios	Opciones: 0 POI, 1 POI, 2 POI, 3 POI, 4 POI, 5 POI

- *Destacar tarea seleccionada*

Título:	Destacar tarea seleccionada
Descripción	El usuario puede decidir si quiere que se destaque con un color diferente la tarea y sus puntos de interés correspondientes en el mapa
Actor	Usuario registrado
Relaciones	
Precondición	
Comentarios	Cuando se selecciona esta opción la tarea seleccionada será representada de color morado y sus puntos de interés de color turquesa. A diferencia del resto que están en color rojo, amarillo o verde dependiendo de la prioridad que tengan asignada

- ***Ajustes de las notificaciones***

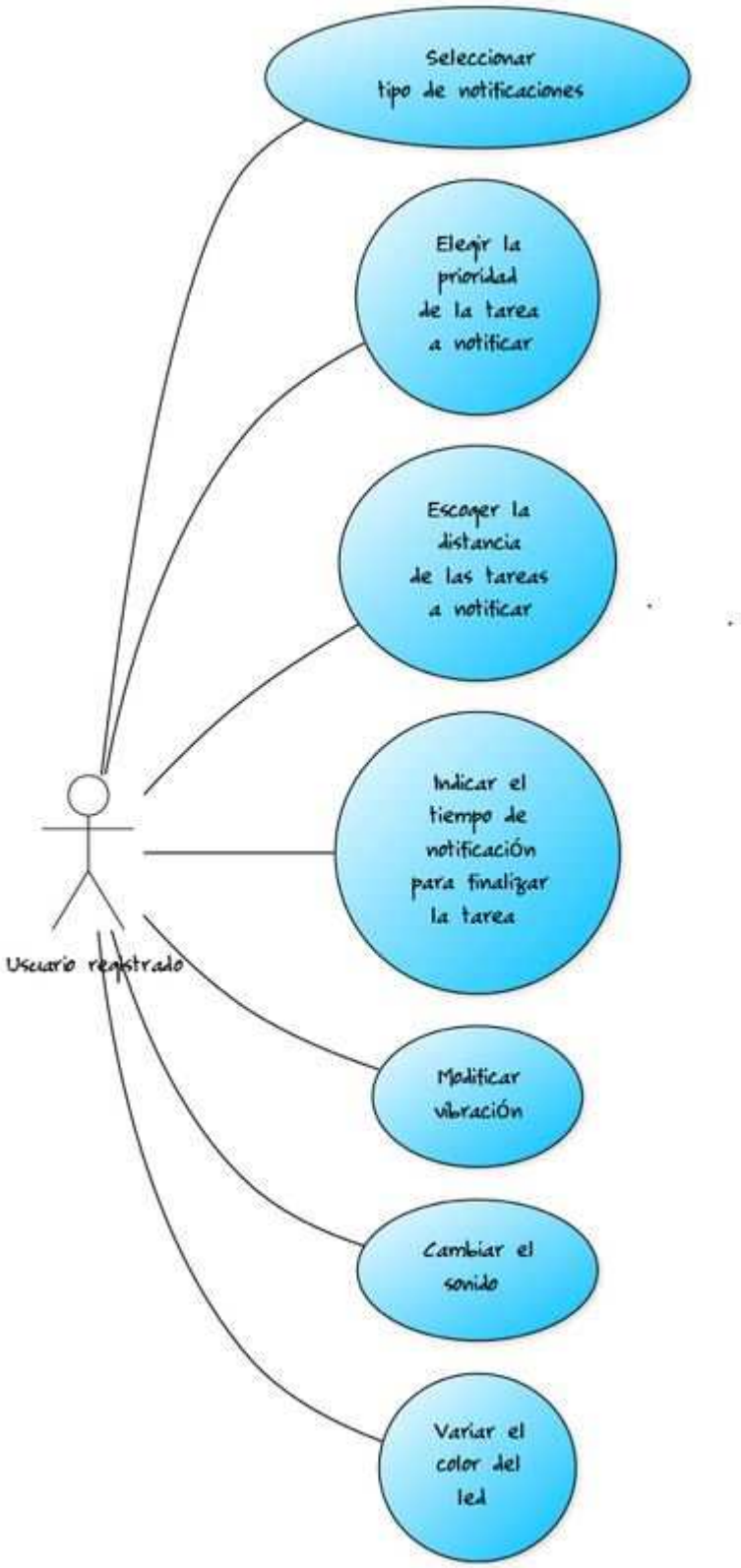


Ilustración 3 - Ajustes de las notificaciones

- *Seleccionar tipo de notificaciones*

Título:	Seleccionar tipo de notificaciones
Descripción	El usuario puede seleccionar el tipo de notificaciones que desea que se le notifiquen
Actor	Usuario registrado
Relaciones	
Precondición	
Comentarios	Existen tres opciones aunque todas ellas son por prioridad se pueden combinar por tiempo o cercanía. La última opción es elegir que las dos tipos de notificación estén activas a la vez

- *Elegir la prioridad de la tarea a notificar*

Título:	Elegir la prioridad de la tarea a notificar
Descripción	El usuario puede elegir qué tipo de tareas quieren que le notifiquen en función de su prioridad
Actor	Usuario registrado
Relaciones	
Precondición	
Comentarios	Opciones: Alta solo se notificaran las tareas con esta prioridad, Alta – Media se notificaran todas las tareas excepto las de prioridad baja y Todas se notificaran todas las tarea

- *Escoger la distancia de las tareas a notificar*

Titulo:	Escoger la distancia de las tareas a notificar
Descripción	El usuario puede escoger la distancia de las tareas que quiere que se le notifiquen
Actor	Usuario registrado
Relaciones	
Precondición	Tener seleccionada la opción de tipo de notificación de distancia o ambas
Comentarios	Opciones: 100 m, 200 m, 300 m, 500 m, 1 Km, 5 Km

- *Indicar el tiempo de notificación para finalizar la tarea*

Titulo:	Indicar el tiempo de notificación para finalizar la tarea
Descripción	El usuario puede indicar el tiempo de notificación que quiere que se le notifique la tarea, es decir, cuanto tiempo antes de que finalice la tarea quiere que se le avise
Actor	Usuario registrado
Relaciones	
Precondición	Tener seleccionada la opción de tipo de notificación de tiempo o ambas
Comentarios	Opciones: 30 min, 1 hora, 2 horas, 6 horas, 1 día, 2 dias

- *Modificar vibración*

Titulo:	Modificar vibración
Descripción	El usuario puede elegir si quiere que el móvil vibre o no cuando reciba una notificación
Actor	Usuario registrado
Relaciones	
Precondición	Tener seleccionada la opción de tipo de notificación de tiempo, distancia o ambas
Comentarios	

- *Cambiar sonido*

Titulo:	Cambiar el sonido
Descripción	El usuario puede seleccionar la melodía de sonido que sonara cuando le llegue una notificación
Actor	Usuario registrado
Relaciones	
Precondición	Tener seleccionada la opción de tipo de notificación de tiempo, distancia o ambas
Comentarios	

- *Variar el color del led*

Titulo:	Variar el color del led
Descripción	El usuario puede personalizar el color del led que se mostrara cuando reciba una notificación
Actor	Usuario registrado
Relaciones	
Precondición	Tener seleccionada la opción de tipo de notificación de tiempo, distancia o ambas
Comentarios	

- ***Gestión de puntos de interés***

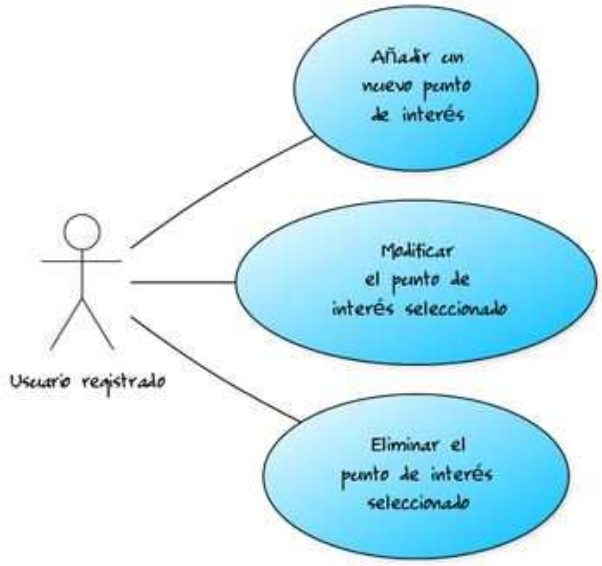


Ilustración 4 - Gestión de puntos de interés

o *Añadir un nuevo punto de interés*

Titulo:	Añadir un nuevo punto de interés
Descripción	El usuario tiene la opción de poder crear sus propios puntos de interés
Actor	Usuario registrado
Relaciones	
Precondición	
Comentarios	Cuando se vaya a crear el punto de interés se tiene la opción de poder elegir a que categoría pertenece entre las ofertadas en la aplicación

- *Modificar el punto de interés seleccionado*

Título:	Modificar el punto de interés seleccionado
Descripción	El usuario puede modificar el punto de interés que seleccione
Actor	Usuario registrado
Relaciones	
Precondición	Que el punto de interés haya sido creado previamente
Comentarios	Se puede modificar cualquier parámetro

- *Eliminar el punto de interés seleccionado*

Título:	Eliminar el punto de interés seleccionado
Descripción	El usuario puede elegir eliminar el punto de interés que tenga seleccionado
Actor	Usuario registrado
Relaciones	
Precondición	Que el punto de interés haya sido creado previamente
Comentarios	

- ***Gestión de tareas***

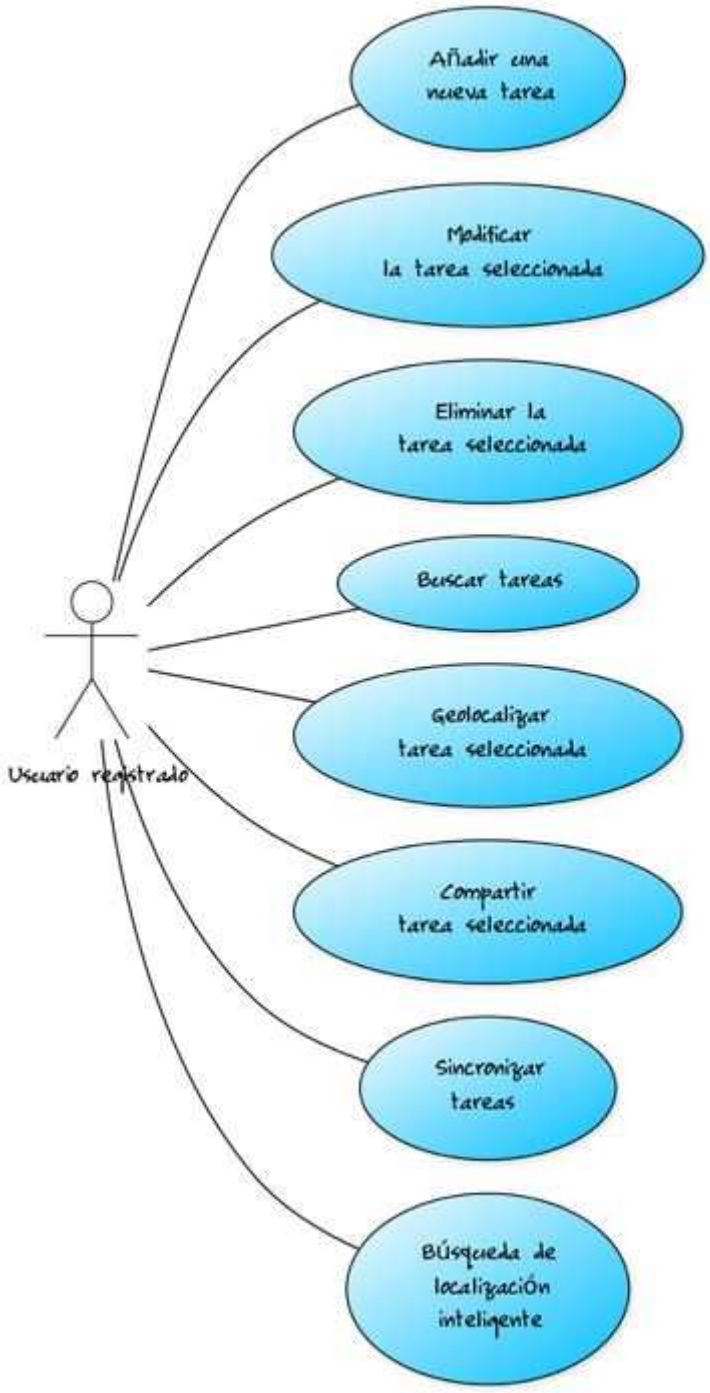


Ilustración 5 - Gestión de tareas

- *Añadir una nueva tarea*

Titulo:	Añadir una nueva tarea
Descripción	El usuario tiene la opción de poder crear su propia tarea
Actor	Usuario registrado
Relaciones	Sincronización con el servidor
Precondición	
Comentarios	Cuando se vaya a crear la tarea se tiene la opción de poder elegir a que categoría pertenece entre las ofertadas en la aplicación, la prioridad que debe de tener y si se le quiere asociar un punto de interés como localización favorita. Además, en cuanto la tarea sea creada automáticamente será sincronizada con el servidor

- *Modificar la tarea seleccionada*

Titulo:	Modificar la tarea seleccionada
Descripción	El usuario puede modificar la tarea que haya seleccionado
Actor	Usuario registrado
Relaciones	Sincronización con el servidor
Precondición	La tarea tiene que ha ver sido creada anteriormente
Comentarios	El usuario puede modificar cualquier parámetro. Una vez, el usuario haya terminado de modificar la tarea será sincronizada con el servidor

- *Eliminar la tarea seleccionada*

Titulo:	Eliminar la tarea seleccionada
Descripción	El usuario puede eliminar la tarea que haya seleccionado
Actor	Usuario registrado
Relaciones	Sincronización con el servidor
Precondición	La tarea tiene que ha ver sido creada anteriormente
Comentarios	En cuanto el usuario elimine la tarea esta será eliminada del servidor, si la tarea pertenece al gestor de Google Tasks será marcada como completada

- *Buscar tareas*

Titulo:	Buscar tareas
Descripción	El usuario tiene la opción de buscar tareas por título o descripción
Actor	Usuario registrado
Relaciones	Con el servidor
Precondición	Las tareas tienen que estar creadas en la aplicación, en caso de que la palabra introducida no tenga coincidencias se avisara al usuario con un mensaje
Comentarios	El usuario podrá indicar una palabra completa o simplemente letras que el servidor se encargara de buscar ese trozo de palabra entre los títulos y las descripciones de todas las tareas. Esta funcionalidad puede ser seleccionada desde el mapa como desde la gestión de tareas

- *Geo-localizar tarea seleccionada*

Título:	Geo-localizar tarea seleccionada
Descripción	El usuario tiene la opción de geo-localizar una tarea, es decir, cuando el usuario seleccione esta opción la tarea será mostrada en el mapa en su localización exacta
Actor	Usuario registrado
Relaciones	
Precondición	La tarea tiene que ha ver sido creada anteriormente
Comentarios	Si la opción de ajustes de tarea seleccionada se visualice esta activa, en cuanto se habrá el mapa y se muestre la tarea su localización principal lo hará de color morado y las de sus puntos de interés de color turquesa. En caso, contrario la tarea como sus puntos de interés estarán pintados en el color de la prioridad rojo (alta), amarillo (media), verde (baja). Esta funcionalidad puede ser seleccionada desde el mapa como desde la gestión de tareas

- *Compartir tarea seleccionada*

Titulo:	Compartir tarea seleccionada
Descripción	El usuario tiene la opción de poder compartir la tarea a través de sus aplicaciones de comunicación de su dispositivo
Actor	Usuario registrado
Relaciones	Con todas las aplicación de comunicación de su dispositivo
Precondición	La tarea tiene que ha ver sido creada anteriormente
Comentarios	

- *Sincronizar tareas*

Titulo:	Compartir tarea seleccionada
Descripción	Si el usuario selecciona esta función, todas sus tareas serán con el servidor
Actor	Usuario registrado
Relaciones	Sincronización con el servidor
Precondición	La tarea tiene que ha ver sido creada anteriormente
Comentarios	Esta funcionalidad puede ser seleccionada desde el mapa como desde la gestión de tareas

- *Búsqueda de localización inteligente*

Titulo:	Búsqueda de localización inteligente
Descripción	Cuando el usuario no añada una dirección o localización de la tarea, el sistema automáticamente le ofrecerá al usuario la dirección más cercana que haga referencia al título o descripción de la tarea
Actor	Usuario registrado
Relaciones	Sincronización con el servidor
Precondición	
Comentarios	Esta funcionalidad es posible gracias a la conexión con la API de Google Places

- ***Mapa***

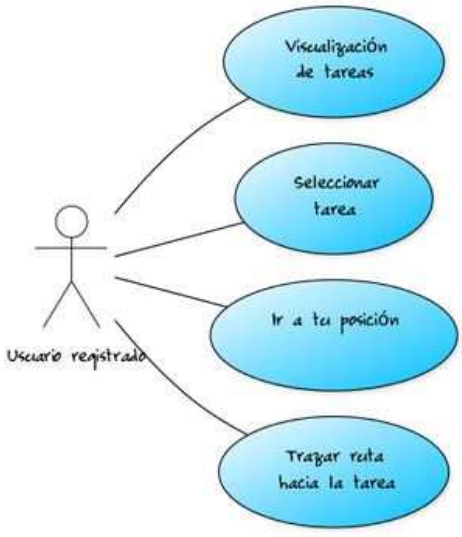


Ilustración 6 - Mapa

- *Visualización de tareas*

Titulo:	Visualización de tareas
Descripción	Cuando el usuario abre el mapa, en este podrá visualizar todas las tareas
Actor	Usuario registrado
Relaciones	
Precondición	Las tareas tienen que estar creadas anteriormente
Comentarios	Las tareas visualizadas se distinguen por su prioridad las que tengan prioridad alta estarán marcadas de color rojo, las de media de color amarillo y las bajas de color verde. En el caso de que se haya seleccionado una tarea desde el gestor de tareas esta aparecerá de color morado su posición principal y de turquesa sus puntos de interés

- *Seleccionar tarea*

Titulo:	Seleccionar tarea
Descripción	Si el usuario pulsa sobre una tarea esta le mostrara su titulo
Actor	Usuario registrado
Relaciones	
Precondición	La tarea tiene que existir
Comentarios	

- *Ir a tu posición*

Titulo:	Ir a tu posición
Descripción	Si el usuario pulsa sobre esta opción el mapa le llevara a su posición actual
Actor	Usuario registrado
Relaciones	
Precondición	El GPS tiene que estar activo
Comentarios	

- *Trazar la ruta hacia la tarea*

Titulo:	Trazar la ruta hacia la tarea
Descripción	Cuando el usuario pulse esta funcionalidad, se le trazara una ruta entre su posición y la tarea seleccionada
Actor	Usuario registrado
Relaciones	Google Maps
Precondición	La tarea tiene que existir
Comentarios	Se le mostrara al usuario una selección de tareas donde tendrá que seleccionar la tarea a la que desea ir

5.3 Requisitos no funcionales

5.3.1 Usabilidad

La navegación debe ser ágil, correcta e intuitiva, y tener una funcionalidad correctamente definida. Los elementos importantes deben quedar bien definidos, siguiendo los patrones de diseño habituales para temas de color, tamaño y fuentes. El diseño de los iconos debe ser claro e intuitivo, y la disposición de los componentes debe ser consistente en todas las pantallas.

5.3.2 Apariencia

Las pantallas y transacciones entre ellas deben tener el mismo patrón de diseño, igual que los elementos explicados en el párrafo anterior deben de resultar simples y agradables.

5.3.3 Adaptación al dispositivo

Todos los elementos gráficos deben de ser adaptables a los dispositivos donde vayan a ser visualizados. Designando los tamaños de objetos y textos en cantidades proporcionales, y ajustables a todas las densidades de pantalla del mercado.

5.3.4 Almacenamiento y persistencia

El ciclo de vida de las actividades debe ser gestionado correctamente, y los ajustes de configuración tienen que estar almacenados de forma correcta. La gestión de la base de datos debe tratarse de forma eficaz y segura, gestionando las diferentes incidencias que ocurran.

5.3.5 Robustez

La aplicación no debe finalizar de forma abrupta o inesperada, ni deben de saltar excepciones sin manejar. Los mensajes de error han de ser legibles y mostrar soluciones orientadas al usuario, si es posible.

5.3.6 Mantenimiento

El código debe de ser claro y mantenible para modificaciones y actualizaciones futuras. Se debe establecer todos los parámetros posibles en constantes para su correcto tratamiento y evitar errores.

5.3.7 Documentación

Se debe realizar una documentación precisa y usable que de soporte al mantenimiento del código. Esto incluye, al menos, los diseños de las bases de datos, los javadoc y phpdoc.

5.3.8 Otros requisitos

El consumo de la batería debe intentar ser lo más óptimo posible. La precisión del GPS debe ser lo más ajustada posible. El sistema de notificaciones no debe ser intrusivo. La aplicación y el mapa tienen que ser navegables con fluidez. El consumo de datos de la aplicación debe minimizarse lo máximo.

6. Planificación

En base a la toma de requisitos realizada por los dos integrantes del grupo en el apartado anterior se han definido las funcionalidades que contiene la aplicación. Además, se ha utilizado una técnica llamada MoSCoW para priorizar los requisitos que deberá contener la aplicación. Está basado en los casos de esta memoria, así como la de mi compañero, que cumplimentan todo el sistema. Como último punto de este apartado, se puede visualizar el Diagrama de Gantt el cual indica el proceso de tiempo requerido y el dedicado en cada caso para cumplir los requisitos definidos.

6.1 Definición de las funcionalidades

- ***Ajustes del mapa:*** Contiene varias opciones que permiten al usuario personalizar el mapa a su gusto, entre ellas se puede destacar la elección del algoritmo para la actualización del GPS, la elección del número de tareas a visualizar como el número de puntos de interés por cada tarea y si quiere que se destaque la tarea seleccionada con colores diferentes.

- ***Ajustes de las notificaciones:*** Permite al usuario elegir el tipo de notificaciones que quiere recibir, según el tipo de notificación que seleccione tendrá la opción de cambiar sus parámetros.

- ***Acceso a la aplicación:*** Incluye todas las acciones relacionadas con el registro e iniciar sesión en la aplicación.

- ***Sincronización con Google Task:*** Funcionalidad que permite al usuario iniciar y autorizar la sincronización con Google Tasks dando acceso al servidor a la lista que el usuario elija para sincronizar. En el caso de que el usuario quiera que el servidor suspenda la sincronización solo tendrá que desconectarse de Google Tasks. A parte, existe la opción de que el usuario pueda sincronizar manualmente sus tareas.

- ***Gestión de puntos de interés:*** Proporciona al usuario la posibilidad de crear, modificar y eliminar los puntos de interés favoritos del usuario, para después poder seleccionarlos como localización en la creación de sus tareas.

- ***Gestión de tareas:*** Posibilita al usuario la creación, modificación y eliminación de sus tareas. También, tiene las opciones de compartir, visualizar tarea en el mapa y buscar tareas. Cada tarea esta categorizada y tiene una prioridad asignada, estos dos parámetros son elegidos al gusto del usuario en función de las opciones que proporciona la aplicación. A

parte, la localización de la tarea puede ser cualquiera que elija el usuario o la de cualquier punto de interés que se haya creado anteriormente.

- ***Mapa:*** Es la pantalla principal de la aplicación, en ella se muestran todas las tareas del usuario con las opciones que haya marcado en ajustes del mapa. Cada tarea y sus puntos de interés si tiene asignados por categoría serán pintados del color de la prioridad de la tarea, en tal caso las tareas con prioridad alta serán pintadas de color rojo, las tareas con prioridad media de color amarillo y las de prioridad baja de color verde. En el caso de que la opción de la tarea seleccionada se muestre cuando se le invoque desde gestión de tareas esta cambiara su localización principal a morado y sus puntos de interés si tiene a turquesa. Además, según el tipo de algoritmo que el usuario haya seleccionado en ajustes del mapa la sincronización del mapa será automática, por distancia o por tiempo. En cualquiera de los casos, el usuario en función de lo que haya elegido cuando su posición cambie vera como en la aplicación se actualizarán las tareas que estén a su alrededor. A parte, el usuario podrá seleccionar el botón que le lleva a su posición, permitiéndole navegar por cualquier sitio del mapa y poder volver a donde esta él de forma rápida y eficaz.

6.2 MoSCoW

MoSCoW es una técnica para asignar diferentes prioridades a las funcionalidades de la aplicación. Esta técnica permite que el desarrollo de la aplicación sea más fácil y eficaz, gracias a dar prioridades a las diferentes funcionalidades el programador puede orientarse de cuáles son las más importantes y que más relevancia tienen para que la aplicación sea funcional. Esta técnica se ha basado en los requisitos funcionales explicados en el apartado anterior. A continuación, se presenta una guía que le ayudara a entender el significado de cada carta:

- ***M – MUST (Debe de tener…):*** Define los requisitos fundamentales para el éxito del proyecto.
- ***S – SHOULD (Debería tener…):*** Representa un punto de alta prioridad que se debe incluir en la solución si es posible. Esto es a menudo un requisito crítico pero que puede estar satisfecho de otras maneras si es estrictamente necesario.
- ***C – COULD (Pueden tener…):*** Describe un requisito que se considera deseable, pero no necesario. Este será incluido si el tiempo y los recursos lo permiten.

- ***W - WON'T (No tendrá… por el momento):*** Detalla los requisitos que pueden ser pospuestos, pero que son buenos y se podrían incluir en el futuro.

A continuación, se detalla la prioridad de los requisitos de la aplicación en función a los casos de uso definidos en el apartado de Especificación de requisitos.

Tabla 1 - MoSCoW

Casos uso	Prioridad
Acceso a la aplicación	
• Registro	Must
• Iniciar sesión	Must
Ajustes del mapa	
• Seleccionar tipo de actualización del GPS	Should
• Cambiar el tiempo de actualización del GPS	Could
• Modificar la distancia de actualización del GPS	Could
• Escoger el número de tareas a visualizar	Should
• Elegir número de puntos de interés a visualizar por cada tarea	Should
• Destacar tarea seleccionada	Must
Ajustes de las notificaciones	
• Seleccionar tipo de notificaciones	Must
• Elegir la prioridad de la tarea a notificar	Should
• Escoger la distancia de las tareas a notificar	Should
• Indicar el tiempo de notificación para finalizar la tarea	Should
• Modificar vibración de la notificación	Won't
• Cambiar el sonido de la notificación	Won't
• Variar el color del led	Won't

Sincronización con Google Tasks	
• Iniciar session con Google Tasks	Must
• Elegir la lista de tareas a descargar	Must
• Sincronizar con Google Tasks manualmente	Could
• Desconectar de Google Tasks	Must
Gestión de puntos de interés	
• Añadir un nuevo punto de interés	Must
• Modificar el punto de interés seleccionado	Should
• Eliminar el punto de interés seleccionado	Should
Gestión de tareas	
• Añadir una nueva tarea	Must
• Modificar la tarea seleccionada	Must
• Eliminar la tarea seleccionada	Must
• Buscar tareas	Could
• Geo-localizar tarea seleccionada	Must
• Compartir tarea seleccionada	Could
• Sincronizar tareas	Must
• Busqueda de localización inteligente	Could
Mapa	
• Visualización de tareas	Must
• Seleccionar tarea	Could
• Ir a tu posición	Should
• Trazar ruta hacia la tarea	Won´t

6.3 Esquema cronológico

Como resultado de todo el análisis que se ha llevado a cabo en los apartados anteriores y en la memoria complementaria de mi compañero, se han

planificado unas fases y el tiempo que llevarían realizarlas. Para representar este esquema se ha utilizado el Diagrama de Gantt, el cual es una representación gráfica y simultánea de la planificación como de la programación del proyecto. Esto permite que se visualice de forma fácil y rápida toda la planificación y programación de las fases de un solo vistazo.

En la primera Ilustración se puede ver las diferentes fases que han constituido el proyecto, a quien están asignadas, el tanto porciento realizado y la fecha de inicio y fin.

	Nombre de la tarea	Fecha de i	Fecha de fin	Duración	Predecesoras	% Completo	Asignado a
1	**Proyecto para geolocalización de tareas pendientes**	**12/02/14**	**23/08/14**	**193**		**98%**	
2	**Reunir requisitos**	12/02/14	25/02/14	14		1	
3	Toma de requisitos	12/02/14	17/02/14	6		100%	Ambos
4	Planificación	20/02/14	25/02/14	6		100%	Ambos
5	**Diseño**	**28/02/14**	**01/04/14**	**33**		1	
6	La interfaz grafica	28/02/14	15/03/14	16		100%	Emanuel
7	De las bases de datos	02/03/14	17/03/14	16		100%	Jessica
8	La aplicación móvil	18/03/14	22/03/14	5	7	100%	Jessica
9	Del servidor	22/03/14	01/04/14	11		100%	Emanuel
10	**Implementación**	05/04/14	26/07/14	113		1	
11	Realización de los requisitos MUST	05/04/14	20/05/14	46		100%	Ambos
12	Realización de los requisitos SHOULD	07/06/14	27/06/14	21		100%	Ambos
13	Realización de los requisitos COULD	05/07/14	26/07/14	22		100%	Ambos
14	Realización de los requistos WON'T					0%	Ambos
15	**Pruebas**	05/08/14	23/08/14	19		85%	Ambos

Ilustración 7 - Planificación 1

A continuación, se mostraran dos Ilustraciones que representan el tiempo que estaba planificado realizarlas.

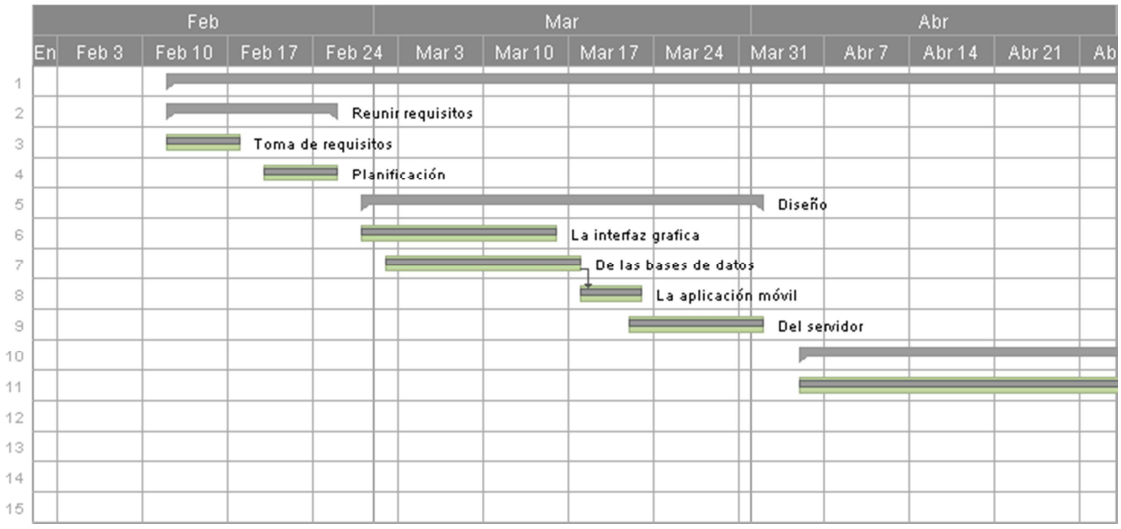


Ilustración 8 - Planificación 2

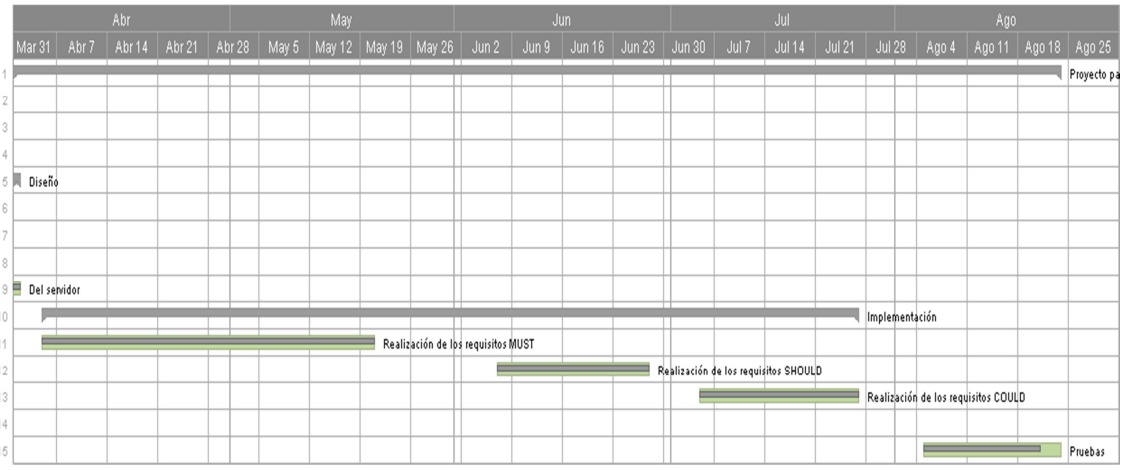


Ilustración 9 - Planificación 3

7. Diseño

En este apartado se describe el modelo de datos del sistema. La interfaz gráfica se encuentra definida en el documento complementario de mi compañero.

7.1 Modelo de datos

El modelo de datos que contiene la aplicación y el servidor. Primero explicaremos las semejanzas y discrepancias que hay entre los objetos y sus parámetros:

- ***Task:*** Objeto que contiene toda la información de las tareas, tales como el título, la dirección, sus coordenadas de localización, la fecha cuando se modificó por última vez, la fecha y hora a la que se tiene que realizar, una descripción y la prioridad que tiene asignada. Aquí se almacenaran todas las tareas de usuario ya se han creadas desde la propia aplicación o provengan de Google Task. En el caso del servidor este objeto contiene un dato más que hace referencia a la id que contiene en su lista de tareas de Google Task.

- ***Category:*** Objeto compuesto por los datos de cada categoría, entre ellos se puede encontrar su nombre, el id a la categoría que pertenece en el caso de que sea subcategoría, la imagen que tiene asociada y la última fecha que se modificó. En este objeto se almacenan todas las categorías necesarias para que el usuario pueda categorizar sus tareas como sus puntos de interés.

- ***POI:*** Objeto donde se guardan los datos de todos los puntos de interés, está formado por un título, las coordenadas de su posición, la dirección, su última fecha de modificación y su descripción. En el caso de la aplicación se

almacenaran los puntos de interés genéricos que serán descargados des servidor como los creados por el usuario. En cambio, en el servidor solo estarán contenidos los datos de los puntos de interés genéricos.

- ***User:*** Objeto formado por los parámetros del usuario los cuales son el nombre, el correo electrónico, su password_hash que es su contraseña cifrada para mantener la seguridad de su cuenta, su api_key que lo identifica inequívocamente en servidor, created_at que es la fecha de registro, sus coordenadas de localización iniciales y la imagen de su perfil.

- ***User_googletask_list:*** Objeto donde se almacenan los datos de las listas de tareas que están contenidas en Google Tasks, entre sus parámetros se encuentran el identificador al usuario a quien pertenece la lista, el nombre de la lista, el identificador de la lista en Google Tasks, access_token perteneciente al usuario para poder acceder a Google Tasks, el refresh _token que permite mantener al usuario conectado a Google Tasks sin pedirle permisos cada vez, el refresh_frequency es el minuto cuando se va lanzar el crontab que sincronizará las tareas automáticamente y isUploadable es el campo que indica al servidor si esta tarea tiene que ser subida a Google Tasks. Este objeto solo está en el servidor disponible ya que es el encargado de almacenar todos los datos correspondientes con la sincronización de Google Tasks.

  A continuación, se van a mostrar la entidad de relación de cada una, con los nombres de los parámetros explicados anteriormente.

1. Servidor

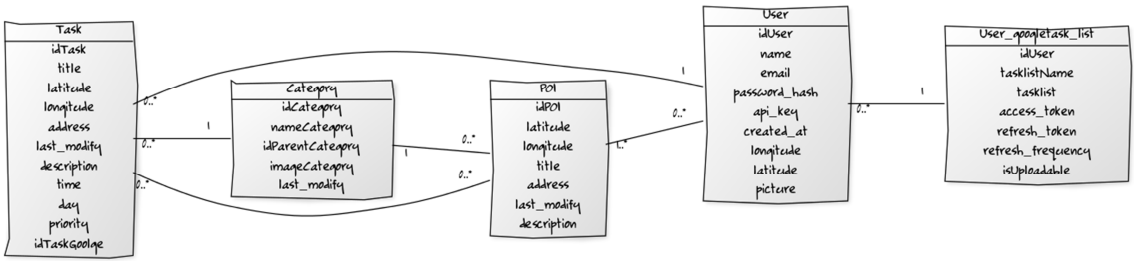


Ilustración 10 – Servidor

2. *Aplicación*

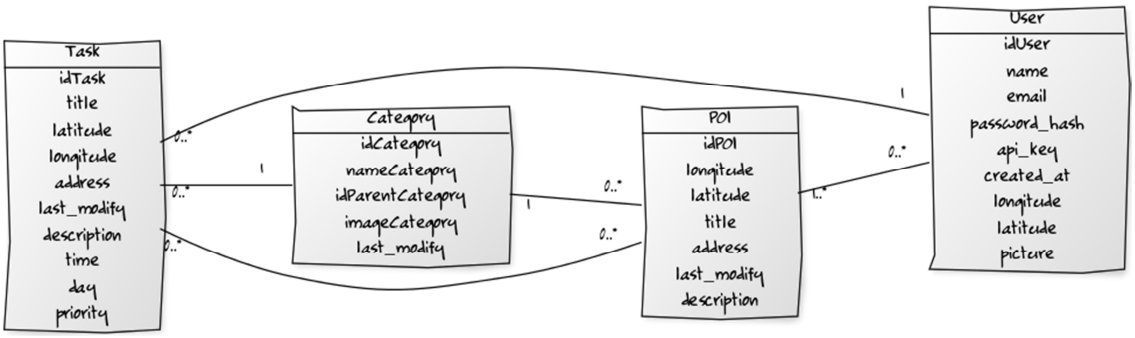


Ilustración 11 – Aplicación

8. Implementación

Este apartado va exponer la implementación que ha supuesto la creación de la aplicación. El desarrollo del servidor se especifica en la memoria de mi compañero. En los dos documentos se van a definir las clases y métodos principales que hacen posible la conexión entre ambos, la geo-localización de las tareas en la aplicación, la sincronización con Google Tasks, las notificaciones al usuario y el resto de funcionalidades que contiene la aplicación.

8.1 Aplicación

En este apartado se dispone a explicar la organización de la aplicación móvil. Se ha diseñado e implementando siguiendo un modelo estructurado de tres capas que separan la persistencia, la lógica y las interfaces, siguiendo una arquitectura orientada a objetos. Se ha diseñado así pensando en el mantenimiento y futuras ampliaciones modulares de este software.

La aplicación esta implementada por los dos integrantes del grupo pero esta memoria solo se define la mi parte. En la siguiente Ilustración se puede observar tal estructuración, como se puede visualizar el proyecto tiene dos partes: la principal donde se implementan todas las clases que nos proporcionan las funcionalidades descritas, más el paquete res donde se almacena toda la capa visual, de idiomas, preferencias,…. Esta última ha sido diseñada completamente por el integrante del grupo Emanuel Alloza Álvarez, la implementación ha sido en conjunto pero la gran mayoría es aportación de Jessica Carpintero Macián:

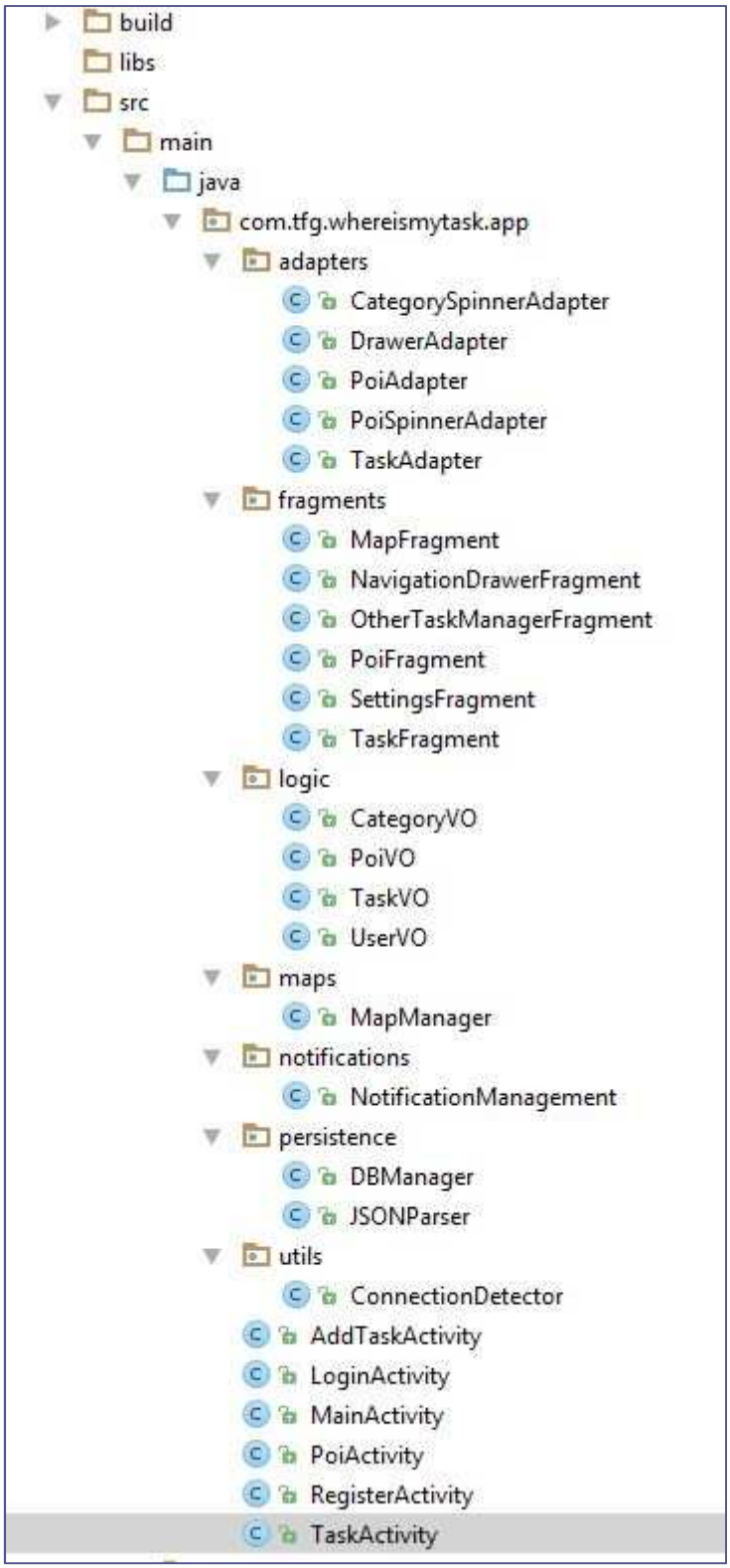


Ilustración 12 - Estructura

Una vez vista la estructuración en la Ilustración se va a proceder a explicar los diferentes paquetes que se visualizan en ella:

8.1.1 Adapters: Paquete realizado para poder personalizar los despegables de los objetos POI y Category, las listas que contienen los contenedores de las pantallas de gestión de tarea y punto de interés, y el menú principal donde se tiene acceso a todas las pantallas. En la siguiente ilustración se muestra un ejemplo de cómo se trabaja la clase encargada de visualizar correctamente los despegables.

```
public class CategorySpinnerAdapter extends BaseAdapter {

    List<CategoryVO> categoryList;
    Context mContext;

    public CategorySpinnerAdapter(Context context, List<CategoryVO> categoryList) {
        this.categoryList = categoryList;
        mContext = context;
    }

    @Override
    public int getCount() { return categoryList.size(); }

    @Override
    public CategoryVO getItem(int position) { return categoryList.get(position); }

    @Override
    public long getItemId(int position) { return position; }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        View rowView = convertView;
        if (rowView == null) {
            LayoutInflater inflater = (LayoutInflater) mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            rowView = inflater.inflate(R.layout.spinneritem_category, null);
        }

        rowView.setEnabled(true);

        CategoryVO category =  getItem(position);

        if (category != null) {

            TextView list_item_diet_name = (TextView) rowView.findViewById(R.id.list_item_category_name);

            list_item_diet_name.setText(category.getName());
        }

        return rowView;
    }
}
```

Ilustración 13 – Adaptador del despegable de la categoría

En la siguiente Ilustración se muestra el ejemplo de la lista de los gestores. Se añade el ejemplo de tarea porque es el más completo.

```java
/**
 * Method that displays the information in the task list
 * @param position
 * @param convertView
 * @param parent
 * @return
 */
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    View result = convertView;
    if (result == null) {
        result = inflater.inflate(R.layout.grid_item, parent, false);
    }

    // Try to get view cache or create a new one if needed
    ViewCache viewCache = (ViewCache) result.getTag();
    if (viewCache == null) {
        viewCache = new ViewCache(result);
        result.setTag(viewCache);
    }

    // Fetch item
    TaskVO task = getItem(position);

    // Bind the data
    viewCache.mTitleView.setText(task.getTitle());
    viewCache.mSubtitleView.setText(task.getDescription());
    viewCache.mDateView.setText(task.getDay().substring(8,10) + "-" + task.getDay().substring(5,7) +
            "-" + task.getDay().substring(0,4) + " " + task.getTime().substring(0,5));

    return result;
}

/**
 * Method encagado loading the new list and notify
 * @param newlist
 */
public void updateList(ArrayList<TaskVO> newlist) {
    allTask.clear();
    allTask.addAll(newlist);
    this.notifyDataSetChanged();
}
```

Ilustración 14 – Adaptador de la lista de tareas

En la última Ilustración se contempla la clase que realiza la personalización del menú permitiendo añadirle imágenes y texto diferentes a los nombres de las pantallas a seleccionar.

```java
/**
 * Responsible method to customize the main menu
 * @param position
 * @param convertView
 * @param parent
 * @return
 */
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    LayoutInflater inflater = (LayoutInflater) context
            .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    View rowView = inflater.inflate(R.layout.fragment_navigation_row, parent, false);
    TextView textView = (TextView) rowView.findViewById(R.id.label);
    ImageView imageView = (ImageView) rowView.findViewById(R.id.icon);
    textView.setText(values[position]);

    switch (position) {
        case 0:
            imageView.setImageResource(android.R.drawable.ic_menu_mapmode);
            break;
        case 1:
            imageView.setImageResource(android.R.drawable.ic_menu_month);
            break;
        case 2:
            imageView.setImageResource(android.R.drawable.ic_menu_myplaces);
            break;
        case 3:
            imageView.setImageResource(android.R.drawable.ic_menu_today);
            break;
        case 4:
            imageView.setImageResource(android.R.drawable.ic_menu_preferences);
            break;

        default:
            imageView.setImageResource(R.drawable.logo_whereismytask);
            break;
    }

    return rowView;
}
```

Ilustración 15 – Adaptador del menú principal

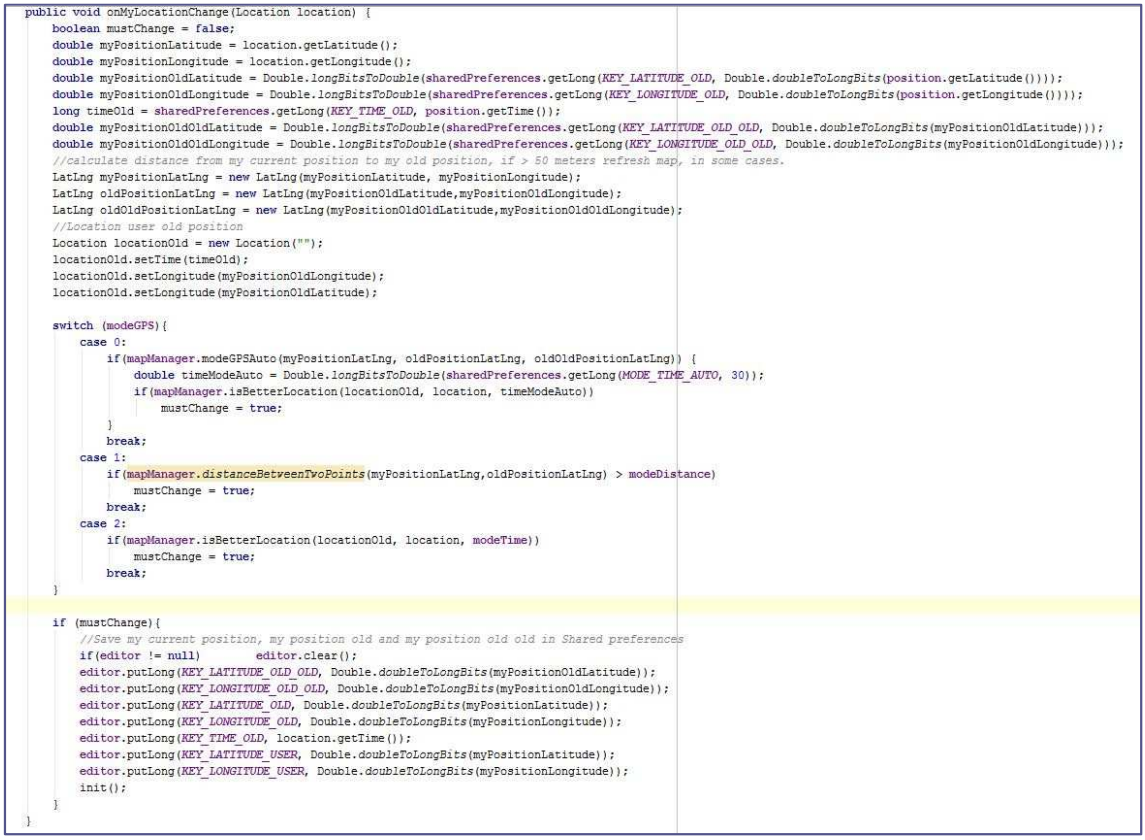
8.1.2 Fragments: Carpeta donde se ha almacenado la implementación de tres de las cuatro clases que componen la aplicación en el menú principal, la cuarta ha sido implementada y explicada en la memoria de mi compañero. Cada clase es la encargada de pintar y realizar todas las acciones necesarias para poder mostrar por al usuario las pantallas con sus funcionalidades.

8.1.2.1 *MapFragment:* es la encargada de pintar el mapa en la pantalla con todas las tareas y sus puntos de interés, para realizar esta funcionalidad se apoya en la clase MapManager que será la que de verdad pinte los puntos en el mapa pero será explicada más adelante. En la Ilustración siguiente puede observar la implantación del método init() que es el encargado de inicializar y guardar los datos principales de la pantalla.

```
private View init(){
    mapManager = new MapManager(mapView.getMap(), getActivity().getApplicationContext());
    // Getting LocationManager object from System Service LOCATION_SERVICE
    LocationManager locationManager = (LocationManager) this.getActivity().getSystemService(getActivity().LOCATION_SERVICE);
    Bundle extras = getArguments();
    dbManager = dbManager.getInstance(this.getActivity());
    taskList = new ArrayList<TaskVO>();
    sharedPreferences = getActivity().getSharedPreferences(PREFERENCES_FILE, Context.MODE_PRIVATE);
    editor = getActivity().getSharedPreferences(PREFERENCES_FILE, Context.MODE_PRIVATE).edit();
    modeGPS = 0;
    loadPreferencesSettings();
    if (!locationManager.isProviderEnabled( LocationManager.GPS_PROVIDER ) &&
            !locationManager.isProviderEnabled( LocationManager.NETWORK_PROVIDER )
            && !locationManager.isProviderEnabled( LocationManager.PASSIVE_PROVIDER ) ) {
        if(!dialogNoGpsShowed) alertNoGps();
    }

    position = mapManager.obtainMyPosition();

    //Get values of SharedPreferences
    mapView.getMap().setMapType(sharedPreferences.getInt(KEY_MAP_TYPE, 1));

    //Obtain list tasks
    taskList = dbManager.getAllTaskOrderPriority();

    //Get values of getArguments from NotificationManager
    if (extras != null){
        idTask = extras.getInt(KEY_ID_TASK, 0);
        latitude = extras.getDouble(KEY_LATITUDE, Double.doubleToLongBits(position.getLatitude()));
        longitude = extras.getDouble(KEY_LONGITUDE, Double.doubleToLongBits(position.getLongitude()));
    } else {
        //Obtain idTask of task selected from TaskActivity
        idTask = 0;
        latitude = position.getLatitude();
        longitude = position.getLongitude();
    }
    //Obtain initial position
    LatLng myPosition = new LatLng(latitude,longitude);
    mapManager.goToPosition(myPosition,sharedPreferences.getFloat(KEY_ZOOM_LEVEL, 16));

    //initializing settings
    initializeOptionsSettings();
    idTask = mapManager.addAllTask(taskList, idTask);
    saveMapView(mapView);
    mapView.getMap().setOnMyLocationChangeListener(this);
    return rootView;
}
```

Ilustración 16 – Inicialización de los parámetros del mapa

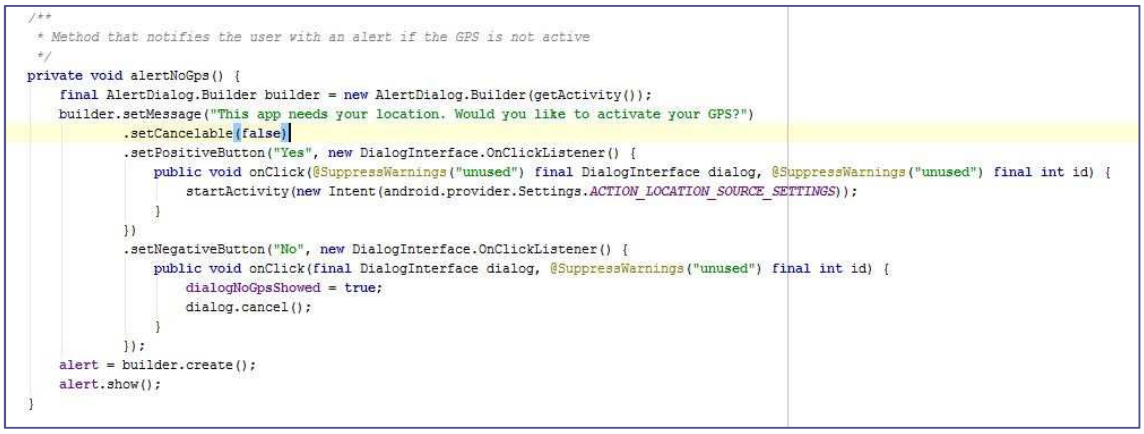
En la Ilustración de abajo se puede ver el método que en todo momento está pendiente del cambio de posición del usuario que, en función de los parámetros elegidos en ajustes, se adecúe al algoritmo seleccionado.



```
public void onMyLocationChange(Location location) {
    boolean mustChange = false;
    double myPositionLatitude = location.getLatitude();
    double myPositionLongitude = location.getLongitude();
    double myPositionOldLatitude = Double.longBitsToDouble(sharedPreferences.getLong(KEY_LATITUDE_OLD, Double.doubleToLongBits(position.getLatitude())));
    double myPositionOldLongitude = Double.longBitsToDouble(sharedPreferences.getLong(KEY_LONGITUDE_OLD, Double.doubleToLongBits(position.getLongitude())));
    long timeOld = sharedPreferences.getLong(KEY_TIME_OLD, position.getTime());
    double myPositionOldOldLatitude = Double.longBitsToDouble(sharedPreferences.getLong(KEY_LATITUDE_OLD_OLD, Double.doubleToLongBits(myPositionOldLatitude)));
    double myPositionOldOldLongitude = Double.longBitsToDouble(sharedPreferences.getLong(KEY_LONGITUDE_OLD_OLD, Double.doubleToLongBits(myPositionOldLongitude)));
    //calculate distance from my current position to my old position, if > 50 meters refresh map, in some cases.
    LatLng myPositionLatLng = new LatLng(myPositionLatitude, myPositionLongitude);
    LatLng oldPositionLatLng = new LatLng(myPositionOldLatitude,myPositionOldLongitude);
    LatLng oldOldPositionLatLng = new LatLng(myPositionOldOldLatitude,myPositionOldOldLongitude);
    //Location user old position
    Location locationOld = new Location("");
    locationOld.setTime(timeOld);
    locationOld.setLongitude(myPositionOldLongitude);
    locationOld.setLongitude(myPositionOldLatitude);

    switch (modeGPS){
        case 0:
            if(mapManager.modeGPSAuto(myPositionLatLng, oldPositionLatLng, oldOldPositionLatLng)) {
                double timeModeAuto = Double.longBitsToDouble(sharedPreferences.getLong(MODE_TIME_AUTO, 30));
                if(mapManager.isBetterLocation(locationOld, location, timeModeAuto))
                    mustChange = true;
            }
            break;
        case 1:
            if(mapManager.distanceBetweenTwoPoints(myPositionLatLng,oldPositionLatLng) > modeDistance)
                mustChange = true;
            break;
        case 2:
            if(mapManager.isBetterLocation(locationOld, location, modeTime))
                mustChange = true;
            break;
    }

    if (mustChange){
        //Save my current position, my position old and my position old old in Shared preferences
        if(editor != null)        editor.clear();
        editor.putLong(KEY_LATITUDE_OLD_OLD, Double.doubleToLongBits(myPositionOldLatitude));
        editor.putLong(KEY_LONGITUDE_OLD_OLD, Double.doubleToLongBits(myPositionOldLongitude));
        editor.putLong(KEY_LATITUDE_OLD, Double.doubleToLongBits(myPositionLatitude));
        editor.putLong(KEY_LONGITUDE_OLD, Double.doubleToLongBits(myPositionLongitude));
        editor.putLong(KEY_TIME_OLD, location.getTime());
        editor.putLong(KEY_LATITUDE_USER, Double.doubleToLongBits(myPositionLatitude));
        editor.putLong(KEY_LONGITUDE_USER, Double.doubleToLongBits(myPositionLongitude));
        init();
    }
}
```

Ilustración 17 - Repintar en mapa

En la próxima Ilustración se puede ver el método que avisara al usuario cuando es requerido el uso del GPS. Que ira en función del algoritmo arriba explicado.



```
/**
 * Method that notifies the user with an alert if the GPS is not active
 */
private void alertNoGps() {
    final AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    builder.setMessage("This app needs your location. Would you like to activate your GPS?")
            .setCancelable(false)
            .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                public void onClick(@SuppressWarnings("unused") final DialogInterface dialog, @SuppressWarnings("unused") final int id) {
                    startActivity(new Intent(android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS));
                }
            })
            .setNegativeButton("No", new DialogInterface.OnClickListener() {
                public void onClick(final DialogInterface dialog, @SuppressWarnings("unused") final int id) {
                    dialogNoGpsShowed = true;
                    dialog.cancel();
                }
            });
    alert = builder.create();
    alert.show();
}
```

Ilustración 18 - Alerta apagado el GPS

Y en la última Ilustración se puede observar el método encargado de cargar las preferencias de ajuste del usuario necesarias para el uso de los métodos explicados arriba.

```
/**
 * Load user preferences
 */
public void loadPreferencesSettings(){
    // Give the system preferences
    SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences(getActivity());

    modeGPS = Integer.parseInt(pref.getString(MODE_GPS, "0"));
    modeTime = Double.longBitsToDouble(pref.getLong(MODE_TIME, 0));
    modeDistance = Integer.parseInt(pref.getString(MODE_DISTANCE, "0"));
}
```

Ilustración 19 - Carga de preferencias del usuario

8.1.2.2 *NavigationDrawerFragment:* Clase encargada de gestionar el funcionamiento del menú principal de la aplicación. En la Ilustración de abajo se puede visualizar como se cargan los diferentes nombres de las pantallas.

```
/**
 * Method responsible for managing the menu that organizes the fragments
 * @param inflater
 * @param container
 * @param savedInstanceState
 * @return
 */
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_navigation_drawer, container, false);
    drawerListView = (ListView) rootView.findViewById(R.id.list_layout);

    drawerListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            selectItem(position);
        }
    });

    drawerListView.setAdapter(new DrawerAdapter(getActionBar().getThemedContext(), new String[]{
            getString(R.string.title_section1),
            getString(R.string.title_section2),
            getString(R.string.title_section3),
            getString(R.string.title_section4),
            getString(R.string.action_settings),
    }));

    drawerListView.setItemChecked(currentSelectedPosition, true);
return rootView;
}

public boolean isDrawerOpen() {
    return drawerLayout != null && drawerLayout.isDrawerOpen(fragmentContainerView);
}
```

Ilustración 20 - Inicialización y carga de los parámetros del menú principal

8.1.2.3 *PoiFragment*: Clase encargada de visualizar la lista de punto de interés del usuario. En la ilustración siguiente se puede observar cómo se recogen los datos y se introducen al objeto que luego los mostrara por pantalla.

```
/**
 * Method responsible for creating the fragment with the data
 * @param inflater
 * @param container
 * @param savedInstanceState
 * @return
 */
@Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Set fragment_poi.xml to be the layout for this fragment
        View rootView = inflater.inflate(R.layout.fragment_poi, container, false);

        // Find the {@link GridView} that was already defined in the XML layout
        gridView = (GridView) rootView.findViewById(R.id.gridPoi);

        dbManager = dbManager.getInstance(this.getActivity());

        Bundle extras = getArguments();
        if(null != extras){
            poiList = extras.getParcelableArrayList(POILIST);
        }

        if (poiList == null || poiList.isEmpty()) {
            poiList = new ArrayList<PoiVO>();
            poiList = dbManager.getAllPoi();
        }

        poiAdapter = new PoiAdapter(inflater, poiList);
        gridView.setOnItemClickListener(this);

        return rootView;
    }
```

Ilustración 21 - Inicialización de la pantalla de los puntos de interés

También, es la encargada abrir la actividad que permitirá visualizar y modificar al usuario toda la información que contiene el punto de interés. En la Ilustración se puede ver como se ha implementado esta funcionalidad.

```
/**
 * Callback method for a when a poi is clicked. A new intent is created with the
 * poi object.
 */
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    // Find poi that was clicked based off of position in adapter
    PoiVO poi = (PoiVO) parent.getItemAtPosition(position);
    if(poi != null) {
        Intent i = new Intent(getActivity(), PoiActivity.class);
        i.putExtra(POI, poi);
        startActivity(i);
    } else {
        Toast.makeText(getActivity(), R.string.noPois, Toast.LENGTH_LONG).show();
    }
}
```

Ilustración 22 - Seleccionar un punto de interés

8.1.2.4 *TaskFragment*: Aquí se gestiona la visualización de las tareas, esta clase mostrara una lista de las tareas. En la Ilustración se puede observar cómo se realiza la carga de los datos recogiendo los valores que proceden de la clase principal o de la base de datos en caso de no recibirlos. Además, se puede ver como se carga las tareas en la lista que los mostrara por pantalla.

```
/**
 * Method responsible for creating the fragment with the data
 * @param inflater
 * @param container
 * @param savedInstanceState
 * @return
 */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Set fragment_task.xml to be the layout for this fragment
        View rootView = inflater.inflate(R.layout.fragment_task, container, false);

        // Find the {@link GridView} that was already defined in the XML layout
        gridView = (GridView) rootView.findViewById(R.id.gridTask);

        dbManager = dbManager.getInstance(this.getActivity());

        Bundle extras = getArguments();
        if(null != extras){
            taskList = new ArrayList<TaskVO>();
            taskList = extras.getParcelableArrayList("taskList");
        }

        if (taskList == null || taskList.isEmpty()) {
            taskList = new ArrayList<TaskVO>();
            taskList = dbManager.getAllTask();
        }

        taskAdapter = new TaskAdapter(inflater, taskList);

        gridView.setOnItemClickListener(this);

        return rootView;
    }
```

Ilustración 23 - inicializar y cargar las tareas en la pantalla

A parte el usuario podrá seleccionar cualquiera de ellas para que se le muestre en una actividad todos sus datos. En la Ilustración se ve los métodos que hacen posibles esta funcionalidad.

```
/**
 * Callback method for a when a task is clicked. A new share intent is created with the
 * task title.
 *
 * @param parent The AdapterView where the click happened.
 * @param view The view within the AdapterView that was clicked (this
 *            will be a view provided by the adapter).
 * @param position The position of the view in the adapter.
 * @param id The row id of the item that was clicked.
 */
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    // Find task that was clicked based off of position in adapter
    TaskVO task = (TaskVO) parent.getItemAtPosition(position);

    if(task != null) {
        Intent i = new Intent(getActivity(), TaskActivity.class);
        i.putExtra("task", task);
        startActivity(i);
    } else {
        Toast.makeText(getActivity(), "ERROR", Toast.LENGTH_LONG).show();
    }
}
```

Ilustración 24 - Selección de una tarea

8.1.2.5 *SettingsFragment*: Clase encargada de gestionar las preferencias del usuario de forma automática. En la Ilustración se visualizara el constructor que se le asigna a la pantalla.

```
public class SettingsFragment extends PreferenceFragment{

    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
    }

    @Override
    public void onResume() {
        super.onResume();
    }

}
```

Ilustración 25 - Ajustes

8.1.3 ***Logic:*** Paquete donde se encuentran todos los objetos de la aplicación.

8.1.3.1 *CategoryVO*: Objeto que contendrá una categoría, dentro de él se almacenaran los valores de id categoría, nombre, el id del padre, la imagen que corresponda a la categoría y su fecha de última modificación. En la Ilustración se pueden visualizar estos parámetros con su constructor.

```
/**
 * Attributes
 */
private int idCategory;
private String name;
private int idParentCategory;
private String imageCategory;
private String last_modify;


/**
 * @param name
 * @param idParentCategory
 */
public CategoryVO(String name, int idParentCategory, String imageCategory, String last_modify) {
    super();
    this.name = name;
    this.idParentCategory = idParentCategory;
    this.imageCategory = imageCategory;
    this.last_modify = last_modify;
}
```

Ilustración 26 - Atributos y constructor de categorías

8.1.3.2 *PoiVO*: Objeto que almacenara los parámetros de un punto de interés. Un ejemplo de alguno de sus parámetros son su id, titulo, descripción, dirección, localización, etc. En la Ilustración se puede observar el resto de campos.

```
/**
 * Attributes
 */
 /** ID of POI. */
private int idPOI;

/** Title of POI. */
private String title;

/** Description of POI. */
private String description;

/** Address of POI. */
private String address;

/** last_modify of POI. */
private String lastModify;

/** Longitude and latitude of POI. */
private Location location;

/** Category  of POI */
private int idCategory;

/**idUser*/
private int idUser;

/** Distance between poi and user */
private double distance;
```

Ilustración 27 - Atributos de POI

En la siguiente Ilustración se puede divisar sus constructores en este caso contiene dos el que le pasas todos los parámetros y el que nos obliga a implementar la clase Parceable necesaria para poder pasear los datos y poder pasarlos entre clases a través del Bundle.

```
/**
 *
 * @param title
 * @param description
 * @param address
 * @param lastModify
 * @param idCategory
 * @param idUser
 */
public PoiVO(String title, String description, String address,
        Location location, int idCategory,
        String lastModify, int idUser) {
    super();
    this.title = title;
    this.description = description;
    this.address = address;
    this.location = location;
    this.lastModify = lastModify;
    this.idCategory = idCategory;
    this.idUser = idUser;
}

private PoiVO(Parcel parcel) {
    this.idPOI = parcel.readInt();
    this.title = parcel.readString();
    this.location = Location.CREATOR.createFromParcel(parcel);
    this.address = parcel.readString();
    this.lastModify = parcel.readString();
    this.description = parcel.readString();
    this.idCategory  = parcel.readInt();
    this.idUser  = parcel.readInt();
}
```

Además del constructor especial implementar la clase Parceable es necesario sobrecargar los métodos que se visualizan en la siguiente Ilustración.

```
//New writeToParcel according new constructor
@Override
public int describeContents() {
    return 0;
}

/**
 * Method responsible for writing the data to the parcel
 * @param parcel
 * @param i
 */
@Override
public void writeToParcel(Parcel parcel, int i) {
    parcel.writeInt(idPOI);
    parcel.writeString(title);
    location.writeToParcel(parcel, i);
    parcel.writeString(address);
    parcel.writeString(lastModify);
    parcel.writeString(description);
    parcel.writeInt(idCategory);
    parcel.writeInt(idUser);
}

public static final Creator<PoiVO> CREATOR = new Creator<PoiVO>() {
    @Override
    public PoiVO createFromParcel(Parcel parcel) {
        return new PoiVO(parcel);
    }

    @Override
    public PoiVO[] newArray(int i) {
        return new PoiVO[i];
    }
};
```

Ilustración 29 - Escritura del parceable

A parte, esta clase implementa la clase Comparable necesaria para comparar dos puntos de interés que se utilizara en otro método que se explicara cuando se llegue a su clase principal. En la Ilustración se muestra como se hace la comparación.

```
/**
 * Returns 0 if equal, -1 if our point is closer to the user and 1 if our point is far
  * @param anotherPoi
 * @return
 */
@Override
public int compareTo(PoiVO anotherPoi) {
    if (this.distance < anotherPoi.getDistance())
        return -1;
    else if (this.distance > anotherPoi.getDistance())
        return 1;
    else {
        return 0;
    }
}
```

Ilustración 30 - Comparación de puntos de interés

8.1.3.3 *TaskVO*: Objeto encargado de almacenar la información de una tarea. Algunos de los parámetros a destacar son el id del servidor que lo obtendrá cuando la tarea sea subida al servidor para tener la relación entre las dos tablas y su prioridad que en función de ella es como se muestra en el mapa. En la Ilustración siguiente se mostrara el resto de parámetros.

Como es el caso anterior también contiene dos constructores el genérico para crear el objeto y el necesario por implementar la clase Parceable. En la Ilustración se pueden visualizar.

```
/**
 * Public enum
 */
public enum Prority {
    HIGH("High"), MEDIUM("Half"), LOW("Low");

    private String priority;

    Prority(String priority) { this.priority = priority; }

    @Override
    public String toString() { return priority; }
}
/** ID of task. */
private int idTask;

/** ID of task. */
private int idTaskServer;

/** Title of task. */
private String title;

/** Description of task. */
private String description;

/** Address of task. */
private String address;

/** LastModify of task. */
private String lastModify;

/** Longitude and latitude of Task. */
private Location location;

/** Hour of task */
private String time;

/** Day of task */
private String day;

/**Priority task*/
private String priority;

/**idUser of task*/
private int idUser;

/**idCategory of task*/
private int idCategory;
```

Ilustración 31 - Atributos tarea

```
     * @param idTaskServer
     * @param title
     * @param location
     * @param address
     * @param lastModify
     * @param description
     * @param time
     * @param day
     * @param priority
     * @param idCategory
     * @param idUser
     * @param imageUri
     */
    public TaskVO(int idTask, int idTaskServer, String title, Location location, String address,
                  String lastModify, String description, String time, String day,
                  String priority, int idCategory, int idUser, Uri imageUri) {
        super();
        this.idTask = idTask;
        this.idTaskServer = idTaskServer;
        this.title = title;
        this.location = location;
        this.address = address;
        this.lastModify = lastModify;
        this.description = description;
        this.time = time;
        this.day = day;
        this.priority = priority;
        this.idCategory = idCategory;
        this.idUser = idUser;
    }
    /**
     * Constructor need to parse the data
     * @param parcel
     */
    private TaskVO(Parcel parcel) {
        this.idTask = parcel.readInt();
        this.idTaskServer = parcel.readInt();
        this.title = parcel.readString();
        this.location = Location.CREATOR.createFromParcel(parcel);
        this.address = parcel.readString();
        this.lastModify = parcel.readString();
        this.description = parcel.readString();
        this.time = parcel.readString();
        this.day = parcel.readString();
        this.priority = parcel.readString();
        this.idCategory  = parcel.readInt();
        this.idUser  = parcel.readInt();
```

Ilustración 32 Constructores tareas

Además del constructor especial implementar la clase Parceable es necesario sobrecargar los métodos que se visualizan en la siguiente Ilustración.

```
//New writeToParcel according new constructor
@Override
public int describeContents() { return 0; }

/**
 * Method responsible for writing the data to the parcel
 * @param parcel
 * @param i
 */
@Override
public void writeToParcel(Parcel parcel, int i) {
    parcel.writeInt(idTask);
    parcel.writeInt(idTaskServer);
    parcel.writeString(title);
    location.writeToParcel(parcel, i);
    parcel.writeString(address);
    parcel.writeString(lastModify);
    parcel.writeString(description);
    parcel.writeString(time.toString());
    parcel.writeString(day.toString());
    parcel.writeString(priority);
    parcel.writeInt(idCategory);
    parcel.writeInt(idUser);
}

public static final Creator<TaskVO> CREATOR = new Creator<TaskVO>() {
    @Override
    public TaskVO createFromParcel(Parcel parcel) { return new TaskVO(parcel); }

    @Override
    public TaskVO[] newArray(int i) { return new TaskVO[i]; }
};
```

Ilustración 33 - Paseable de tareas

8.1.3.4 *UserVO*: Objeto donde se almacenan todos los datos del usuario. De los parámetros más destacables son el email necesario para identificar al usuario y su apiKey que es el parámetro de seguridad para poder acceder a su cuenta sin que nadie pueda averiguar sus datos. El resto de parámetros se visualizan en la siguiente Ilustración.

```
/**
 * Attributes
 */

/** idUser of User. */
private int idUser;

/** email of User. */
private String email;

/** name of User. */
private String name;

/** apiKey of User */
private String apiKey;

/** lastModify of User */
private String lastModify;

/** picture of User. */
private String picture;

/** Longitude of User. */
private double longitude;

/** Latitude of User. */
private double latitude;
```

Ilustración 34 -Atributos usuario

Además, en la Ilustración siguiente se muestra su constructor encargado de instanciar todos sus parámetros con los datos que le proporcionan.

```
/**
 * Constructor that instance all data
 * @param idUser
 * @param email
 * @param name
 * @param lastModify
 * @param apiKey
 * @param picture
 * @param longitude
 * @param latitude
 */
public UserVO(int idUser, String email, String name, String apiKey, String lastModify,
        String picture, double longitude,
        double latitude) {
    super();
    this.idUser = idUser;
    this.email = email;
    this.name = name;
    this.lastModify = lastModify;
    this.apiKey = apiKey;
    this.picture = picture;
    this.longitude = longitude;
    this.latitude = latitude;
}
```

Ilustración 35 - Constructor usuario

8.1.4 Maps: Paquete desarrollado por Jessica Carpintero Macián donde se encuentra la clase encargada de gestionar el mapa.

8.1.4.1 *MapManager*: Como su nombre indica es la clase que gestiona el mapa, es la que se ocupa de mostrar las tareas con sus respectivos puntos de interés en función de los parámetros que ha elegido el usuario en ajustes. A parte, también es la que recoge la posición del usuario. A continuación se mostraran las respectivas Ilustraciones de los métodos que se encargan de gestionar las funcionalidades mencionadas antes.

Primero de todo se visualizara el método de obtener la posición actual del usuario usado en MapFragment para poder pintarle su localización en el mapa. El método se encarga de seleccionar el mejor proveedor que haya disponible y obtener la posición del usuario, en caso de no encontrarla se ha introducido una localización fija en este caso se ha elegido Valencia capital por ser una aplicación para esta ciudad.

```java
/**
 * Through the best location provider that is available get the user's position
 * @return positionUser
 */
public Location obtainMyPosition(){

    SharedPreferences sharedPreferences = mContext.getSharedPreferences(PREFERENCES_FILE, Context.MODE_PRIVATE);
    int status = GooglePlayServicesUtil.isGooglePlayServicesAvailable(mContext);

    // Showing status
    if(status!= ConnectionResult.SUCCESS){ // Google Play Services are not available

    }else { // Google Play Services are available
        // Enabling MyLocation Layer of Google Map
        map.setMyLocationEnabled(true);

        // Getting LocationManager object from System Service LOCATION_SERVICE
        LocationManager locationManager = (LocationManager) mContext.getSystemService(mContext.LOCATION_SERVICE);


            // Creating a criteria object to retrieve provider
            Criteria criteria = new Criteria();

            // Getting the name of the best provider
            String provider = locationManager.getBestProvider(criteria, true);

            // Getting Current Location
            positionUser = locationManager.getLastKnownLocation(provider);


            if (positionUser == null) {
                positionUser = new Location("");
                positionUser.setLatitude(Double.longBitsToDouble(sharedPreferences.getLong(KEY_LATITUDE_USER, Double.doubleToLongBits(39.470863))));
                positionUser.setLongitude(Double.longBitsToDouble(sharedPreferences.getLong(KEY_LONGITUDE_USER, Double.doubleToLongBits(-0.376412))));
            }

            locationManager.requestLocationUpdates(provider, 20000, 0, this);

        return positionUser;
    }
    return positionUser;
}
```

Ilustración 36 - Obtener posición del usuario

Después en las siguientes Ilustraciones se puede ver los métodos utilizados para visualizar las tareas y sus puntos de interés. Que serán pintados en función de la prioridad, el número de tareas y puntos de interés que usuario hay seleccionado en ajustes. El primer método que se puede ver es el encargado de pintar todas las tareas en el mapa, en función de si hay una tarea seleccionada. En el caso de que haya una tarea seleccionada, esta se la pintara a través de otro método que se explicara más adelante. En el otro caso todas las tareas y sus puntos de interés serán pintadas en base a su prioridad.

```
/**
 * Method that shows all tasks on the map, even call different method depending on whether you have a selected task
 * @param taskVOList
 * @param idTask
 * @return
 */
public int addAllTask(List<TaskVO> taskVOList, int idTask){
    int numTaskAdd = 0;

    if (idTask != 0 && showTaskSelected && taskVOList != null){
        TaskVO task = dbManager.getTask(idTask);
        for (int i = 0; i < taskVOList.size(); i++) {
            if (task.getTitle().equals(taskVOList.get(i).getTitle())){
                taskVOList.remove(i);
            }
        }
        numTaskAdd++;
        addTask(task);
    }
    if (numTask != 0 && taskVOList != null) {
        for (TaskVO task1 : taskVOList) {
            if (numTaskAdd < numTask) {
                addMarkerTask(task1);
                obtainPoisToCategory(task1);
                numTaskAdd++;
            }
        }
    } else {
        if (taskVOList != null) {
            for (TaskVO task1 : taskVOList) {
                addMarkerTask(task1);
                obtainPoisToCategory(task1);
            }
        }
    }

    return idTask;
}
```

Ilustración 37 - Añadir todas las tareas

En la Ilustración se puede visualizar los métodos mencionados en el párrafo anterior que pintaran las tareas en función de su prioridad. En el caso genérico se pintaran de rojo para las de tareas de alta prioridad, amarillo para las tareas medias y verde para las de baja esto se puede observar en el segundo método de la Ilustración. A parte tenemos el método que se encargara de pintar la tarea y los puntos de interés de la tarea que haya seleccionado el usuario en el gestor de tareas que será pintada de color morado a la localización principal de la tarea y de color turquesa a sus puntos de interés, además pintara también el resto de tareas con el color que le corresponda a su prioridad llamando al segundo método.

```
    */
    public void addTask(TaskVO task){
        if (task.getLocation().getLatitude() != 0.000000 && task.getLocation().getLongitude() != 0.000000) {
                Log.v("VERBOSE", "Entro en la prioridad alta ");
                map.addMarker(new MarkerOptions()
                        .position(new LatLng(task.getLocation().getLatitude(), task.getLocation().getLongitude())).
                                title(task.getTitle()).icon(BitmapDescriptorFactory.fromResource(R.drawable.squat_marker_purple_t)));
        }
        List<PoiVO> poiList = new ArrayList<>();
        int numPointsAdd = 0;
        poiList = getPoiListSortByDistance(dbManager.getPoisByCategory(task.getIdCategory()));
        if(!poiList.isEmpty()) {
            for (PoiVO poi : poiList) {
                Log.v("VERBOSE", "dentro de addTask El POI es : " + poi.getIdPOI());
                if (numPointsAdd < numPointShow) {
                    Log.v("VERBOSE", "dentro de addTask El POI es  dentro del if: " + poi.getIdPOI());
                    if (poi.getLocation().getLatitude() != 0.000000 && poi.getLocation().getLongitude() != 0.000000) {
                        map.addMarker(new MarkerOptions()
                                .position(new LatLng(poi.getLocation().getLatitude(), poi.getLocation().getLongitude())).
                                        title(task.getTitle()).icon(BitmapDescriptorFactory.fromResource(R.drawable.squat_marker_cyan_t)));
                        numPointsAdd++;
                    }
                }
            }
        }
    }

    /**
     * Method which adds the corresponding markers for each task, painted according to task priority
     * @param task
     */
    public void addMarkerTask(TaskVO task){
        if (task.getLocation().getLatitude() != 0.000000 && task.getLocation().getLongitude() != 0.000000) {
            if ("High".equals(task.getPriority())) {
                Log.v("VERBOSE", "Entro en la prioridad alta ");
                map.addMarker(new MarkerOptions()
                        .position(new LatLng(task.getLocation().getLatitude(), task.getLocation().getLongitude())).
                                title(task.getTitle()).icon(BitmapDescriptorFactory.fromResource(R.drawable.squat_marker_red_f)));
            } else if ("Half".equals(task.getPriority())) {
                Log.v("VERBOSE", "Entro en la prioridad media ");
                map.addMarker(new MarkerOptions()
                        .position(new LatLng(task.getLocation().getLatitude(), task.getLocation().getLongitude())).
                                title(task.getTitle()).icon(BitmapDescriptorFactory.fromResource(R.drawable.squat_marker_yellow_t)));
            } else {
                Log.v("VERBOSE", "Entro en la prioridad baja ");
                map.addMarker(new MarkerOptions()
                        .position(new LatLng(task.getLocation().getLatitude(), task.getLocation().getLongitude())).
                                title(task.getTitle()).icon(BitmapDescriptorFactory.fromResource(R.drawable.squat_marker_green_f)));
            }
        }
    }
```

Ilustración 38 - Añadir todas las tareas al mapa

El siguiente método que se va a observar es el encargado de obtener los puntos de interés de la tarea en función a la categoría que pertenece y pintarlos en función de la prioridad de la tarea. Este método solo es utilizado cuando pintamos los puntos de interés de las tareas que no hayan sido seleccionadas.

```
/**
 * Method which adds the corresponding markers for each POI, painted according to task priority
 * @param task
 */
public void obtainPoisToCategory(TaskVO task){
    List<PoiVO> poiList = new ArrayList<>();
    int numPointsAdd = 0;
    poiList = getPoiListSortByDistance(dbManager.getPoisByCategory(task.getIdCategory()));
    for (PoiVO poi: poiList) {
        if (numPointsAdd < numPointShow) {
            if (poi.getLocation().getLatitude() != 0.000000 && poi.getLocation().getLongitude() != 0.000000) {
                if ("High".equals(task.getPriority())) {
                    map.addMarker(new MarkerOptions()
                            .position(new LatLng(poi.getLocation().getLatitude(), poi.getLocation().getLongitude())).
                                    title(task.getTitle()).icon(BitmapDescriptorFactory.fromResource(R.drawable.squat_marker_red_f)));
                    numPointsAdd++;
                } else if ("Half".equals(task.getPriority())) {
                    map.addMarker(new MarkerOptions()
                            .position(new LatLng(poi.getLocation().getLatitude(), poi.getLocation().getLongitude())).
                                    title(task.getTitle()).icon(BitmapDescriptorFactory.fromResource(R.drawable.squat_marker_yellow_t)));
                    numPointsAdd++;
                } else {
                    map.addMarker(new MarkerOptions()
                            .position(new LatLng(poi.getLocation().getLatitude(), poi.getLocation().getLongitude())).
                                    title(task.getTitle()).icon(BitmapDescriptorFactory.fromResource(R.drawable.squat_marker_green_f)));
                    numPointsAdd++;
                }
            }
        }
    }
}
```

Ilustración 39 - Obtener puntos de interés

Continuando, se encuentra el método que nos devuelve la lista de puntos de interés ordenada en función de la distancia a la que encuentren del usuario, como se ha visualizado en las Ilustraciones anteriores este método se utiliza para que al usuario se le muestre los puntos más cercanos a él.

```
/**
 * Method that calculates the distance of the POI and sorts
 * @param poiList
 * @return
 */
public List<PoiVO> getPoiListSortByDistance(List<PoiVO> poiList){
    LatLng locationPoi = null, locationUser = null;

    if (poiList.size() != 0) {
        Location locationUserAux = obtainMyPosition();
        locationUser = new LatLng(locationUserAux.getLatitude(), locationUserAux.getLongitude());

        //Calculate distance between the user and a poi
        for (PoiVO poi : poiList) {
            locationPoi = new LatLng(poi.getLocation().getLatitude(), poi.getLocation().getLongitude());
            poi.setDistance(distanceBetweenTwoPoints(locationUser, locationPoi));
        }

        Comparator<PoiVO> compare = (poi1, poi2) -> {
                return poi1.compareTo(poi2);
        };

        //Sort POIs by distance
        Collections.sort(poiList, compare);
    }
    return poiList;
}
```

Ilustración 40 - Obtener puntos de interés por distancia

Además, se tiene el método encargado de indicar si es la mejor localización en función del momento que se recogió con la mejor precisión posible. Este método es usado en el algoritmo de actualizar el GPS en función de los ajustes indicados por el usuario.

```
/**
 * Decide if new location is better than older by following some basic criteria.
 * This algorithm can be as simple or complicated as your needs dictate it.
 * Try experimenting and get your best location strategy algorithm.
 *
 * @param oldLocation Old location used for comparison.
 * @param newLocation Newly acquired location compared to old one.
 * @return If new location is more accurate and suits your criteria more than the old one.
 */
public boolean isBetterLocation(Location oldLocation, Location newLocation, double modeTime) {
    // If there is no old location, of course the new location is better.
    if(oldLocation == null) {
        return true;
    }

    // Check if new location is newer in time.
    boolean isNewer = newLocation.getTime() > oldLocation.getTime();

    // Check if new location more accurate. Accuracy is radius in meters, so less is better.
    boolean isMoreAccurate = newLocation.getAccuracy() < oldLocation.getAccuracy();
    if(isMoreAccurate && isNewer) {
        // More accurate and newer is always better.
        return true;
    } else if(isMoreAccurate && !isNewer) {
        // More accurate but not newer can lead to bad fix because of user movement.
        // Let us set a threshold for the maximum tolerance of time difference.
        long timeDifference = newLocation.getTime() - oldLocation.getTime();

        // If time difference is not greater then allowed threshold we accept it.
        if(timeDifference > -modeTime) {
            return true;
        }
    }

    return false;
}
```

Ilustración 41 - Mejor localización del GPS

Y por último, se puede visualizar en la Ilustración el método encargado de gestionar el modo Auto del algoritmo de actualización del GPS.

```
/**
 * Method responsible for updating the GPS position of the user
 * @param myPositionLatLng
 * @param oldPositionLatLng
 * @param oldOldPositionLatLng
 * @return
 */
public boolean modeGPSAuto(LatLng myPositionLatLng, LatLng oldPositionLatLng, LatLng oldOldPositionLatLng) {
    double distanceNow = distanceBetweenTwoPoints(myPositionLatLng, oldPositionLatLng);
    double distanceOld = distanceBetweenTwoPoints(oldPositionLatLng, oldOldPositionLatLng);
    double differenceDistances = distanceNow - distanceOld;
    double modeTimeAuto = Double.longBitsToDouble(sharedPreferences.getLong(MODE_TIME_AUTO,30));
    if(distanceNow > (distanceOld * BIG_DISTANCE_PERCENTAGE)){
        //If the differences are similar, set modeTimeAuto decreased by 30%
        editor.putLong(MODE_TIME_AUTO,Double.doubleToLongBits(modeTimeAuto * LOW_TIME_PERCENTAGE));
        return  true;
    }else if( (distanceNow * LOW_DISTANCE_PERCENTAGE) < distanceOld ){
        //If the user move much more than later,  set modeTimeAuto increased 30%
        editor.putLong(MODE_TIME_AUTO,Double.doubleToLongBits(modeTimeAuto * BIG_TIME_PERCENTAGE));
        return  true;
    } else if (differenceDistances > 20 ){
        //If the user move more, set modeTimeAuto decreased by 10 seconds
        editor.putLong(MODE_TIME_AUTO,Double.doubleToLongBits(modeTimeAuto)-10);
        return  true;
    } else if(differenceDistances > -20){
        //If the user move less, set modeTimeAuto increased by 10 seconds
        editor.putLong(MODE_TIME_AUTO,Double.doubleToLongBits(modeTimeAuto)+10);
        return  true;
    }  else if(differenceDistances <= 20 && differenceDistances >= -20){
        //If the differences are similar, set modeTimeAuto increased by 5 seconds
        editor.putLong(MODE_TIME_AUTO,Double.doubleToLongBits(modeTimeAuto)+5);
        return  true;
    }
    return false;
}
```

Ilustración 42 - Modo GPS automático

8.1.5 Notifications: Paquete donde se gestionan las notificaciones de la aplicación que recuerdan al usuario cuando debe o tiene cerca una tarea.

8.1.5.1 *NotificationMangement*: Clase donde se encuentra todos los métodos que permiten lanzar la notificación al usuario. A continuación en las Ilustraciones se visualizaran todos los métodos encargados de realizar esta funcionalidad.

El primer método que se puede visualizar es el encargado de iniciar el servicio primero de todo se filtran las tareas en función de la prioridad que el usuario a elegido en ajustes. Después, en función de esa elección se lanzara los métodos que controlan el lanzamiento de las notificaciones.

```
/**
 *
  * @param intent
 * @param flags
 * @param startId
 * @return
 */
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    //Start background task
    if(!taskList.isEmpty())
         taskList = filterByPriority();

    if (ENABLE_BY_PRIORITY_TIMEOUT.equals(enableNotifications)) {
        notificationsByTimeout();
    } else if (ENABLE_BY_PRIORITY_NEAR.equals(enableNotifications)) {
        notificationsByNear();
    } else if (ENABLE_BOTH_NOTIFICATIONS.equals(enableNotifications)){
        notificationsByTimeout();
        if (!taskList.isEmpty())
            notificationsByNear();
    }

    return Service.START_STICKY;
}
```

Ilustración 43 - Carga de notificaciones

En la siguiente Ilustración se muestra el lanzamiento de la notificación de cercanía, para la de tiempo hay un método casi idéntico que se encargara de lanzar la notificación de tiempo. En ellos es donde se modifica y determina la vibración, sonido, etc. En el caso de la que vamos a seleccionar es necesario guardarnos la tarea que se notifique para que después MapFragment pueda pintarla como la tarea seleccionada, por tanto cuando el usuario seleccione la notificación esta le abrirá el mapa con la tarea indicada.

```
    * Prepares and launch the notification by near
    */
   private void notificationByNear(double distanceTask) {

       NotificationManager mManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

       //Prepares activity that will open when the user clicks the notification
       Intent intentNot = new Intent(this, MainActivity.class);
       if (task.getLocation().getLongitude() == 0 && task.getLocation().getLatitude() == 0){
           intentNot.putExtra(KEY_ID_TASK, task.getIdTask());
           List<PoiVO> poisList = dbManager.getPoisByCategory(task.getIdCategory());
           if (!poisList.isEmpty()) {
               intentNot.putExtra(KEY_LATITUDE, poisList.get(0).getLocation().getLatitude());
               intentNot.putExtra(KEY_LONGITUDE, poisList.get(0).getLocation().getLongitude());
           }
       } else {
           intentNot.putExtra(KEY_ID_TASK, task.getIdTask());
           intentNot.putExtra(KEY_LATITUDE, task.getLocation().getLatitude());
           intentNot.putExtra(KEY_LONGITUDE, task.getLocation().getLongitude());
       }
       //Prepares the notification
       Notification notification = new Notification(R.drawable.squat_marker_cyan_t, task.getTitle(), System.currentTimeMillis());
       notification.setLatestEventInfo(this, "Where is my task", task.getTitle() + " " + String.format("%.0f", distanceTask) + "m",
               PendingIntent.getActivity(this.getBaseContext(), 0, intentNot, PendingIntent.FLAG_CANCEL_CURRENT));

       //Add sound
       notification.defaults |= Notification.DEFAULT_SOUND;
       //Add vibration
       notification.defaults |= Notification.DEFAULT_VIBRATE;

       //Adds LED light
       notification.defaults |= Notification.DEFAULT_LIGHTS;

       //The notification will stop when the user clicks on it
       notification.flags = Notification.FLAG_AUTO_CANCEL;

       //Try to set the color and flashing LED
       try {
           notification.ledARGB = Color.BLUE;
           notification.ledOnMS = 300;
           notification.ledOffMS = 1000;
           notification.flags |= Notification.FLAG_SHOW_LIGHTS;
       } catch (Exception ex) {
           //Nothing
       }

       //Launch notification
       mManager.notify(APP_ID_NOTIFICATION_NEAR, notification);
   }
```

Ilustración 44 - Lanzamiento notificación de cercanía

A continuación podemos visualizar el método que se encarga de crear una tarea programada por cada tarea en función del tiempo que el usuario haya elegido en ajustes de que se le avise. Más adelante se explicara la clase que lanzara la notificación cuando la hora y el día de la tarea programada se haya finalizado.

```java
/**
 * Method of charge to send the notification time according
 * to the value selected by the user settings and the date containing the task.
 * Notification jump on the hour and day indicated by the difference between the previous data
 */
public void notificationsByTimeout() {
    if(!taskList.isEmpty()) {
        for (TaskVO task : taskList) {
            this.task = task;
            Date dateTask = convertStringToDate(task.getDay(), task.getTime());
            dateTask = saveDate(dateTask);
            Timer timer = new Timer(true);
            // running timer task as daemon thread
            if (!task.isTimerTaskActive()) {
                TimerTask timerTask = new TimerTaskExample(task);
                timer.schedule(timerTask, dateTask);
                task.setTimerTaskActive(true);
                task.setNotificationFinished(false);
                HashMap<String, Boolean> values = new HashMap<String, Boolean>(3);
                values.put(TIMER_TASK_ACTIVE, true);
                values.put(NOTIFICATION_FINISHED, false);
                values.put(TIMER_TASK_CANCEL, false);
                dbManager.updateTaskValuesNotifications(values, task);
            } else if ((!task.isTimerTaskActive() && task.isTimerTaskCancel()) ||
                    (task.isTimerTaskActive() && (!dbManager.existsTask(task.getIdTask()) || task.isNotificationFinished()))) {
                timer.cancel();
                task.setTimerTaskActive(false);
                task.setNotificationFinished(false);
                task.setTimerTaskCancel(false);
                HashMap<String, Boolean> values = new HashMap<String, Boolean>(3);
                values.put(TIMER_TASK_ACTIVE, false);
                values.put(NOTIFICATION_FINISHED, false);
                values.put(TIMER_TASK_CANCEL, false);
                dbManager.updateTaskValuesNotifications(values, task);
            }
        }
    }

}
```

Ilustración 45 – Notificación de tiempo

El siguiente es el método encargado de comprobar que tareas son cercanas al usuario en función de los metros que haya seleccionado en ajustes, en el caso de tener tareas cercanas se llamara al método de lanzar la notificación explicado anteriormente.

```java
/**
 * I charge to send method notification when we close proximity to the user tasks
 */
public void notificationsByNear() {
    //Get values of SharedPreferences
    latitude = Double.longBitsToDouble(prefs.getLong(KEY_LATITUDE_USER, 0));
    longitude = Double.longBitsToDouble(prefs.getLong(KEY_LONGITUDE_USER, 0));
    LatLng locationUser = new LatLng(latitude, longitude);
    for (TaskVO task : taskList) {
        this.task =  task;
        LatLng locationTask = new LatLng(task.getLocation().getLatitude(), task.getLocation().getLongitude());
        if (task.getLocation().getLatitude() != 0 && task.getLocation().getLongitude() != 0) {
            //Distance in meters
            double distanceTask = MapManager.distanceBetweenTwoPoints(locationUser, locationTask);
            if (distanceTask < notificationsByNear) {
                notificationByNear(distanceTask);
            }
        }
    }
}
```

Ilustración 46 - Notificación por cercanía

Y la última Ilustración se muestra la clase encargada de implementar la tarea programada y es la responsable de llamar al método de lanzar la notificación por tiempo.

```
/**
 * Implements the scheduled task and responsible for notifying the tasks by time
 */
public class TimerTaskExample extends TimerTask {

    private TaskVO taskVO;

    public TimerTaskExample(TaskVO task) {
        taskVO = task;
    }

    @Override
    public void run() {
        notificationByTimeout(taskVO);
        HashMap<String, Boolean> values = new HashMap<String, Boolean>(3);
        taskVO.setNotificationFinished(true);
        values.put(TIMER_TASK_ACTIVE, taskVO.isTimerTaskActive());
        values.put(NOTIFICATION_FINISHED, true);
        values.put(TIMER_TASK_CANCEL, taskVO.isTimerTaskCancel());
        dbManager.updateTaskValuesNotifications(values, taskVO);
    }
}
```

Ilustración 47 - Tarea programada

8.1.6 Persistence: Paquete donde se encuentran la clase realizada por mi compañero y explicada en su memoria que está encargada de crear los datos que devuelve el servidor en un json y el gestor de base de datos que desarrollado por mí y donde se gestiona la base de datos interna de la aplicación.

8.1.6.1 *DBManager*: Clase encargada de gestionar toda la persistencia, es quien crea, borra, modifica, obtiene,… todos los datos de la base de datos local del usuario. A continuación se verán unos métodos de ejemplo de la clase.

La primera Ilustración nos muestra la creación de las tablas cuando se instancia la clase.

```
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_CATEGORY);

        db.execSQL("CREATE TABLE IF NOT EXISTS " + TABLE_CATEGORY + " (idCategory INTEGER  PRIMARY KEY, " +
                "nameCategory VARCHAR(100) NOT NULL, " +
                "idParentCategory INTEGER NOT NULL, " +
                "imageCategory VARCHAR(100), " +
                "last_modify TIMESTAMP DEFAULT CURRENT_TIMESTAMP);");

        db.execSQL("CREATE TABLE IF NOT EXISTS " + TABLE_POI + " (idPOI INTEGER  PRIMARY KEY, " +
                "longitude DOUBLE PRECISION DEFAULT 0.000000, " +
                "latitude DOUBLE PRECISION DEFAULT 0.000000, " +
                "title VARCHAR(250) NOT NULL, " +
                "address VARCHAR(200), " +
                "last_modify TIMESTAMP DEFAULT CURRENT_TIMESTAMP, " +
                "description VARCHAR(400), " +
                "idCategory INTEGER NOT NULL, " +
                "idUser INTEGER NOT NULL, " +
                "FOREIGN KEY (idCategory) REFERENCES category (idCategory), " +
                "FOREIGN KEY (idUser) REFERENCES user (idUser))");

        db.execSQL("CREATE TABLE IF NOT EXISTS " + TABLE_USERS + " (idUser INTEGER PRIMARY KEY," +
                " name varchar(250)," +
                " email varchar(255) UNIQUE NOT NULL, " +
                "api_key varchar(32) NOT NULL, " +
                "last_modify timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP, " +
                "longitude DOUBLE PRECISION DEFAULT 0.000000, " +
                "latitude DOUBLE PRECISION DEFAULT 0.000000, " +
                "picture varchar(200));");

        db.execSQL("CREATE TABLE IF NOT EXISTS " + TABLE_TASK + " (idTask INTEGER PRIMARY KEY, " +
                "idTaskServer INTEGER DEFAULT 0, " +
                "title VARCHAR(250) NOT NULL, " +
                "longitude DOUBLE PRECISION DEFAULT 0.000000, " +
                "latitude DOUBLE PRECISION DEFAULT 0.000000, " +
                "address VARCHAR(200), " +
                "last_modify TIMESTAMP DEFAULT CURRENT_TIMESTAMP, " +
                "description VARCHAR(300), " +
                "time TIMESTAMP, " +
                "day DATE, " +
                "priority VARCHAR(50) NOT NULL DEFAULT 'Low', " +
                "idCategory INTEGER NOT NULL, " +
                "idUser INTEGER NOT NULL, " +
                "timerTaskActive INTEGER DEFAULT 0, " +
                "timerTaskCancel INTEGER DEFAULT 0, " +
                "notificationFinished INTEGER DEFAULT 0, " +
                "FOREIGN KEY (idUser) REFERENCES user (idUser), FOREIGN KEY (idCategory) REFERENCES category (idCategory))");
    } catch (SQLiteException sqlError) {
```

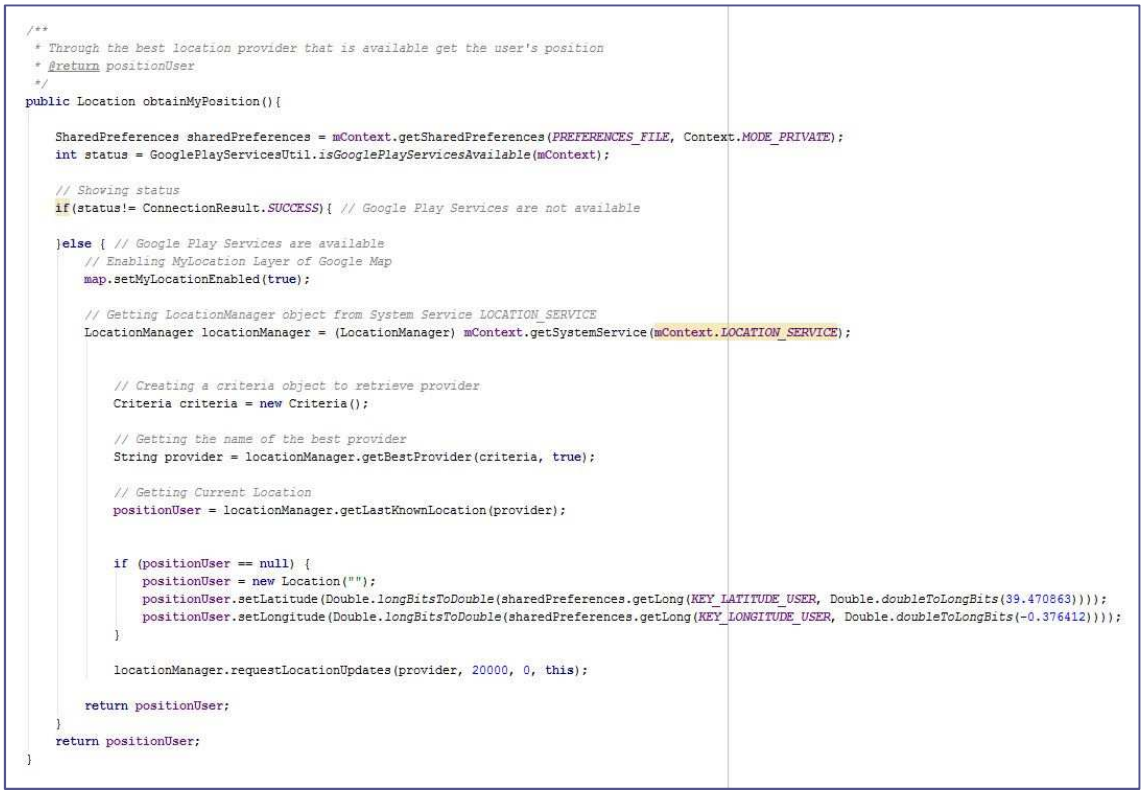
Ilustración 48 - Tablas de la base de datos

En la Ilustración siguiente se muestra un ejemplo de los métodos de añadir y obtener en este caso el ejemplo está basado en el objeto categoría. De todas formas por cada objeto existe el mismo método.

```
// Add new Category
public void addCategory(CategoryVO category) {
    SQLiteDatabase db = this.getWritableDatabase();
    try{
        ContentValues values = new ContentValues();
        values.put("idCategory", category.getIdCategory());
        values.put("nameCategory", category.getName());
        values.put("idParentCategory", category.getIdParentCategory());
        values.put("imageCategory", category.getImageCategory());
        values.put("last_modify", category.getLast_modify());
        // Inserting Row
        db.insert(TABLE_CATEGORY, null, values);
    } catch (SQLiteException sqlError){
        Toast toast = Toast.makeText(this.myContext, "ERROR. Not been able to be inserted in the database", Toast.LENGTH_SHORT);
        toast.show();
    } finally{
        // db.close(); // Closing database connection
    }

}
// Getting Categories
public List<CategoryVO> getAllCategory() {
    List<CategoryVO> categoryList = new ArrayList<>();
    Cursor cursor = null;
    SQLiteDatabase db = this.getWritableDatabase();
    try{
        cursor = db.rawQuery("select idCategory, nameCategory, idParentCategory, imageCategory, last_modify from " + TABLE_CATEGORY + ";", null);
        // looping through all rows and adding to list
        if (cursor.moveToFirst()) {
            do {
                CategoryVO category = new CategoryVO();
                category.setIdCategory(Integer.parseInt(cursor.getString(0)));
                category.setName(cursor.getString(1));
                category.setIdParentCategory(cursor.getInt(2));
                category.setImageCategory(cursor.getString(3));
                category.setLast_modify(cursor.getString(4));
                categoryList.add(category);
            } while (cursor.moveToNext());
        }
    } catch (SQLiteException sqlError){
        Toast toast = Toast.makeText(this.myContext, "ERROR. Could not execute query", Toast.LENGTH_SHORT);
        toast.show();
    } finally{
        // db.close(); // Closing database connection
    }

    // return points list
    return categoryList;
}
```

Ilustración 49 - Añadir y obtener categoría

Y por último, se muestra el ejemplo de modificar y borrar con el mismo ejemplo de antes.



```
/**
 * Through the best location provider that is available get the user's position
 * @return positionUser
 */
public Location obtainMyPosition(){

    SharedPreferences sharedPreferences = mContext.getSharedPreferences(PREFERENCES_FILE, Context.MODE_PRIVATE);
    int status = GooglePlayServicesUtil.isGooglePlayServicesAvailable(mContext);

    // Showing status
    if(status!= ConnectionResult.SUCCESS){ // Google Play Services are not available

    }else { // Google Play Services are available
        // Enabling MyLocation Layer of Google Map
        map.setMyLocationEnabled(true);

        // Getting LocationManager object from System Service LOCATION_SERVICE
        LocationManager locationManager = (LocationManager) mContext.getSystemService(mContext.LOCATION_SERVICE);


            // Creating a criteria object to retrieve provider
            Criteria criteria = new Criteria();

            // Getting the name of the best provider
            String provider = locationManager.getBestProvider(criteria, true);

            // Getting Current Location
            positionUser = locationManager.getLastKnownLocation(provider);


            if (positionUser == null) {
                positionUser = new Location("");
                positionUser.setLatitude(Double.longBitsToDouble(sharedPreferences.getLong(KEY_LATITUDE_USER, Double.doubleToLongBits(39.470863))));
                positionUser.setLongitude(Double.longBitsToDouble(sharedPreferences.getLong(KEY_LONGITUDE_USER, Double.doubleToLongBits(-0.376412))));
            }

            locationManager.requestLocationUpdates(provider, 20000, 0, this);

        return positionUser;
    }
    return positionUser;
}
```

Ilustración 50 - Modificar y borrar categoría

A parte de estos métodos la clase contiene muchos más, dependiendo de lo que se necesita recoger de la base de datos de las diferentes clases de la aplicación.

8.1.7 Utils: Paquete que contiene la clase que proporciona comprobaciones de conexión.

8.1.7.1 *ConnectionDetecto*: es una clase que detecta si la conexión a internet o el GPS están activos. Son métodos utilizados en otras clases de la aplicación. En la Ilustración se muestran estos dos métodos.

```
/**
 * Check for internet connection
 * @return
 */
public boolean isConnectingToInternet(){
    ConnectivityManager connectivity = (ConnectivityManager) mContext.getSystemService(Context.CONNECTIVITY_SERVICE);
    if (connectivity != null)
    {
        NetworkInfo[] info = connectivity.getAllNetworkInfo();
        if (info != null)
            for (int i = 0; i < info.length; i++)
                if (info[i].getState() == NetworkInfo.State.CONNECTED)
                {
                    return true;
                }

    }
    return false;
}

/**
 * Check if GPS is active
 * @return
 */
public boolean isConnectingToGps(){
        LocationManager locManager = (LocationManager) mContext.getSystemService(Context.LOCATION_SERVICE);
        if(!locManager.isProviderEnabled(LocationManager.GPS_PROVIDER)){
            return false;
        }
        else return true;

}
```

Ilustración 51 - Conexión a Internet

8.1.8 Activitys: La última parte que se puede ver en la Ilustración de la estructura son las actividades de la aplicación son las encargadas de mostrar la información de los objetos de la aplicación por pantalla. A continuación se detallara la funcionalidad de cada actividad realizada por mí, en el caso de registro y inicio de sesión se encontraran en la memoria de mi compañero:

8.1.8.1 *AddTaskActivity*: Es la actividad que muestra al usuario la pantalla de añadir o modificar una tarea. A continuación se mostraran los diferentes métodos que implementa esta clase:

En la primera Ilustración se puede observar el método que permite que se visualice una nueva pantallita con un despegable donde el usuario podrá elegir el punto de interés favorito para esa tarea, una vez lo seleccione su dirección y localización serán las mismas que punto de interés.

```
    * Displays a window where part user can select the desired point of interest
    */
    private void showDialogFromAdd() {
        LayoutInflater inflater = this.getLayoutInflater();

        List<PoiVO> poiList = dbManager.getAllPoi();


        AlertDialog.Builder alertDialog;
        alertDialog = new AlertDialog.Builder(this);
        alertDialog.setTitle(R.string.spinner_title_add_task);
        View view = inflater.inflate(R.layout.fragment_name_pois, null);
        alertDialog.setView(view);

        final Spinner spinnerPois = (Spinner) view.findViewById(R.id.fragment_name_pois_spinner);

        PoiSpinnerAdapter poiTitleAdapter = new PoiSpinnerAdapter(this, poiList);
        spinnerPois.setAdapter(poiTitleAdapter);

        spinnerPois.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {

            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {

            }
        });

        alertDialog.setPositiveButton(android.R.string.ok, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                PoiVO poiSelected = (PoiVO) spinnerPois.getSelectedItem();
                address.setText(poiSelected.getAddress());
                locationTaskNew = poiSelected.getLocation();
                addressTaskNew = poiSelected.getAddress();
            }
        });

        alertDialog.setNegativeButton(android.R.string.cancel, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                dialogInterface.dismiss();
            }
        });
        alertDialog.show();
    }
```

Ilustración 52 - Despegable punto de interés en añadir tarea

A parte de este método, está la versión para el caso de la pantalla de modificar que se muestra en la siguiente Ilustración.

```java
/**
 * Method that shows points of interest on the screen modified
 * @param task
 */
public void showDialogFromUpdate(TaskVO task) {
    LayoutInflater inflater = this.getLayoutInflater();

    List<PoiVO> poiList = dbManager.getAllPoi();

    AlertDialog.Builder alertDialog;
    alertDialog = new AlertDialog.Builder(this);
    alertDialog.setTitle(R.string.spinner_title_add_task);
    View view = inflater.inflate(R.layout.fragment_name_pois, null);
    alertDialog.setView(view);

    final Spinner spinnerPois = (Spinner) view.findViewById(R.id.fragment_name_pois_spinner);

    PoiSpinnerAdapter poiTitleAdapter = new PoiSpinnerAdapter(this, poiList);
    spinnerPois.setAdapter(poiTitleAdapter);

    spinnerPois.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {

        }

        @Override
        public void onNothingSelected(AdapterView<?> parent) {

        }
    });

    alertDialog.setPositiveButton(android.R.string.ok, (dialogInterface, i) -> {
            PoiVO poiSelected = (PoiVO) spinnerPois.getSelectedItem();
            address.setText(poiSelected.getAddress());
            taskUpdate.setAddress(poiSelected.getAddress());
            taskUpdate.setLocation(poiSelected.getLocation());
    });

    alertDialog.setNegativeButton(android.R.string.cancel, (dialogInterface, i) -> {
            dialogInterface.dismiss();
    });
    alertDialog.show();
}
```

Ilustración 53 - Despegable punto de interés en modificar tarea

Seguidamente en la siguiente Ilustración se muestra casi toda la implementación del método de añadir tarea, como se puede visualizar se recogen todos los datos que se haya introducido el usuario por pantalla y se añaden a la base de datos local como al servidor.

```
newTask.setIdTask(idTask);
newTask.setIdTaskServer(0);
newTask.setTitle(String.valueOf(title.getText()));
Location l = new Location("");
l.setLatitude(0);
l.setLongitude(0);
newTask.setLocation(l);
if ("".equals(addressTaskNew))
    newTask.setAddress(String.valueOf(address.getText()));
else
    newTask.setAddress(addressTaskNew);
newTask.setDescription(String.valueOf(description.getText()));
if(hour.getCurrentHour() < 10){
    hourMin = "0" + hour.getCurrentHour();
} else {
    hourMin = String.valueOf(hour.getCurrentHour());
}
if(hour.getCurrentMinute() < 10){
    minMin = "0" + hour.getCurrentMinute();
} else {
    minMin = String.valueOf(hour.getCurrentMinute());
}
newTask.setTime(hourMin + ":" + minMin);

if(day.getDayOfMonth() < 10){
    dayMin = "0" + day.getDayOfMonth();
} else {
    dayMin = String.valueOf(day.getDayOfMonth());
}
if(day.getMonth() < 10){
    monthMin = "0" + day.getMonth();
} else {
    monthMin = String.valueOf(day.getMonth());
}
newTask.setDay(day.getYear() + "-" + monthMin + "-" + dayMin);
newTask.setPriority(priorityValues);
newTask.setIdCategory(idCategory);
newTask.setIdUser(getIdUser());
dbManager.addTask(newTask);
if (locationTaskNew != null)
    newTask.setLocation(locationTaskNew);
else
    checkAddressLocation(newTask);
if (newTask != null){
    try{
        CreateTask createTask = new CreateTask();
        synchronized (createTask) {
            createTask.execute(newTask).wait(1000);
        }
    } catch (InterruptedException e) {
```

Ilustración 54 - Añadir tarea

A continuación, se puede ver el método que llama a recoger los datos de la tarea modificada y después hace los cambios en base de datos y en el servidor.

```
/**
 * Method that modifies a task based on the parameters entered by the user
 */
public void updateTask(){
    if (taskUpdate != null) {
        getUpdateTask(taskUpdate);
        dbManager.updateTask(taskUpdate);
        if (0.0 == taskUpdate.getLocation().getLongitude() && 0.0 == taskUpdate.getLocation().getLatitude()
                && taskUpdate.getAddress() != null && !"".equals(taskUpdate.getAddress())) {
            checkAddressLocation(taskUpdate);
        }
        new UpdateTask().execute(taskUpdate);
    }
}
```

Ilustración 55 - Modificar tarea

Después de este método se puede observar el método encargado de recoger los datos de la pantalla y modificar la tarea para que pueda ser actualizada en el método anterior.

```
/**
 * Method which gets the values of the task to change the display to show the data that already existed
 * @param taskUpdate
 */
private void getUpdateTask(TaskVO taskUpdate) {
    String hourMin = "", minMin = "", dayMin = "", monthMin = "";
    taskUpdate.setTitle(String.valueOf(title.getText()));
    taskUpdate.setAddress(String.valueOf(address.getText()));
    taskUpdate.setDescription(String.valueOf(description.getText()));
    if(hour.getCurrentHour() < 10){
        hourMin = "0" + hour.getCurrentHour();
    } else {
        hourMin = String.valueOf(hour.getCurrentHour());
    }
    if(hour.getCurrentMinute() < 10){
        minMin = "0" + hour.getCurrentMinute();
    } else {
        minMin = String.valueOf(hour.getCurrentMinute());
    }
    taskUpdate.setTime(hourMin + ":" + minMin);

    if(day.getDayOfMonth() < 10){
        dayMin = "0" + day.getDayOfMonth();
    } else {
        dayMin = String.valueOf(day.getDayOfMonth());
    }
    if(day.getMonth() < 10){
        monthMin = "0" + (day.getMonth()+1);
    } else {
        monthMin = String.valueOf((day.getMonth()+1));
    }
    taskUpdate.setDay(day.getYear() + "-" + monthMin + "-" + dayMin);

    taskUpdate.setPriority(priorityValues);
    taskUpdate.setIdCategory(idCategory);
    taskUpdate.setTimerTaskActive(false);
    taskUpdate.setTimerTaskCancel(true);
}
```

Ilustración 56 - Obtener datos tarea a modificar

En la siguiente Ilustración se muestra la funcionalidad de obtener la dirección de la tarea en función de la dirección, descripción o título de la tarea en el caso del que el usuario no haya elegido un punto de interés favorito para la tarea.

```
@Override
protected String doInBackground(TaskVO... args) {
    // TODO Auto-generated method stub
    JSONObject jsonResults = null;
    String success;
    _prefs =  getSharedPreferences(PREFERENCES_FILE, Context.MODE_PRIVATE);
    latitudeUser = Double.longBitsToDouble(_prefs.getLong("latitudeUser", 0));
    longitudeUser = Double.longBitsToDouble(_prefs.getLong("longitudeUser", 0));
    try {
        task = args[0];
        if(0.0 == task.getLocation().getLongitude() && 0.0 == task.getLocation().getLatitude()
                && task.getAddress() != null && !"".equals(task.getAddress())){
            address = task.getAddress().replace(" " , "+");
            json = jsonParser.makeHttpRequest(GOOGLE_MAPS_ADDRESS_URL  + address, "GET", "", null);
        }  else {
            if (0.0 == task.getLocation().getLongitude() && 0.0 == task.getLocation().getLatitude() &&
                    task.getDescription() != null && "".equals(task.getDescription())){
                address = "location=" + longitudeUser  + "," + latitudeUser + "&radius="
                        + radius + "&name=" + task.getDescription().replace(" " , "+") + "&sensor=false&key=AIzaSyAW3KbJ5BHeCV--tp-fYiQtASX5PDV_dA0" ;
                json = jsonParser.makeHttpRequest(GOOGLE_PLACES_ADDRESS_URL , "GET", "", null);
            }
            if(0.0 == task.getLocation().getLongitude() && 0.0 == task.getLocation().getLatitude()
                    && !"".equals(task.getTitle()) && task.getTitle() != null ) {
                address = "location=" + longitudeUser  + "," + latitudeUser + "&radius="
                        + radius + "&name=" + task.getTitle().replace(" " , "+") + "&sensor=false&key=AIzaSyAW3KbJ5BHeCV--tp-fYiQtASX5PDV_dA0" ;
                json = jsonParser.makeHttpRequest(GOOGLE_PLACES_ADDRESS_URL , "GET", "", null);
            }
        }

        // full json response
        success = json.getString("status");
        if ("OK".equals(success)) {
            jsonResults = (JSONObject) json.getJSONArray("results").get(0);
            JSONObject jsonGeometry = jsonResults.getJSONObject("geometry");
            JSONObject jsonLocation = jsonGeometry.getJSONObject("location");
            Location l = new Location("");
            l.setLatitude(jsonLocation.getDouble("lng"));
            l.setLongitude(jsonLocation.getDouble("lat"));
            task.setLocation(l);
            dbManager.updateTask(task);
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
}
```

Ilustración 57 - Obtener dirección

A parte se puede encontrar la llamada al servidor para crear la nueva tarea, en la Ilustración se puede observar cómo se realiza.

```
@Override
protected String doInBackground(TaskVO... args) {
    // TODO Auto-generated method stub
    try {
        task = args[0];
        // Building Parameters
        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair("title", task.getTitle()));
        params.add(new BasicNameValuePair("longitude", String.valueOf(task.getLocation().getLongitude())));
        params.add(new BasicNameValuePair("latitude", String.valueOf(task.getLocation().getLatitude())));
        params.add(new BasicNameValuePair("address", task.getAddress()));
        params.add(new BasicNameValuePair("description", task.getDescription()));
        params.add(new BasicNameValuePair("time", task.getTime()));
        params.add(new BasicNameValuePair("day", task.getDay()));
        params.add(new BasicNameValuePair("priority", task.getPriority()));
        params.add(new BasicNameValuePair("idCategory", String.valueOf(task.getIdCategory())));

        Log.d("request!", "starting");

        _prefs = getSharedPreferences("myPreferences", MODE_PRIVATE);
        idUser = _prefs.getInt("idUserCurrent", 0);

        //Posting user data to script
        JSONObject json = jsonParser.makeHttpRequest(
                CREATE_TASK_URL, "POST",dbManager.getApiKeyById(idUser), params);

        // full json response
        Log.d("CreateTask attempt", json.toString());

        // json success element
        success = json.getString("error");
        if (success == "false") {
            Log.d("Task Created!", json.toString());
            int idTask = json.getInt("idTask");
            task.setIdTaskServer(idTask);
            dbManager.updateIdTaskServer(task);
            return json.getString(TAG_MESSAGE);
        }else{
            Log.d("Login Failure!", json.getString(TAG_MESSAGE));
            return json.getString(TAG_MESSAGE);

        }
    } catch (JSONException e) {
        e.printStackTrace();
    }

    return null;
}
```

Ilustración 58 - Llamada al servidor para crear tarea

Y por último, se encuentra el método que realiza la llamada al servidor para modificar la tarea. En la Ilustración se puede visualizar como se realiza la siguiente funcionalidad.

```java
@Override
protected String doInBackground(TaskVO... args) {
    // TODO Auto-generated method stub
    try {
        task = args[0];
        // Building Parameters
        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair("title", task.getTitle()));
        params.add(new BasicNameValuePair("longitude", String.valueOf(task.getLocation().getLongitude())));
        params.add(new BasicNameValuePair("latitude", String.valueOf(task.getLocation().getLatitude())));
        params.add(new BasicNameValuePair("address", task.getAddress()));
        params.add(new BasicNameValuePair("description", task.getDescription()));
        params.add(new BasicNameValuePair("time", task.getTime()));
        params.add(new BasicNameValuePair("day", task.getDay()));
        params.add(new BasicNameValuePair("priority", task.getPriority()));
        params.add(new BasicNameValuePair("idCategory", String.valueOf(task.getIdCategory())));

        Log.d("request!", "starting update");

        _prefs = getSharedPreferences("myPreferences", MODE_PRIVATE);
        idUser = _prefs.getInt("idUserCurrent", 0);

        Log.v("VERBOSE", "Valor de la url update task: " + UPDATE_TASK_URL + "" + task.getIdTask());

        //Posting user data to script
        JSONObject json = jsonParser.makeHttpRequest(
                UPDATE_TASK_URL + task.getIdTask(), "PUT", dbManager.getApiKeyById(idUser), params);

        // full json response
        Log.d("UpdateTask attempt", json.toString());

        // json success element
        success = json.getString("error");
        if (success == "false") {
            Log.d("Task Updated!", json.toString());
            return json.getString(TAG_MESSAGE);
        }else{
            Log.d("Login Failure!", json.getString(TAG_MESSAGE));
            return json.getString(TAG_MESSAGE);
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
}
/**
```

Ilustración 59 - Llamada al servidor para modificar tarea

8.1.8.2 *MainActivity*: Clase principal que se encarga de gestionar las pantallas que se encuentran en el despegable del menú principal, también recoge los datos de las tareas y las sincroniza. Además, gestiona las diferentes funcionalidades de los menús de cada barra de estado de las pantallas. A continuación, en las Ilustraciones se muestran los diferentes métodos que implementan estas funcionalidades.

La primera de todas las Ilustraciones muestra la gestión de las pantallas del menú despegable principal (Navigation Drawer), se puede observar comen función de la pantalla seleccionada se lanzara la clase correspondiente que pinta cada pantalla.

```
     * @param position
     */
    @Override
    public void onNavigationDrawerItemSelected(int position) {
        // Update the main content by replacing fragments
        Bundle bundle = getIntent().getExtras();
        if (bundle != null) {
            position = 0;
        }
        currentFragment = position;
        switch (position + 1) {
            case 1:
                if (mapFragment == null) {
                    mapFragment = new MapFragment();
                }
                if (bundle != null) {
                    mapFragment.setArguments(bundle);
                    Bundle clearBundle = null;
                    getIntent().replaceExtras(clearBundle);
                }
                replaceFragment(mapFragment, "Map");
                break;
            case 2:
                taskFragment = new TaskFragment();
                taskFragment.setTasks(taskList);
                replaceFragment(taskFragment, "Task Manager");
                break;
            case 3:
                if (poiFragment == null) {
                    poiFragment = new PoiFragment();
                }
                if (bundle != null) {
                    poiFragment.setArguments(bundle);
                    Bundle clearBundle = null;
                    getIntent().replaceExtras(clearBundle);
                }
                replaceFragment(poiFragment, "Poi Manager");
                break;
            case 4:
                otherTaskManagerFragment = new OtherTaskManagerFragment();
                replaceFragment(otherTaskManagerFragment, "Google Tasks");
                break;
            case 5:
                Intent i = new Intent(MainActivity.this, SettingsActivity.class);
                startActivity(i);
                break;
        }
```

Ilustración 60 - Selección de la pantalla del menú principal

En la siguiente Ilustración se puede visualizar los diferentes menús que se cargan en la barra de estado en función de la pantalla seleccionada por el usuario.

```java
/**
 * Method responsible for managing the menus of different fragments
 * @param menu
 * @return
 */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    if (!navigationDrawerFragment.isDrawerOpen() && (currentFragment == 0 || currentFragment == 1)) {
        MenuItem m = null;
        // Only show items in the action bar relevant to this screen
        // if the drawer is not showing. Otherwise, let the drawer
        // decide what to show in the action bar.
        getMenuInflater().inflate(R.menu.main, menu);

        m = menu.findItem(R.id.action_search);
        searchView = (SearchView) m.getActionView();
        searchView.setQueryHint("Search task");
        searchView.setOnQueryTextListener(this);

        restoreActionBar();
        return true;
    } else if (!navigationDrawerFragment.isDrawerOpen() && currentFragment == 2 ){
        getMenuInflater().inflate(R.menu.menu_poi_fragment, menu);
        restoreActionBar();
        return true;
    } else if (!navigationDrawerFragment.isDrawerOpen() && currentFragment == 3){
        getMenuInflater().inflate(R.menu.menu_othe_task_fragment, menu);
        restoreActionBar();
        return true;
    }
    return super.onCreateOptionsMenu(menu);
}
```

Ilustración 61 - Carga del menú

Seguidamente se puede visualizar en la Ilustración el método que gestiona todos los botones que contiene cada menú. En la Ilustración se puede ver que se lanza otras pantallas en función de lo que el usuario pulse.

```java
/**
 * Method that depending on the selected menu button, makes your respective action
 * @param item
 * @return
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
    switch (id){
        case R.id.action_search:
            try {
                doMySearch(String.valueOf(searchView.getQuery()));
            } catch (JSONException e) {
                e.printStackTrace();
            }
            break;
        case R.id.action_settings:
            Intent i = new Intent(MainActivity.this, SettingsActivity.class);
            startActivity(i);
            break;
        case R.id.synchronize:
            new AttemptTask().execute();
            checkAddressLocation();
            replaceFragment(mapFragment, getString(R.string.title_map));
            break;
        case R.id.addTask:
            Intent in = new Intent(MainActivity.this, AddTaskActivity.class);
            finish();
            startActivity(in);
            break;
        case R.id.addPoi:
            Intent inte = new Intent(MainActivity.this, PoiActivity.class);
            finish();
            startActivity(inte);
            break;
        case R.id.synchronize_google_tasks:
            new AttemptSynchronizeGoogleTasks().execute();
            break;
    }
    if (id == R.id.action_settings) {
        return true;
    } else if (id == R.id.action_search) {
        return true;
    }
```

Ilustración 62 - Selección de los botones

A continuación, en la Ilustración de la parte de abajo se encuentra el método de búsqueda de la tarea, se encarga de comprobar primero de todo si hay conexión si la hay solo hace una llamada a la clase que conecta con el servidor y los guardara en la base de datos para evitar pérdidas de información. En el caso de no tener conexión la búsqueda se realizara a través de un método de la base de datos local. En cualquiera de los casos estos datos serán pasados a la clase que gestiona la lista de tareas (TaskFragment).

```
/**
 * Method seeking user-task indicates
 * @param queryStr
 * @throws JSONException
 */
private void doMySearch(String queryStr) throws JSONException {
    taskListSearch = new ArrayList<TaskVO>();
    if(cd.isConnectingToInternet()){
        new AttemptTaskSearch().execute(queryStr);
        if(taskListSearch.isEmpty()){
            taskListSearch = dbManager.getTaskByName(queryStr);}
    } else {
        taskListSearch = dbManager.getTaskByName(queryStr);
    }


    if (taskListSearch.isEmpty()){
        Toast.makeText(this, "Task introduced is not valid, please try again", Toast.LENGTH_LONG).show();
    } else {
        taskFragment = new TaskFragment();
        extras.putParcelableArrayList("taskList", taskListSearch);
        taskFragment.setArguments(extras);
        replaceFragment(taskFragment, getString(R.string.title_task));
    }
}
```

Ilustración 63 - Búsqueda tarea

Justo después en la Ilustración se puede visualizar el método que se encarga de hacer la llamada al servidor para que le devuelva la lista con las tareas que coincidan en su título y descripción con la cadena introducida por el usuario.

```
@Override
protected String doInBackground(String... args) {
    // TODO Auto-generated method stub
    JSONObject jsonTask = null;
    String success;
    Uri imageUri = null;
    try {
        JSONObject json = jsonParser.makeHttpRequest(TASK_URL_SEARCH + args[0].trim(), "GET",
                dbManager.getApiKeyById(idUser), null);
        // json success tag
        success = json.getString("error");
        if ("false".equals(success)) {
            for (int i = 0; i < json.getJSONArray("message").length(); i++) {
                jsonTask = (JSONObject) json.getJSONArray("message").get(i);
                Location l = new Location("");
                l.setLatitude(jsonTask.getDouble("latitude"));
                l.setLongitude(jsonTask.getDouble("longitude"));
                int idCategory = jsonTask.getInt("idCategory");
                TaskVO task = new TaskVO(jsonTask.getInt("idTask"),
                        0,
                        jsonTask.getString("title"),
                        1,
                        jsonTask.getString("address"),
                        jsonTask.getString("last_modify"),
                        jsonTask.getString("description"),
                        jsonTask.getString("time"),
                        jsonTask.getString("day"),
                        jsonTask.getString("priority"),
                        idCategory,
                        idUser, imageUri);
                taskListSearch.add(task);
            }
```

Ilustración 64 - Búsqueda de la tarea en el servidor

En la próxima Ilustración se muestra el método que realiza la sincronización de tareas con el servidor. Esta elimina todas las tareas locales y se descarga todas las del servidor. Esto se realiza así porque todas las tareas que son creadas son automáticamente subidas al servidor por eso motivo no hay peligro de que se pierda información al borrar.

```
@Override
protected String doInBackground(String... args) {
    // TODO Auto-generated method stub
    JSONObject jsonTask = null;
    String success;
    Uri imageUri = null;
    try {
        dbManager.deleteAllTask();
        JSONObject json = jsonParser.makeHttpRequest(TASK_URL, "GET", dbManager.getApiKeyById(idUser), null);

        // json success tag
        success = json.getString("error");
        if ("false".equals(success)) {
            if (!"null".equals(json.getString("message"))){
                for (int i = 0; i < json.getJSONArray("message").length(); i++) {
                    jsonTask = (JSONObject) json.getJSONArray("message").get(i);
                    Location l = new Location("");
                    l.setLatitude(jsonTask.getDouble("latitude"));
                    l.setLongitude(jsonTask.getDouble("longitude"));
                    int idCategory = jsonTask.getInt("idCategory");
                    TaskVO task = new TaskVO(jsonTask.getInt("idTask"),
                            0,
                            jsonTask.getString("title"),
                            1,
                            jsonTask.getString("address"),
                            jsonTask.getString("last_modify"),
                            jsonTask.getString("description"),
                            jsonTask.getString("time"),
                            jsonTask.getString("day"),
                            jsonTask.getString("priority"),
                            idCategory,
                            idUser, imageUri);
                    Log.v("VERBOSE", "ATTEMPTASK  inserto la tarea num: " + task.getIdTask());
                    taskList.add(task);
                    Log.v("VERBOSE", taskList.size() +  "Valores " +  taskList.toString());
                    dbManager.addTask(task);
                }
            } else {
                //Obtain current date to initial task
                Calendar c = Calendar.getInstance();
                SimpleDateFormat formatedText = new SimpleDateFormat("yyyy-MM-dd HH:mm");
                String strDateInitial = "", day, time;
                Date dateInitial =  c.getTime();
                strDateInitial = formatedText.format(dateInitial);
                day = strDateInitial.substring(0,10);
                time = strDateInitial.substring(11,strDateInitial.length());
```

Ilustración 65 - Descarga y sincronización de tareas con el servidor

El siguiente método que se visualiza en la siguiente Ilustración es el que realiza la sincronización manual de la pantalla de Google Task. Simplemente se realiza una llamada al servidor que se encarga de sincronizar todas entre el servidor y Google Task y al mismo tiempo también llama al método encargado de descargar todas las tareas.

```
@Override
protected String doInBackground(String... args) {
    // TODO Auto-generated method stub
    JSONObject jsonGoogleTasks = jsonParser.makeHttpRequest(SYNCHRONIZE_TASK_URL + idUser, "GET", "", null);

    try{
        // json success tag
        success = jsonGoogleTasks.getString("error");
        if ("false".equals(success)) {
            new AttemptTask().execute();
        } else {
            Toast.makeText(getApplicationContext(), "Failed synchronize , please try again", Toast.LENGTH_SHORT).show();
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }

    return null;
}
```

Ilustración 66 - Descarga manual

Y por último, se puede observar en la Ilustración siguiente el método encargado de gestionar la pulsación de botón para atrás del dispositivo en este caso se ha tenido en cuenta que si se pulsa dos veces el usuario podrá cerrar la aplicación.

```
/**
 * Method that controls the button back to back, in the case of giving twice gets you out of the application
 */
@Override
public void onBackPressed() {
    if (doubleBackToExitPressedOnce) {
        super.onBackPressed();
        return;
    }

    this.doubleBackToExitPressedOnce = true;
    Toast.makeText(this, "Please click BACK again to exit", Toast.LENGTH_SHORT).show();

    new Handler().postDelayed(() -> {
            doubleBackToExitPressedOnce=false;
    }, 2000);

    mapFragment = new MapFragment();
    replaceFragment(mapFragment, "Map");

}

/**
```

Ilustración 67 - Botón Back

8.1.8.3 *PoiActivity*: Clase encargada de mostrar al usuario las pantallas de crear y modificar sus puntos de interés favoritos, inicialmente la aplicación lleva unos cuantos puntos de interés de muestra. Hay que tener en cuenta que los puntos de interés favoritos del usuario se guardan internamente en la base de datos del dispositivo. A continuación, se explicaran los métodos de los que está compuesta la clase.

En primer todo que nos muestra la siguiente Ilustración es el encargado de instanciar los objetos y de tener en cuenta si se ha pulsado el mapa para la localización del punto de interés en el caso de que se pulse este método se encargará de hacer la llamada al método que recupera la dirección que será explicado más adelante.

```
/**
 * Method that initializes the parameters, and if the map is clicked the address stored
 * in the corresponding text box that initializes the parameters
 * @param savedInstanceState
 */
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add_poi);
    getActionBar().setTitle("POIs Management");
    dbManager = dbManager.getInstance(this);
    getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_HIDDEN);

    if (googleMap == null)
        googleMap = ((MapFragment) getFragmentManager().findFragmentById(R.id.fragment_poi_map_view)).getMap();
    googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);

    jsonParser = new JSONParser();
    initializeComponents();

    googleMap.setOnMapClickListener((point) -> {
            // TODO Auto-generated method stub
            String addressPosition = "";
            positionPoi = point;
            if (markerAddress != null)
                markerAddress.remove();
            markerAddress = googleMap.addMarker(new MarkerOptions().position(point));
            try {
                 addressPosition = new AttemptLocation().execute().get();
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (ExecutionException e) {
                e.printStackTrace();
            }
            address.setText(addressPosition);
    });

}
```

Ilustración 68 - Obtención de dirección del punto de interés

El siguiente método que se visualiza en la Ilustración es el encargado de recoger los datos que el usuario a introducido y crear en la base de datos local el nuevo punto de interés.

```
/**
 * Add new Poi
 */
private void addNewPoi() {
    PoiVO newPoi = new PoiVO();
    int idPoi = dbManager.getMaxIdPoi() + 1;
    newPoi.setIdPOI(idPoi);
    newPoi.setTitle(String.valueOf(title.getText()));
    Location l = new Location("");
    l.setLatitude(positionPoi.latitude);
    l.setLongitude(positionPoi.longitude);
    newPoi.setLocation(l);
    newPoi.setAddress(String.valueOf(address.getText()));
    newPoi.setDescription(String.valueOf(description.getText()));
    newPoi.setIdCategory(idCategory);
    newPoi.setIdUser(getIdUser());
    dbManager.addPoi(newPoi);
}
```

Ilustración 69 - Añadir punto de interés

A continuación, en la Ilustración se muestra la gestión del botón de la barra de estado del menú de la pantalla. Es el encargado de borrar el punto de interés que tienes seleccionado, en el caso de que el punto de interés no exista se le notificara al usuario con un mensaje.

```
/**
 *Method that controls the pulsation of the buttons on the menu bar state,
 *  in this case only the button is deleted from the local database the sights removed.
 * @param item
 * @return
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.menu_item_delete_poi:
            if (poiUpdate != null) {
                dbManager.deletePoi(poiUpdate.getIdPOI());
                Intent inte = new Intent(this, MainActivity.class);
                startActivity(inte);
            } else {
                Toast toast = Toast.makeText(this, "There isn´t Poi to delete", Toast.LENGTH_SHORT);
                toast.show();
            }
            break;
        default:
            return super.onOptionsItemSelected(item);

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

Ilustración 70 - Menú borrar punto de interés

En la última Ilustración se encuentra el método encargado de modificar un punto de interés en la base de datos local con los parámetros que ha introducido el usuario.

8.1.8.4 *TaskActivity:* Clase que crea la pantalla donde se muestra toda la información de la tarea que el usuario haya elegido de la lista de la clase TaskFragment comentada anteriormente. Seguidamente se explicaran las distintas funcionalidades que implementa:

En la primera Ilustración se puede visualizar el método encargado de gestionar los diferentes botones del menú, como se puede ver hay cuatro botones el primero es el encargado de llamar al método de compartir tarea, el segundo llama al método de conexión con el servidor para borrar la tarea, el siguiente botón manda los datos de localización de la tarea seleccionada para que cuando se pulse este botón te redirija al mapa y este pinte esta tarea y por último el botón encargado de abrir la actividad que permitirá al usuario modificar la tarea.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.menu_item_share:
            shareTask();
            break;
        case R.id.menu_item_delete_task:
            try {
                DeleteTask deleteTask = new  DeleteTask();
                synchronized (deleteTask) {
                    deleteTask.execute(task).wait(1000);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            Intent inte = new Intent(this, MainActivity.class);
            startActivity(inte);
            break;
        case R.id.menu_item_point:
            Intent intent = new Intent(this, MainActivity.class);
            if (task.getLocation().getLongitude() == 0 && task.getLocation().getLatitude() == 0){
                intent.putExtra(KEY_ID_TASK, task.getIdTask());
                List<PoiVO> poisList = dbManager.getPoisByCategory(task.getIdCategory());
                if (!poisList.isEmpty()) {
                    intent.putExtra(KEY_LATITUDE, poisList.get(0).getLocation().getLatitude());
                    intent.putExtra(KEY_LONGITUDE, poisList.get(0).getLocation().getLongitude());
                }
            } else {
                intent.putExtra(KEY_ID_TASK, task.getIdTask());
                intent.putExtra(KEY_LATITUDE, task.getLocation().getLatitude());
                intent.putExtra(KEY_LONGITUDE, task.getLocation().getLongitude());
            }
            startActivity(intent);
            break;
        case R.id.menu_item_update_task:
            Intent in = new Intent(this, AddTaskActivity.class);
            in.putExtra("idTask", task.getIdTask());
            finish();
            startActivity(in);
            break;
        default:
            return super.onOptionsItemSelected(item);

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

Ilustración 71 - Gestión del menú principal

En la siguiente Ilustración se muestra el método encargado de conectar con el servidor para eliminar la tarea. Además, también la borra de la base de datos local.

```
@Override
protected String doInBackground(TaskVO... args) {
    // TODO Auto-generated method stub
    try {
        task = args[0];

        Log.d("request!", "starting delete");

        _prefs = getSharedPreferences("myPreferences", MODE_PRIVATE);
        idUser = _prefs.getInt("idUserCurrent", 0);

        //Posting user data to script
        JSONObject json = jsonParser.makeHttpRequest(
                DELETE_TASK_URL + task.getIdTask(), "DELETE", dbManager.getApiKeyById(idUser), null);

        // full json response
        Log.d("DeleteTask attempt", json.toString());

        // json success element
        success = json.getString("error");
        if (success == "false") {
            Log.d("Task Delete!", json.toString());
            dbManager.deleteTask(task.getIdTask());
            return json.getString(TAG_MESSAGE);
        } else {
            Log.d("Delete Failure!", json.getString(TAG_MESSAGE));
            return json.getString(TAG_MESSAGE);

        }
    } catch (JSONException e) {
        e.printStackTrace();
    }

    return null;

}
```

Ilustración 72 - Eliminar tarea del servidor

Seguidamente se encuentra el método de compartir que se puede observar en la siguiente Ilustración, este procedimiento simplemente hace una llamada a todas las aplicaciones que tenga el usuario para poder compartir datos. Permitiendo que el usuario seleccione una y mande su tarea a cualquier persona.

```
/**
 * Share a task
 */
private void shareTask() {
    // Create share intent
    Intent shareIntent = null;
    shareIntent = ShareCompat.IntentBuilder.from(this)
            .setText("Hi, I'm sharing with you my task: \n " +
                    task.getTitle() + "\n in address" + task.getAddress() +
                    "\n at " + task.getTime() + " " + task.getDay())
            .setType("text/plain")
            .setChooserTitle("Where do you want to share?")
            .createChooserIntent();
    startActivity(shareIntent);
}
```

Ilustración 73 - Compartir tarea

Y por último, se puede visualizar en la Ilustración el método encargado de cargar las imágenes de cada tarea en función su categoría, si esta categoría no contuviera imagen se cogería la imagen de la categoría padre.

```
/**
 * Method commissioned to show the picture of the task as a function of the category to which it belongs
 */
  public void loadTaskImages(){
      if (statusExternalStorage()) {
          int j = 0;
          List<CategoryVO> categoriesParents = dbManager.getAllCategoryParentsByCategory(task.getIdCategory());
          CategoryVO category = categoriesParents.get(0);
          while ("".equals(category.getImageCategory())) {
              j++;
              category = categoriesParents.get(j);
          }
          File file = new File(getImagePath() + category.getImageCategory());
          imageUri = Uri.fromFile(file);
      }
      taskImageView = (ImageView) findViewById(R.id.image_header);
      taskImageView.setImageURI(imageUri);
  }
```

Ilustración 74 - Carga de las imagenes para las tareas

Después, de haber explicado la implantación se va comentar las diferentes pruebas que se han realizado y con los problemas que se han planteado.

9. Versión final

Aquí se muestra el resultado final de la aplicación desarrollada por los dos integrantes del grupo a lo largo de este proyecto con unas capturas de pantalla. Se puede probar la versión final con la apk instable suministrada.

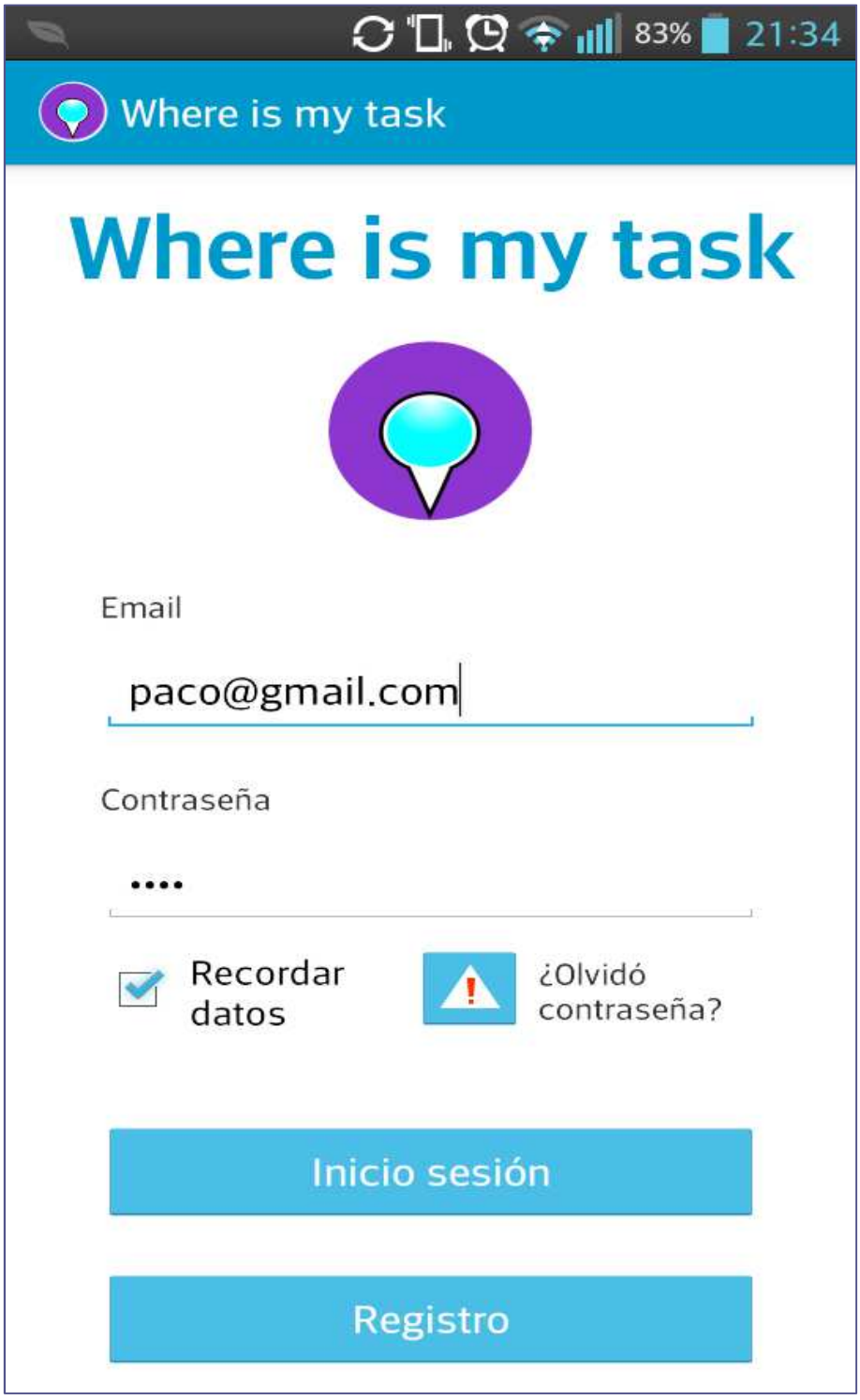


Ilustración 75 - Pantalla inicio

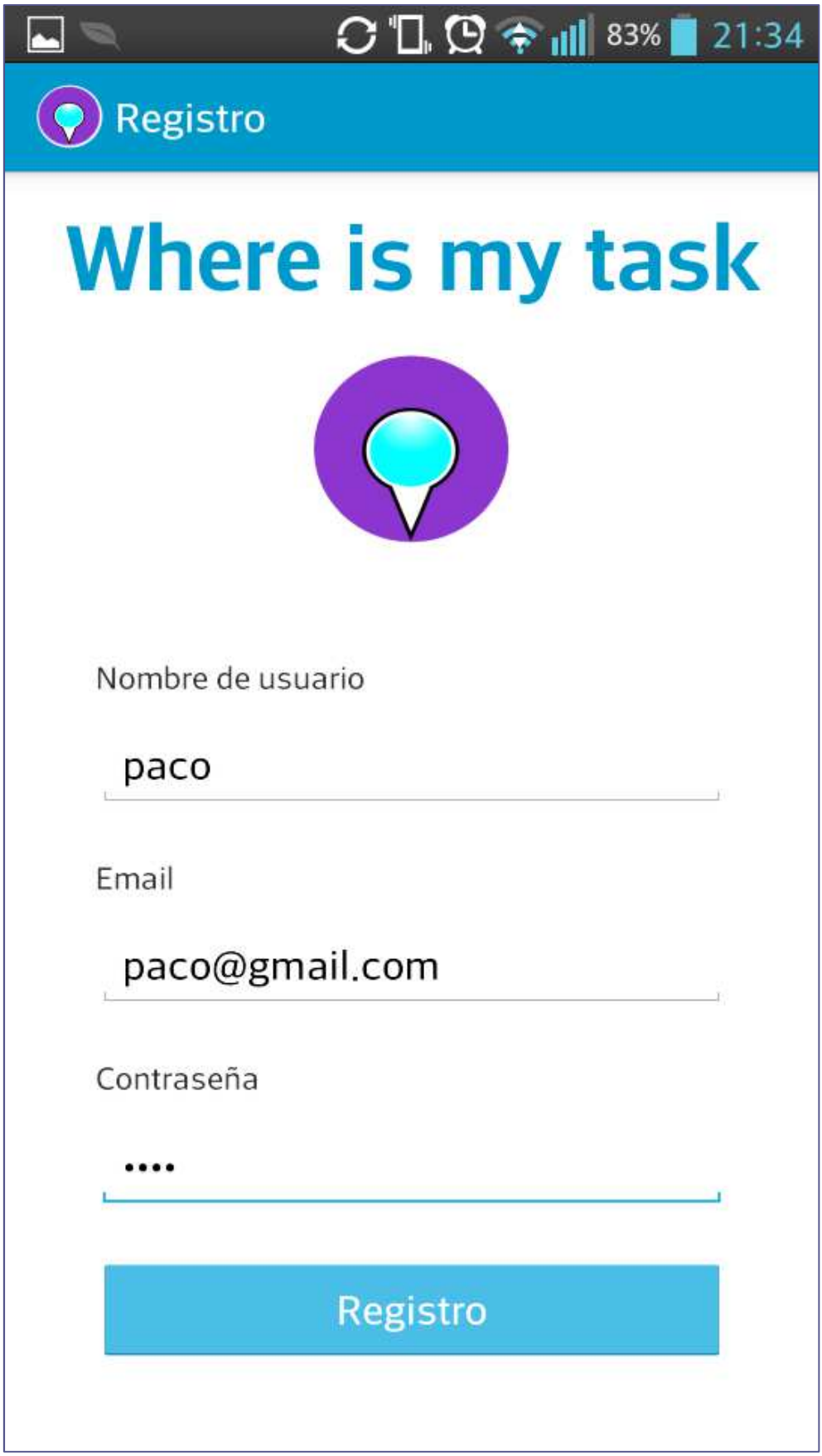


Ilustración 76 - Pantalla registro

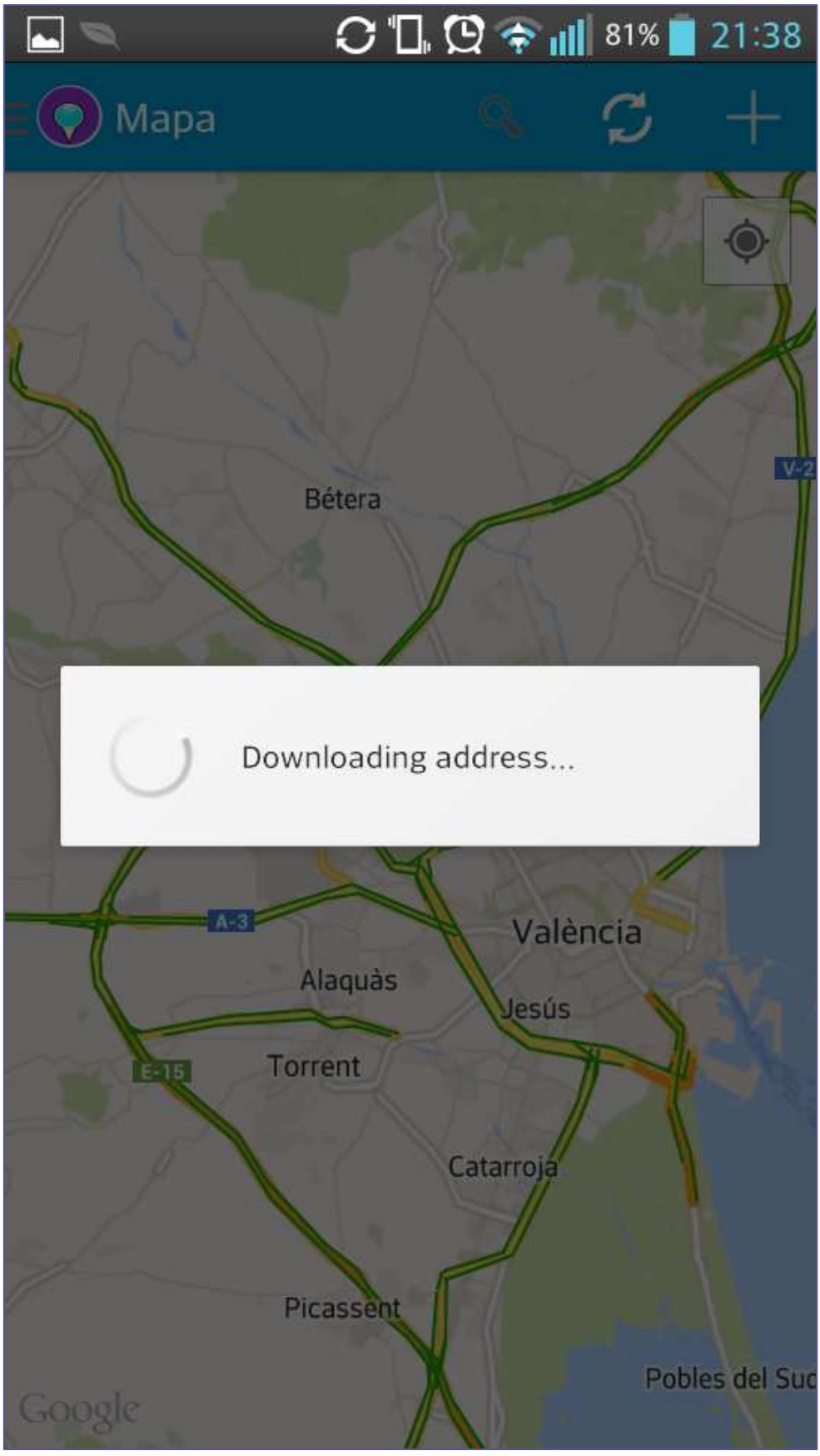


Ilustración 77 - Sincronización de datos

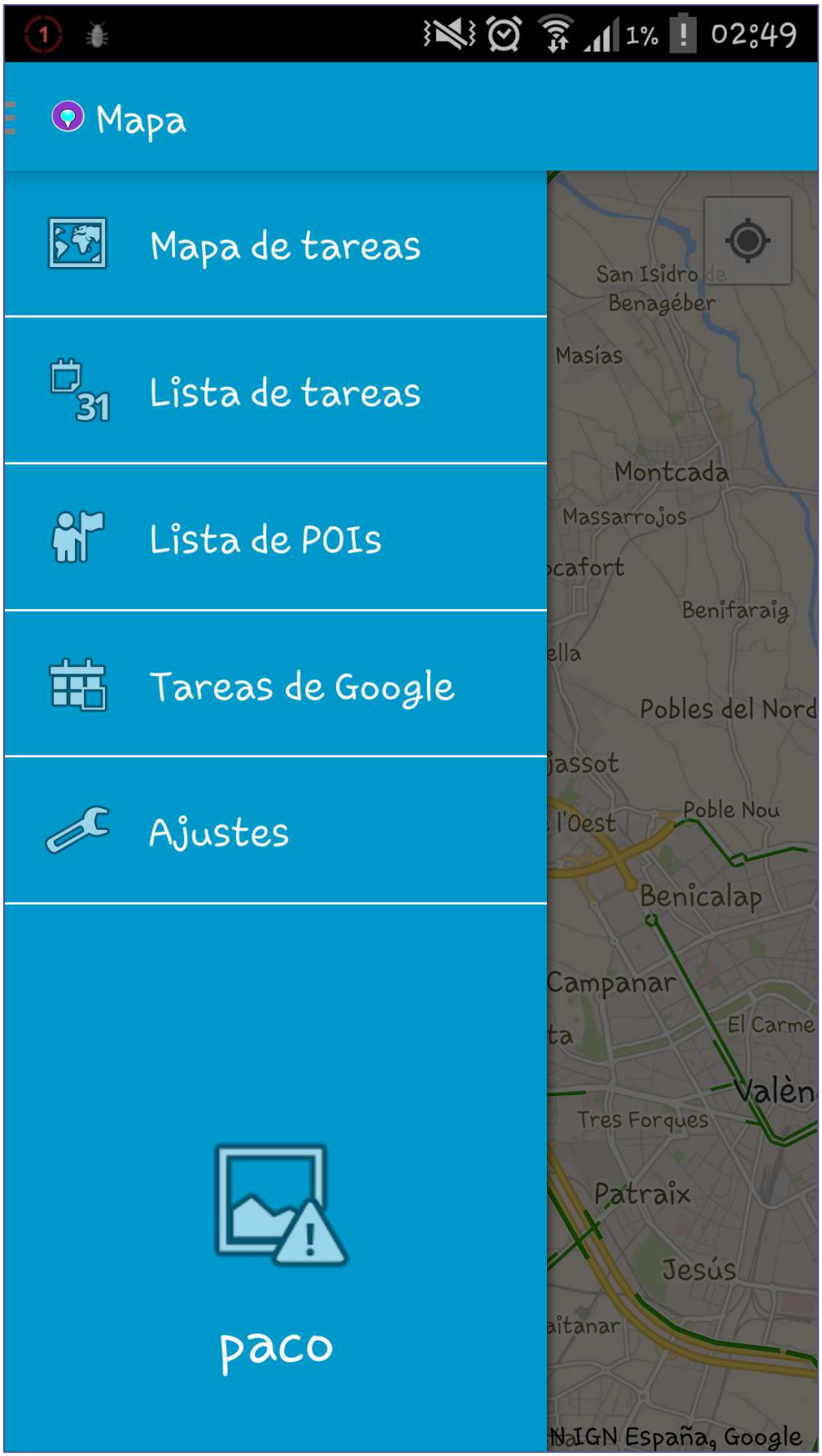


Ilustración 78 - Menú principal

Ilustración 79 - Visualización de mapa

Ilustración 80 - Visualización de tarea seleccionada

Ilustración 81 - Lista de tareas

Ilustración 82 - Tarea seleccionada

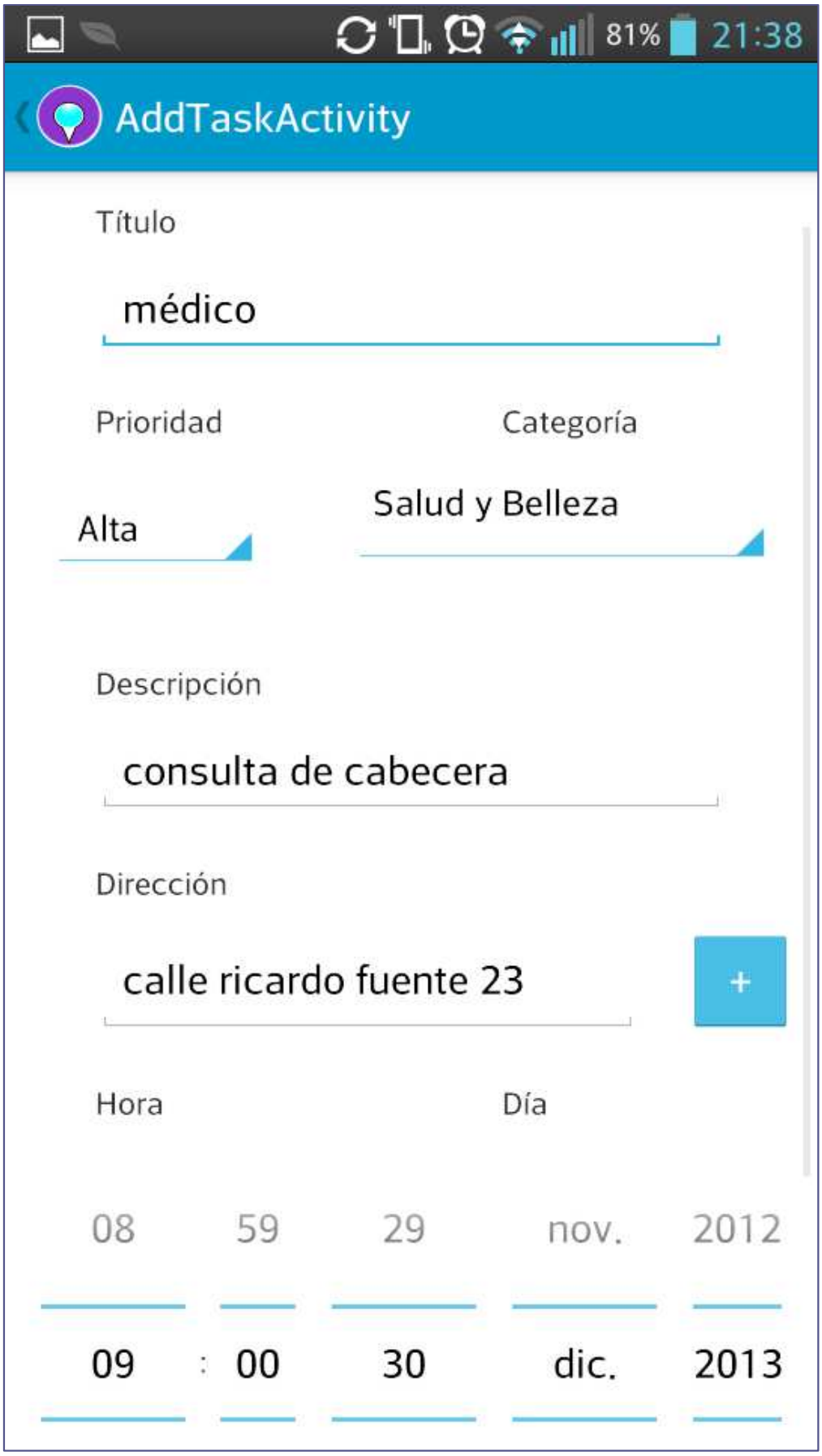


Ilustración 83 - Añadir tarea

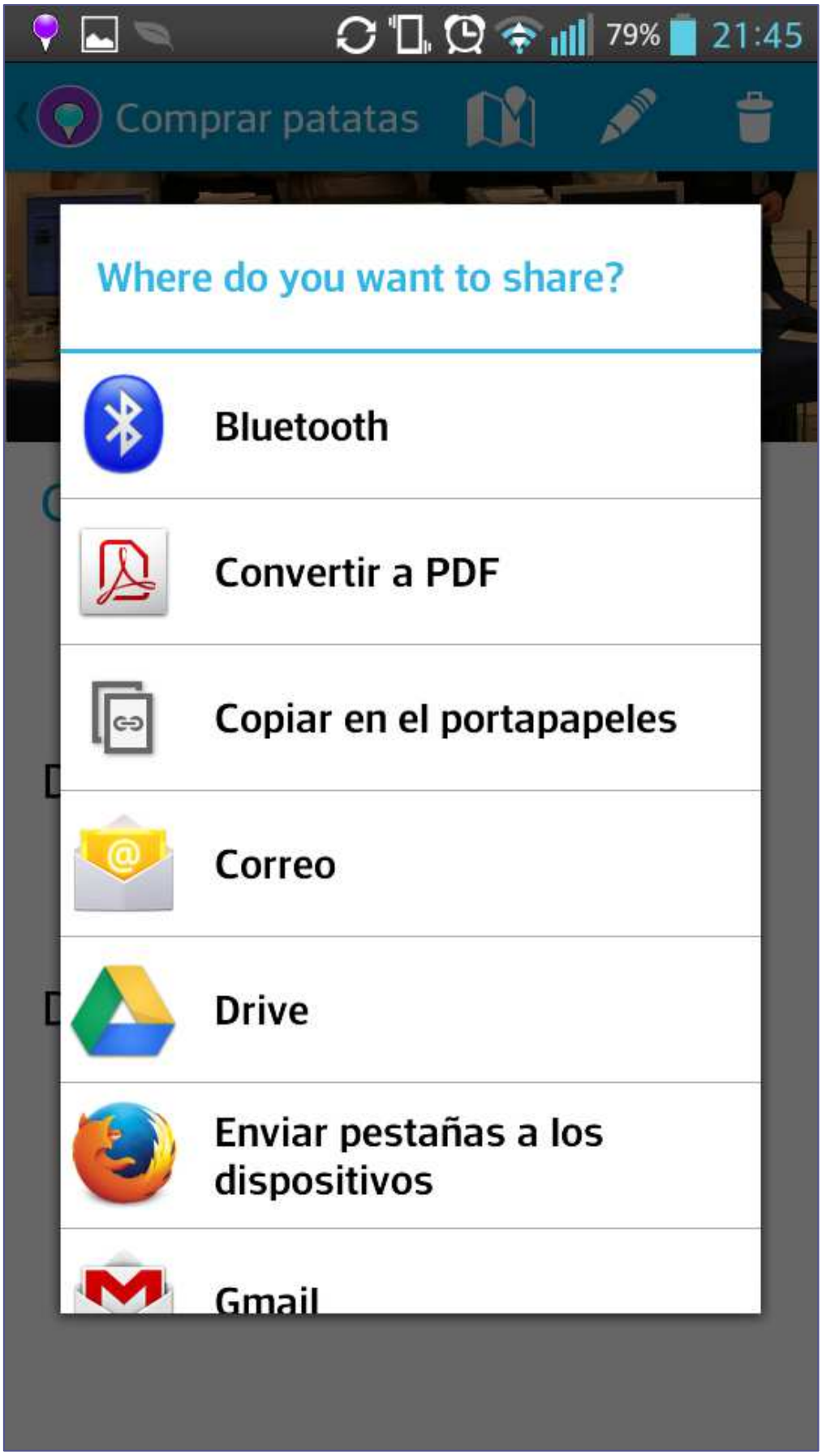


Ilustración 84 - Compartir tarea

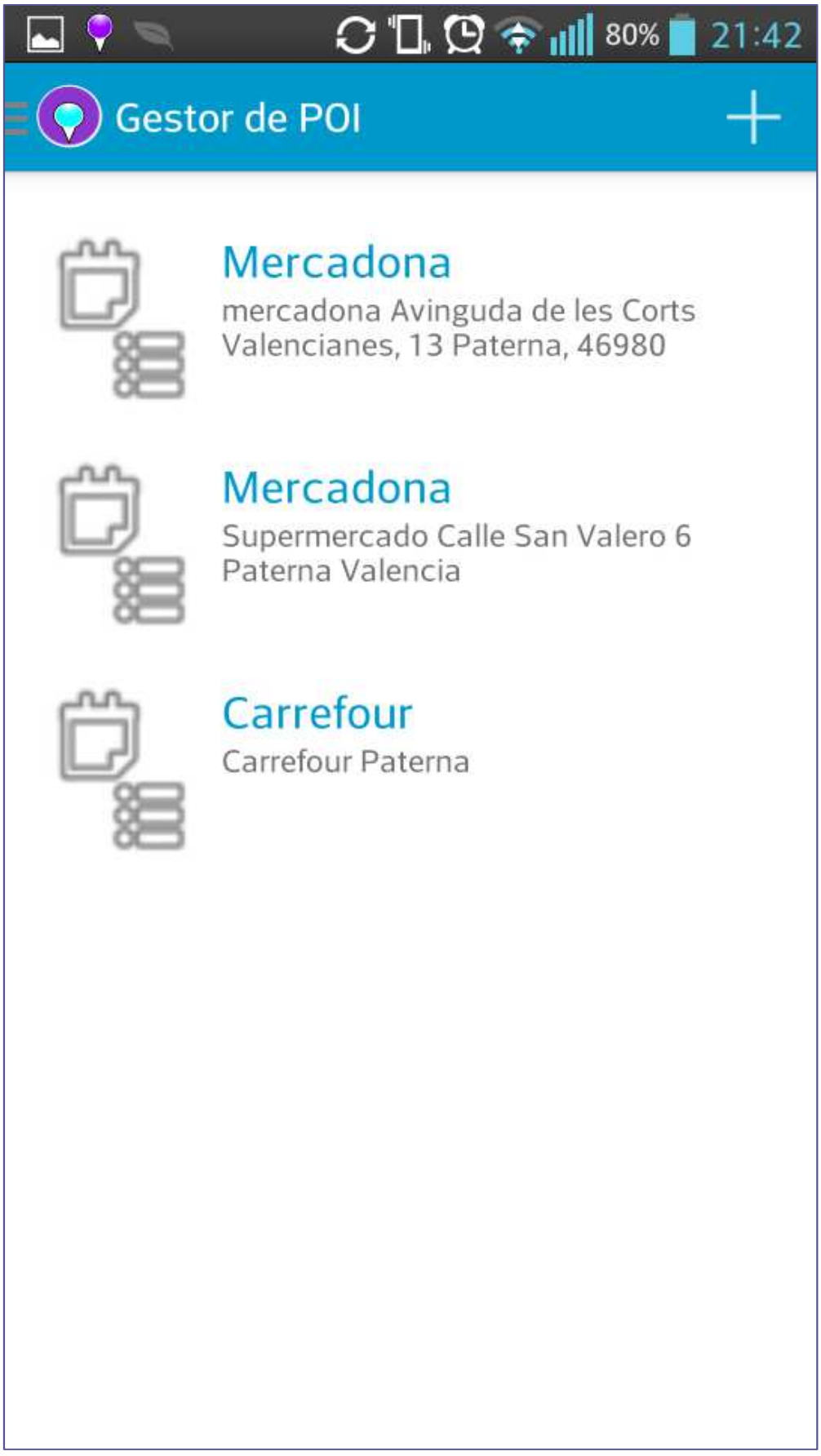


Ilustración 85 - Lista puntos de interés

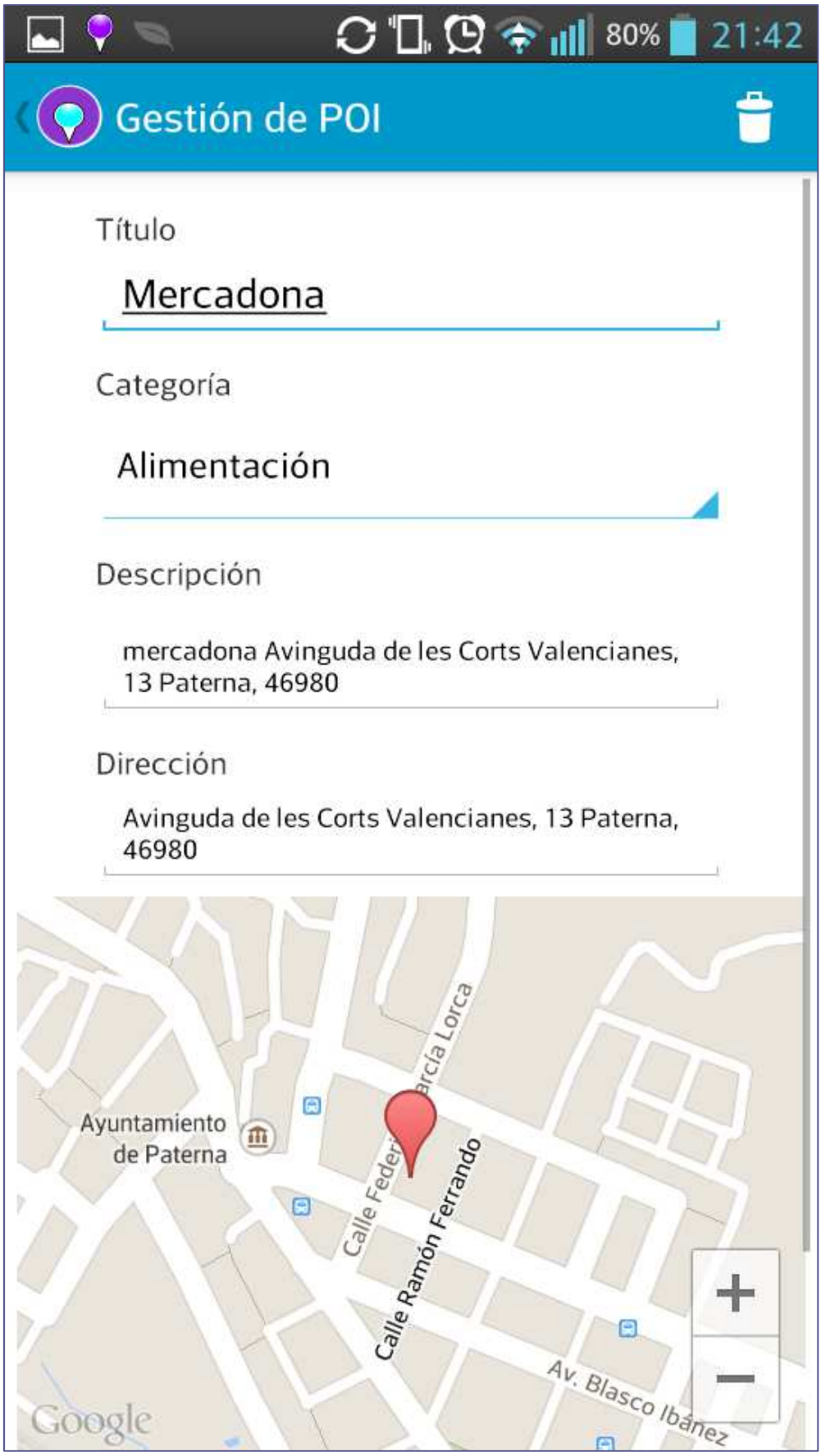


Ilustración 86 - Gestión punto de interés

Ilustración 87 - Sincronizar con Google Tasks

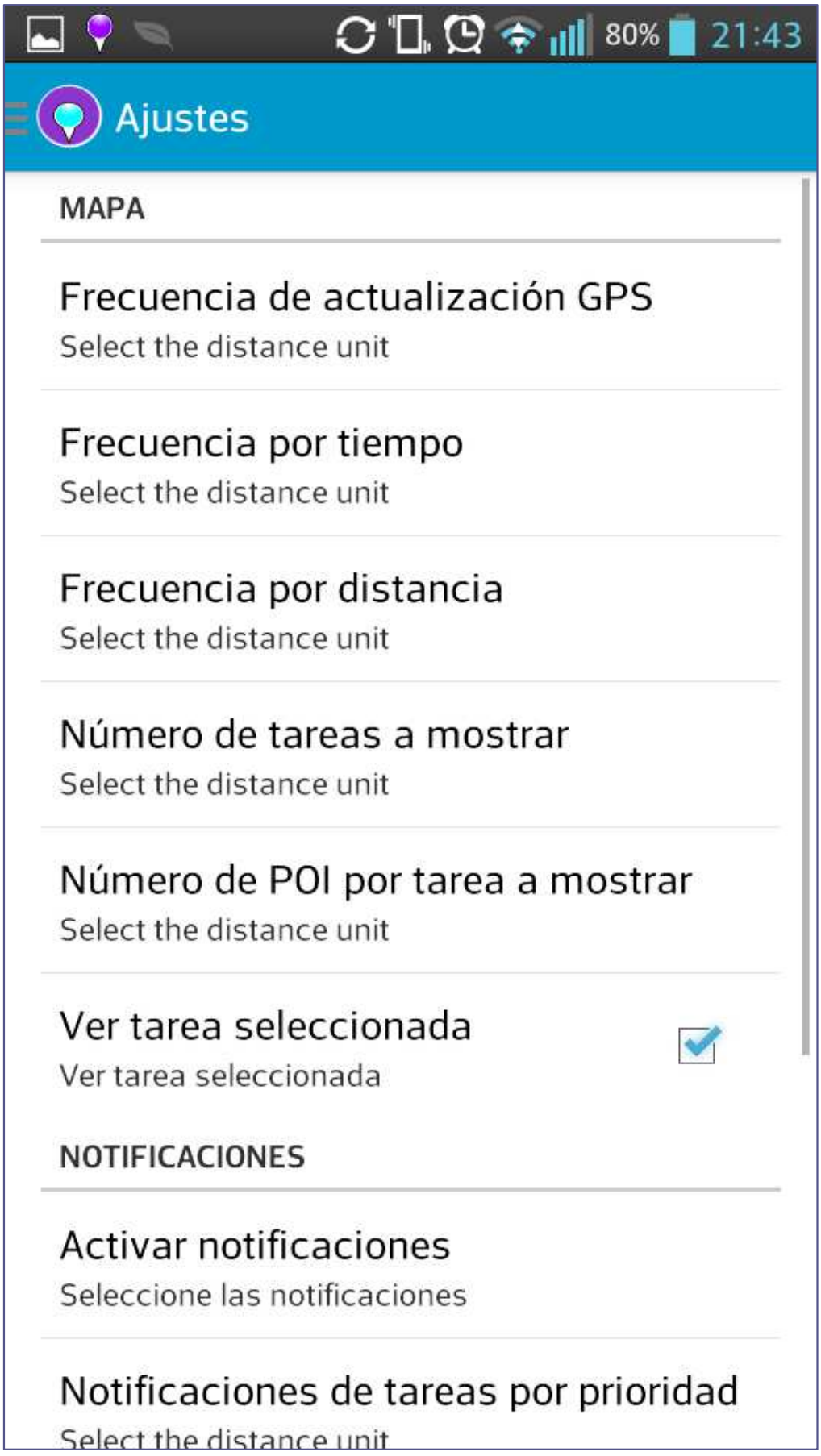


Ilustración 88 - Ajustes

Ilustración 89 - Búsqueda de tarea

10. Pruebas

En este apartado se va a comentar las diferentes pruebas realizadas en esta memoria se definen las realizadas por mi parte en la memoria de mi compañero se encuentran las realizadas por él. La gran mayoría han sido desarrolladas sin problemas aunque ciertas funcionalidades han dado ciertos problemas, ellas son:

10.1 Notificar:

10.1.1 Notificaciones de cercanía: Estas notificaciones cuando las recibe el usuario y son pulsadas le llevan automáticamente a la posición de la tarea en el mapa. Esto nos dio problemas con el paso de la tarea, ya que esta funcionalidad es la misma que la procedente de seleccionar tarea de la clase TaskActivity. Y no había manera para indicarle de dónde provenía. La solución fue utilizar el mismo tipo de dato a través de variables globales del sistema (Shared Preferences).

10.1.2 Notificaciones de tiempo: Se encargan de notificar x minutos antes al usuario de la tarea que tenía que realizar, se creó por cada tarea de este tipo diferentes tareas programadas. Pero por algún motivo desconocido las tareas programadas saltan el orden y la hora que quieren. Se estuvo investigando bastante el tema pero por tiempo no se pudo arreglar y de momento no la muestra cuando toca.

10.2 Repintar el mapa: Esta funcionalidad da problemas cuando se proviene de otra pantalla o se modificaba un ajuste del mapa. Esta implementado un método que repinta el mapa, el cual debe ser usado cuando se den estos casos.

11 Conclusiones

Tras la realización de este proyecto se evalúa el trabajo realzado y el nivel de cumplimiento de objetivos. Tal como se especificó en el apartado de objetivos se procede a detallar el nivel de implementación llevado a cabo.

La implementación del mapa y posición geográfica del usuario ha sido una de las funcionalidades más importante y costosas de realizar. En concreto la visualización de los marcadores adecuados a cada momento. A pesar de ello se ha conseguido la funcionalidad esperada con añadidos como un algoritmo propio de actualización del GPS.

Otro de los objetivos más costosos fue la sincronización con el gestor de Google Tasks, por el modo de trabajo offline sin intervención del usuario. Pero finalmente se consiguió solucionar, e incluso habilitar al usuario a subir sus tareas, lo cual proporciona mayor integración.

La gestión de la información en las base de datos no fue tan difícil pero si costosa como se esperaba. La inserción de información genérica para el uso de la aplicación (categorías y puntos de interés) se ha reducido al mínimo para realizar pruebas por falta de tiempo.

El diseño y posterior implementación de la capa REST se ha realizado tal y como se había esperado debido a la experiencia previa en la asignatura de IAP (Integración de aplicaciones) y otros proyectos.

El sistema de notificaciones ha sido costoso pero es asumible porque aún no teniendo conocimientos previos se ha conseguido dar funcionalidad a la notificación requerida e incluso se ha desarrollado otra notificación no esperada.

Por todo ello, la planificación inicial a pesar de haberse complicado los objetivos requeridos por problemas externos y no contemplados, algunos puntos representados en el diagrama cronológico no se han cumplido en los plazos esperados. Esta experiencia nos ha aportado una visión realista de un proyecto de mayor envergadura a los realizados, también nos aportado un punto de vista más cauteloso en las planificaciones de tiempo. En general nos ha sido una experiencia gratificante y nos ha contribuido conocimientos muy útiles cara a nuestro fututo laboral.

12 Monetización

He realizado un estudio de mercado a nivel de monetización sobre la aplicación y ha concluido en los siguientes resultados.

Actualmente tanto la aplicación como el servidor se encuentran funcionales y operativos, con la mayoría de funcionalidades disponibles para el usuario. Esto implica que la aplicación, a falta de ciertos retoques y una estrategia de marketing y promoción, está disponible para la venta al público.

Se han estudiado los diversos sistemas de monetización existentes y para este estilo de aplicación lo más rentable son los sistemas de publicidad no intrusiva (pequeños banners a pie de página o cabecera) e intrusiva (banners a pantalla completa), así como versión Lite con menor funcionalidad y versión completa de pago de la aplicación.

Dado el caso particular de nuestro sistema, considerando como necesaria toda la funcionalidad existente en la aplicación, se consideraría más óptimo establecer una única versión con publicidad intrusiva y no intrusiva de coste gratuito, y una versión completa sin publicidad a un precio simbólico.

13 Ideas de ampliación

Como ideas de ampliación por parte del grupo para una versión más profesional y completa que la actual se ha pensado las siguientes funcionalidades. A continuación se detallaran cada una de ellas.

- Optimizar el tráfico de datos con la creación de un sistema inteligente de sincronización de tareas consistente en tener una tabla de relación con la última inserción, modificación o borrado de una tarea, en vez de eliminar y descargar las tareas al sincronizar.
- Agregar una gestión más completa del perfil del usuario, incluido guardar sus preferencias en el servidor.
- Desarrollo de un sistema de sincronización push-up para descarga de tareas al dispositivo tras sincronizarse con otros gestores.
- Diseñar una versión offline de mapas para reducir el consumo de tráfico 3G y batería.
- Implementar una versión móvil o aplicación en otras plataformas para cubrir un mayor rango de clientes.
- Realizar la sincronización con gestores adicionales (Evernote,…).

14 Bibliografía

Amaro, J. E. (2012). *El gran libro de programación avanzada con Android.* Marcombo.

Gómez, S. (2012). *sgoliver.net.* Obtenido de Mapas en Android (Google Maps Android API v2): http://www.sgoliver.net/blog/?p=3244

Google. (2013). *Google Developers.* Obtenido de API de Google Maps para Android: https://developers.google.com/maps/documentation/android/

Google. (2014). *Google Developers.* Obtenido de La API de Google Places: https://developers.google.com/places/training/

Google. (s.f.). *Android Developers.* Obtenido de Location Strategies: http://developer.android.com/intl/es/guide/topics/location/strategies.html

Tomás, J. (2013). *El gran libro de Android.* Marcombo.

V.A. (s.f.). *Wikipedia*. Obtenido de http://en.wikipedia.org/

V.A. (s.f.). *SQLlite*. Obtenido de http://www.sqlite.org/