UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarollo de un módulo de comunicaciones entre una ciudad y sus ciudadanos en el ámbito de la Internet de las Cosas. Aplicación práctica al proyecto de ecoMobility

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Jaime Madrigal García

Tutor: Joan Fons Cors
Vicente Pelechano

Curso 2017-2018

Resum

Avui en dia existeix una creixent demanda per part de la societat, relacionada amb la salut ambiental. Ja que ha crescut la preocupació per l'aire que respirem, a causa de l'aument dels nivells de contaminació en les ciutats. Aquest fet ha fomentat la implantació de solucions tecnològiques com les ciutats intel·ligents, que poden ajudar a gestionar aquest problema, per part d'empreses i administracions públiques. Per tant, ha aparegut la necessitat de desenvolupar un mòdul de comunicació entre la ciutat intel·ligent i els seus ciutadans. Aquest mòdul ha d'informar en temps real de la situació ambiental de la zona on es troba el seu usuari.

Aquest treball de fi de grau recerca com objectiu de satisfer aquesta necessitat i mostra l'análisis, disseny i implementació d'una aplicació Android. Aquesta ha d'informar a l'usuari de la situació ambiental de la suea zona. Aquesta solució s'ajuda de la informació presa de serveis y sensors de la ciutat intel·ligent per generar notificacions , mapes en temps real amb diferent informació i missatges de l'àmbit municipal.

Finalment, s'ha creat una aplicació funcional amb Android, que ha sigut desenvolupada tant per funcionar en dispositius mòbils intel·ligents com en automòbils intel·ligents. Aquesta aplicació pot informar a l'usuario en tot moment de l'estat ambiental on està situat, gràcies a la informació rebuda pels diferents serveis i sensors de la ciutat intel·ligent en la que es trobe.

Paraules clau: Internet de les coses, ecoMobility, desenvolupament android, ciutat intel·ligent

Resumen

Hoy en día existe una creciente demanda por parte de la sociedad relacionada con la salud ambiental. Ya que ha crecido la preocupación por el aire que respiramos, a causa del aumento de los niveles de contaminación en las ciudades. Este hecho ha fomentado la implantación de soluciones tecnológicas como las ciudades inteligentes, que pueden ayudar a gestionar este problema, por parte de empresas y administraciones públicas. A causa de esto, ha aparecido la necesidad de desarrollar un módulo de comunicación entre la ciudad inteligente y sus ciudadanos. Este módulo debe informar en tiempo real de la situación ambiental de la zona en la cual se sitúe su usuario.

Este trabajo final de grado busca como objetivo satisfacer esta necesidad y muestra el análisis, diseño e implementación de una aplicación Android. Esta deberá informar al usuario de la situación ambiental de su zona. Esta solución se ayuda de la información recibida de servicios y de los sensores de la ciudad inteligente para generar notificaciones, mapas en tiempo real con diferente información y avisos de ámbito municipal.

Finalmente, se ha construido una aplicación funcional en Android que ha sido diseñada tanto para funcionar en dispositivos móviles inteligentes como en automóviles inteligentes. Esta aplicación es capaz de informar al usuario en todo momento del estado ambiental donde esta situado, gracias a la información recibida por los diferentes servicios y sensores de la ciudad inteligente en la que se encuentre.

Palabras clave: Internet de las cosas, ecoMobility, Desarrollo android, Ciudad Inteligente

Abstract

Today exist a growing demand from society related to environment health. Since the concern for the air we breathe has grown, because of increased levels of pollution in cities. This fact has promote the implementation of technological solutions such as smart cities, that can help manage this problem, by companies and public administrations.Because of this, the need to develop a communication module between the intelligent city and its citizens has appeared. This module must inform in real time of the environmental situation of the area in which the user is located.

This final degree project aims to meet this need and shows the analysis, design and implementation of an Android application. This Android application should inform the user of the environmental situation in their area. This solution is based on information received from services and smart city sensors to generate notifications, real-time maps with different information and warnings from locality.

Finally, a functional application has been built on Android that has been designed to work on both smart mobile devices and smart cars.This application is able to inform the user at all times of the environmental state where it is located, thanks to the information received by the different services and sensors of the smart city in which it is located.

Key words: Internet of things, ecoMoility, android developtment, smart cities

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En este primer capítulo se va a exponer la motivación que ha llevado a la realización del proyecto, los objetivos que se buscan conseguir y el plan de trabajo que se ha realizado.

1.1 Motivación

La preocupación por la movilidad sostenible ha ido creciendo en los entornos urbanos a raíz de los inconvenientes generados por el modelo de transporte urbano actual, basado en su gran mayoría en vehículos de combustión, generando que los niveles emitidos de gases contaminantes sean insostenibles y puedan llegar a afectar a la salud pública.

Según la OMS[1] , 1 de cada 8 muertes en el mundo ha sido a causa de la contaminación atmosférica[1]. A partir de este estudio, diferentes ciudades de todo el mundo han ido mejorando sus políticas sostenibles e invirtiendo en transformar sus ciudades en ciudades inteligentes intentando mejorar la calidad del aire que respiran sus ciudadanos. Ciudades inteligentes como Zurich, Helsinki y Munich, son consideradas según la IESE[2] como las más sostenibles[2].

Ser una ciudad inteligente implica no solo analizar y responder a los datos de medio ambiente sino que engloba indicadores como: capital humano, cohesión social, gestión pública, gobierno, movilidad y transporte, planificación urbana, proyección internacional y tecnología.[3]

Debido a este hecho ciudades que apuestan por ser o son ciudades inteligentes han ido implementando lo que se conoce como el internet de las cosas. Concepto que engloba la interconexión digital a través de internet de diferentes dispositivos o sensores.[4] Las ciudades inteligentes se valdrían de estos sensores o dispositivos repartidos por la ciudad y conocer así en todo momento el estado global de la ciudad en ámbitos como la contaminación, el tráfico, temperatura, accidentes, etc.

A causa de estos avances en infraestructura urbana, es necesario hacer a los ciudadanos sabedores de lo que sucede a su alrededor en tiempo real mediante un canal de información constante y fiable. Un canal de información que les suministre diferentes avisos según niveles de gravedad o ámbitos relacionados con la ciudad(contaminación, tráfico, etc). Ya que de esta forma se puede conseguir una mayor conciencia de la población sobre el problema medio ambiental. Promover una movilidad sostenible y la sostenibilidad de la ciudad.

[1] OMS:Organización Mundial de la Salud.
[2] IESE:Business School Universidad de Navarra

Este proyecto nace para satisfacer la necesidad de un canal de comunicación constante y fiable con los ciudadanos.

Por lo tanto como alumno este proyecto es una gran oportunidad, ya que permite demostrar los conocimientos adquiridos y por otro lado trabajar en un proyecto que forma parte de uno más grande,(ecoMobility), y que promueve una movilidad sostenible y un aprovechamiento de los recursos de la ciudad de forma eficiente y limpia. Usando para ello las infinitas posibilidades que ofrecen las ciudades inteligentes y el internet de las cosas. Por otro lado y no menos importante el poder programar en Java uno de los lenguajes de programación más usados del mundo tanto a nivel académico como profesional y el realizar una aplicación para Android, una de las plataformas más usadas del mundo es una gran motivación, ya que aumenta el número de aptitudes de cara a aumentar posibilidades laborales.

1.2 Problemática

Debido al crecimiento de la conciencia de la sociedad sobre movilidad sostenible y la implantación de la ciudad inteligente por parte de las administraciones públicas. Solo en España existen 81 ciudades inteligentes que suponen el total del 40 % de la población española ,según datos de la Red Española de Ciudades Inteligentes[5].

En base a estos datos y a causa de la existencia de la ley 27/2006, del 18 de julio, por la que se regulan los derechos de acceso a la información, de participación pública y de acceso a la justicia en materia de medio ambiente[6] , que permite a cualquier persona solicitar información sobre el estado y la evolución del medio ambiente, se genera la necesidad de desarrollar un módulo de comunicaciones entre una ciudad y sus ciudadanos. Este módulo servirá como herramienta difusora de la información medioambiental.

Para poder crear el desarollo de un módulo de comunicaciones entre una ciudad y sus ciudadanos, como se explicará más adelante, se necesitará desarrollar una aplicación con diferentes mecanismos implementados que pueda recibir información de la ciudad y de servicios relacionados con la misma. Además ha de ser capaz de interpretarla y transformarla para el usuario final. Utilizando para ello mecanismos de notificaciones con diferentes grados de aviso. Por otro lado se necesitará el desarrollo de una interfaz pensada en la usabilidad y en la experiencia de usuario.

1.3 Objetivos

El objetivo general del proyecto es la creación de un módulo de comunicación capaz de conectarse con una ciudad inteligente.

Objetivos específicos en la creación de un módulo de comunicación:

- Ser capaz de conectarse a partir de su ubicación a su ciudad inteligente.
- Que pueda conectarse a los diferentes sensores y dispositivos que conforman el internet de las cosas de la ciudad inteligente a través del estándar de comunicación MQTT[3].
- Que pueda conectarse a servicios API REST[7], relacionados con información de la ciudad inteligente.
- Tiene que transmitir la información recibida mediante notificaciones y alertas al usuario.

[3]MQTT:Message Queuing Telemetry Transport es un protocolo de comunicación ligero pensado para funcionar en el internet de las cosas

- Tiene que ser usable y amigable para el usuario final.

1.4 Impacto esperado

Tras el desarrollo del módulo de notificaciones entre una ciudad y sus ciudadanos en el ámbito de la Internet de las Cosas, el usuario final dispondrá:

- Información en tiempo real del estado ambiental próximas a su ubicación actual (contaminación y tiempo).
- Información de la situación del tráfico en la zona.
- Poder visualizar en el módulo un mapa integrado con la información de polución y tráfico representada en tiempo real.
- Poder recibir alertas de entradas en zonas con polución elevada.
- Poder recibir notificaciones directas de la administración de la ciudad por la que estemos pasando.

1.5 Metodología

La metodología empleada para el desarrollo del proyecto ha sido un conjunto de diferentes formas de trabajo, ya que en un principio, se enfocó con metodología en cascada pero también se ha hecho uso de metodologías como la iterativa/incremental[8] y la de construcción de prototipos. Esto se decidió así dado que se necesitaba una mayor flexibilidad durante el diseño y desarrollo de la solución. Además de esta manera se aportan mejoras que solo pueden verse en estos procesos y así conseguir un resultado más útil para el usuario objetivo, enfocándonos en la calidad de la solución. Estas metodologías se aplican a lo largo del trabajo y dependiendo de la sección o parte del mismo se han aplicado unas u otras o en conjunción.

El desglose de la metodología en el proyecto ha sido el siguiente:

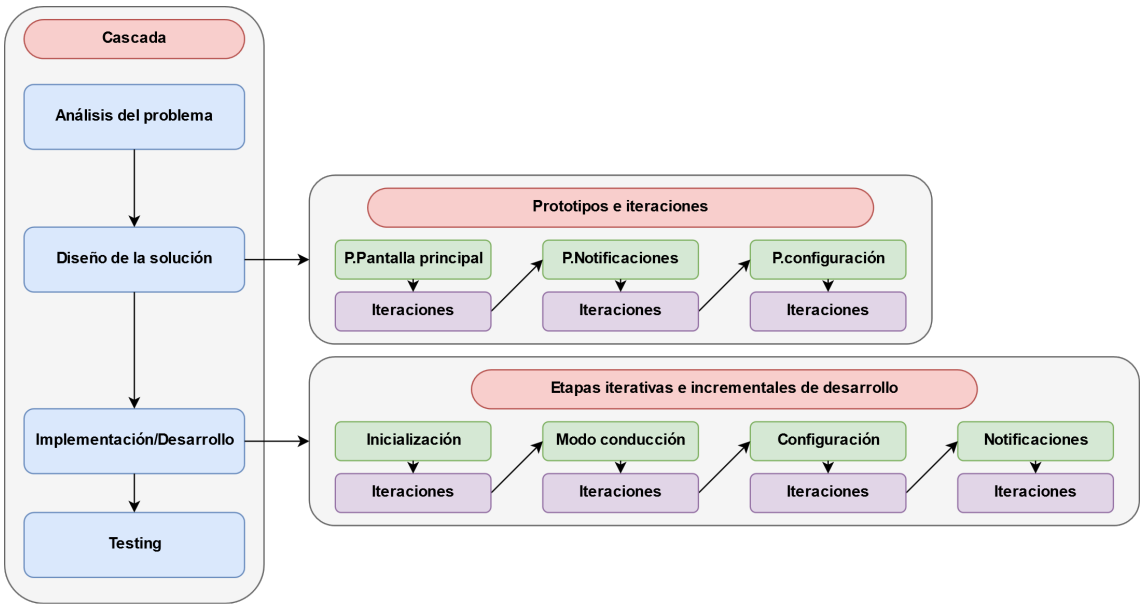

Figura 1.1: Metodología empleada en el proyecto.

1.6 Plan de trabajo

Según lo mencionado en la metodología se ha desarrollado el plan de trabajo para realizar pequeños bloques de trabajo secuencialmente a lo largo del tiempo.

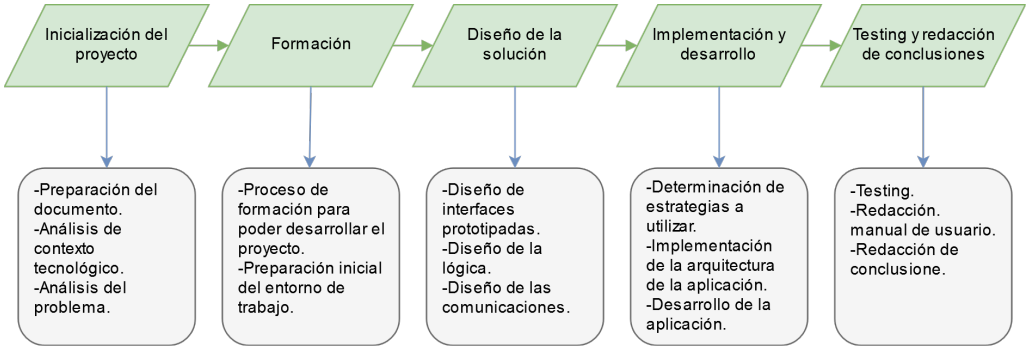

Figura 1.2: Plan de trabajo del proyecto.

Cada uno de los bloques se planificó para que durara alrededor de una semana a excepción del bloque de formación e implementación los cuales se planificó para que tuvieran un tiempo determinado de dos semanas con flexibilidad de ampliar.

1.7 Estructura de la memoria

La memoria está estructurada por nueve capítulos y una bibliografía que explican el proceso de desarrollo del módulo de comunicaciones. En el capítulo uno se muestra la introducción al proyecto abarcando aspectos como la motivación para su realización hasta como se plantea el proyecto. El segundo capítulo explicara la situación actual de las soluciones similares a la aplicación a desarrollar exponiendo y criticando su funcionalidad. El capítulo tres explica el contexto actual tecnológico y mostrará todas las tecnologías usadas para desarrollar este proyecto. En el capítulo cuatro se analiza en profundidad el problema en profundidad y se pondrá en contexto la aplicación a desarrollar. En el capítulo cinco se diseñará la aplicación explicando su estructura y cada una de sus partes. El capítulo seis es la implementación donde se explicara el como se ha realizado el desarrollo. El capítulo siete contendrá la explicación de diferentes pruebas que se le han realizado a la aplicación. En el capítulo ocho se realiza un manual de uso de la aplicación desarrollada. Por último el capítulo nueve es la conclusión donde se expondrán diferentes conclusiones aspectos de mejora y relaciones de este proyecto con los conocimientos cursados a lo largo de la carrera.

CAPÍTULO 2
Estado del arte

En este capítulo realizaremos la búsqueda y estudio de las diferentes aplicaciones que se pueden encontrar hoy en día relacionadas con la solución que se quiere desarrollar y determinando las novedades o aportaciones que tiene el proyecto al sector.

2.1 Estudio estratégico

2.1.1. Plume Air Report

Plume Air Report[1] es una aplicación disponible para Android e IOs. Proporciona una información detallada de la polución de diferentes zonas gracias a sensores públicos de las ciudades y si estas no los tienen, esta aplicación se encarga a raíz de los datos de hacer un pronóstico de que situación debería tener el aire donde el usuario se localiza.

Características:

- Datos de polución globales.
- Datos almacenados para consultar la evolución temporal de la polución.
- Datos de todos los contaminantes del ambiente.
- Comparar poluciones de varias localizaciones.
- Diferentes grados de alerta

Figura 2.1: Pantallas de la aplicación Plume Air Report.

[1]Página de la aplicación: plumelabs.com

2.1.2. Air Quality Index BreezoMeter

Air Quality Index BreezoMeter[2] aplicación móvil disponible en Android e IOs capaz de informar a su usuario de la calidad del aire de su posición actual y en tiempo real.

Sus características son:

- Mapas de calidad del aire en tiempo real de la zona.
- Recomendaciones para la salud.
- Notificaciones sobre el cambio de la calidad del aire en diferentes ubicaciones que se marquen como favoritas.

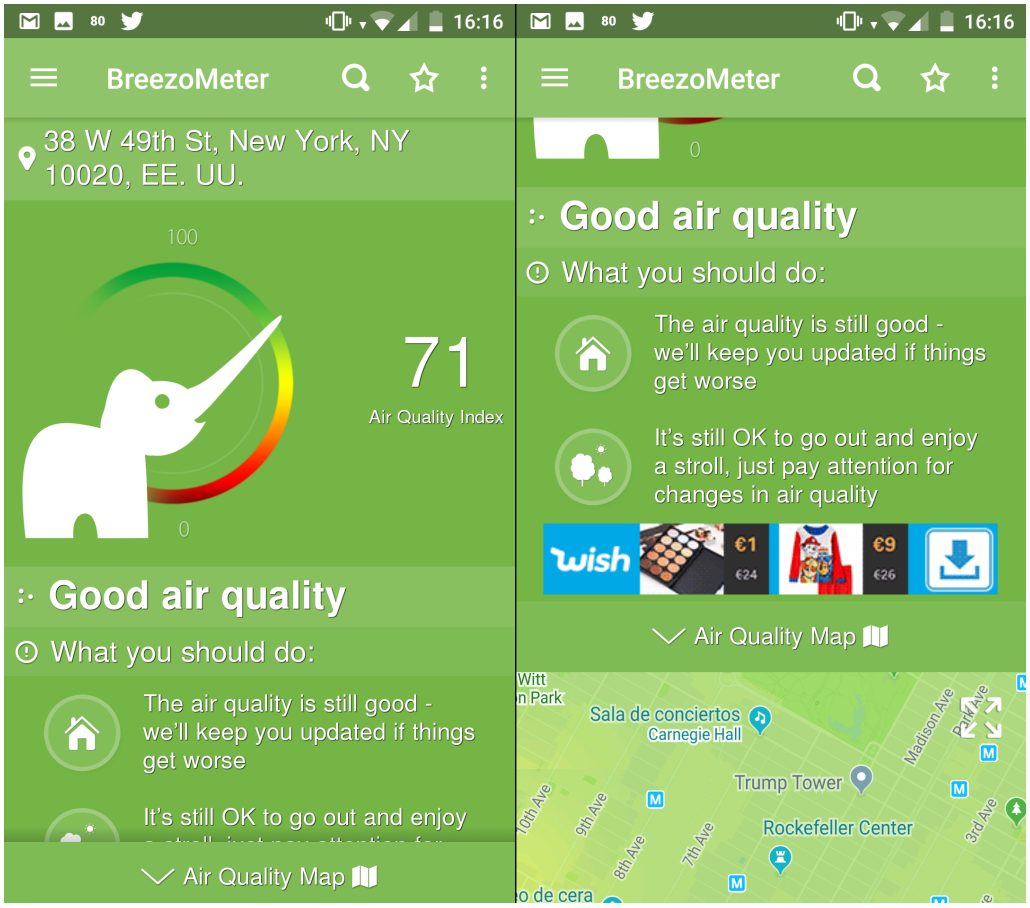

Figura 2.2: Pantallas de la aplicación Air Quality Index BreezoMeter.

[2]Página de la aplicación: breezometer.com

2.1.3. AirVisual

AirVisual[3] aplicación compatible con Android e IOs tiene una gran cantidad de datos de la polución de las ciudades, ya que alberga datos de polución de más de 10.000 ciudades de 100 países diferentes. Por otro lado esos datos los va recopilado en tiempo real y también es capaz de dar un pronóstico de 72 horas desde el momento actual. Gracias a la red de estaciones propias que posee.

Sus características son:

- Histórico de poluciones.
- Polución en tiempo real.
- Pronósticos de polución de 72 horas.
- Información meteorológica.
- Información de la polución para grupos sensibles de personas.

Figura 2.3: Pantallas de la aplicación AirVisual.

2.2 Crítica al estado del arte

Los anteriores programas presentados tienen en común el dar acceso a sus usuarios el estado de la contaminación de sus zonas donde se sitúan y cada uno de ellos aporta valores añadidos a esta información que genera la diferenciación entre unas y otras aplicaciones. Por ejemplo, Plume se centra en tener una interfaz sencilla y bonita , además de desglosar los contaminantes, por otro lado BreezoMeter sugiere recomendaciones para la salud de sus usuarios y por ultimo AirVisual hace de su aplicación una mezcla entre estación meteorológica y de contaminación con pronósticos. Todas estas aplicaciones pueden aparecer muy completas pero sin embargo no hacen uso de recursos que están apareciendo hoy en día con la implementación de las ciudades inteligentes, además de que muchos de sus datos pueden ser imprecisos por la cantidad de logistica de que dispongan. Por otro lado ninguna de estas aplicaciones dispone de medidas de tráfico de las ciudades o localidades. Algo extraño teniendo en cuenta que estos datos ayudaran a

[3]Página de la aplicación: www.airvisual.com

entender el incremento de la polución en ciertas horas y mediante la conciencia de sus usuarios se podría mejorar este aspecto medio ambiental.

2.3 Propuesta

En el siguiente trabajo final de grado se propone crear un módulo de comunicaciones entre el usuario y las ciudades inteligentes. Esta solución será capaz de incorporar características como los programas anteriores mencionados (Calidad del aire, tiempo, notificaciones sobre cambios). Pero además implementar ainformación sobre el tráfico y aplicando como novedad el uso de tecnologías relacionadas con el internet de las cosas y el contexto de la ciudad inteligente.

CAPÍTULO 3
Contexto tecnológico

En este capítulo, se hablará de los diferentes apartados tecnológicos elegidos y usados para desarrollar el trabajo final de grado: Entornos, librerías, componentes, lenguajes, etc.

3.1 Plataforma objetivo

El proyecto final de grado tiene como objetivo la realización de un módulo de comunicaciones entre la ciudad y las personas situadas en su término municipal. Para conseguir ese objetivo es necesario desarrollar el módulo en una de las plataformas, en el ámbito de los sistemas operativos móviles, más usadas del mundo (Android). Aunque es cierto que existen otras alternativas como IOS, el principal motivo para decantarse por Android radica en que, por el momento, es el sistema operativo más usado del mundo, en lo que a dispositivos móviles se refiere. Además en ámbito nacional España es el 5 mercado europeo donde más móviles inteligentes con android se encuentran[9].

Android[1] es un sistema operativo basado en el núcleo Linux, concebido en un principio como el sistema operativo para móviles inteligentes, aunque hoy en día está presente en dispositivos como tabletas, relojes inteligentes, televisores y centros multimedia de automóviles.

3.2 Lenguajes

La aplicación ha sido desarrollada en:

- **Java:**[2] Lenguaje de programación orientado a objetos, concurrente y que se puede usar para multiples propositos. Además de ser uno de los lenguajes de programación con mayor documentación y con una de las comunidades de desarrolladores más grandes del mundo. Añadir que java es el lenguaje utilizado para la creación de aplicaciones nativas en android junto con Kotlin[11, 10].
- **XML:** Metalenguaje creado por el W3C que permite definir lenguajes de marcas y utilizado para almacenar datos de forma legible. Dentro de desarrollos para android nos permite construir las interfaces de usuario y además poder almacenar valores u opciones para nuestra aplicación.[12]
- **JSON:** Formato de texto ligero que puede ser usado para el intercambio de datos.

[1] Página de los propietarios de la técnologia: www.android.com
[2] Página de los propietarios de la técnologia: www.java.com

Por otro lado mencionar que la aplicación también podria haberse podido desarrollar en tecnologías como Kotlin[3] o encapsulados como Ionic[4].

3.3 Entorno de desarrollo

El entorno de desarrollo utilizado para la creación de la aplicación ha sido:

- **Android Studio**[5]:Es un entorno de desarrollo integrado creado por Google como reemplazo al uso de Eclipse como entorno de desarrollo integral oficial. Está creado en base al software IntelliJ IDEA de jetBrains y fue publicado de forma gratuita.

Android Studio dispone de las siguientes características para el desarrollo de aplicaciones android:

- Integración de ProGuard[6] y funciones de firma de aplicaciones.
- Estadísticas de uso de la aplicación en ejecución.
- Construcción basada en Gradle[7].
- Refactorización específica de Android y arreglos rápidos.
- Inclusión de un editor de diseño basado en arrastrar y soltar elementos para construir la interfaz.
- Herramientas para detectar problemas de rendimiento, usabilidad y compatibilidad.
- Disposición de plantillas para Android.
- Soporte para android wear y auto.
- Soporte de versión de controles mediante herramientas como github[8].

3.4 Librerías

Para el correcto funcionamiento y operatividad de la aplicación, esta necesita del uso de librerías de terceros. Estas librerías aumentan la funcionalidad de la aplicación y las posibilidades de la misma de cara a su desarrollo.

Las librerías usadas en el desarrollo han sido:

- **Eclipse paho:** Librería open source que permite que la aplicación desarrollada se transforme en un cliente capaz de conectarse mediante el protocolo MQTT a los sensores del Internet de las Cosas de la ciudad inteligente.

[3]Página de los propietarios de la técnologia: https://kotlinlang.org/
[4]Página de los propietarios de la técnologia: https://ionicframework.com/
[5]Página de los propietarios de la técnologia: https://developer.android.com/studio/
[6]ProGuard: Técnologia que detecta y quita clases, campos, métodos y atributos sin usar en la aplicación ya empaquetada.
[7]Gradle: Herramienta de automatización de la construcción del código de la aplicación.
[8]Página del control de versión github: github.com/

- **appcompat v7:** Añade compatibilidad a la aplicación desarrollada para el patrón de diseño de la interfaz de usuario relacionado con barra de acciones, además se incluye compatibilidad con las implementaciones de interfaz de usuario con material design.
- **constraint-layout:** Librería que nos permite crear interfaces más grande sy complejas en la aplicación creada.
- **Design:** Nos permite la adición de componentes y patrones de material design en la aplicación desarrollada.
- **support-v4:** Proporciona compatibilidad con varias APIs del framework de la aplicación realizada.
- **play-services:** Librería que ayuda a la aplicación a disponer de los mapas de google maps para poder trabajar con ellos.

3.5 Componentes

En este punto, pasaremos a enumerar y describir los diferentes componentes que componen la aplicación desarrollada en Android.

- **Actividades:** Componente de la aplicación que contiene una pantalla con la que el usuario objetivo puede interactuar.Cada actividad tiene asignada una ventana en la que se puede diseñar la interfaz del usuario.[13]
- **Fragmentos:** Es el componente que representa un comportamiento o una parte de la interfaz de usuario de un activity.[14]
- **Notificaciones:** Son los diferentes elementos de los que dispone el sistema operativo android y que nos permite interactuar con el usuario en base a eventos generados por el exterior o dispositivo.

CAPÍTULO 4
Análisis del problema

4.1 Introducción

En este capítulo se analizará la problemática de la solución que se quiere desarrollar. Para ello este capítulo se basa en distribuir el análisis en cinco apartados. Se comenzará por determinar el contexto de la aplicación, que nos ayudará a entender su entorno y los datos que circulan por él. Seguidamente la sección de caso de estudio nos ayudará a determinar qué es lo que se quiere desarrollar. A partir de lo anterior se procederá a realizar una descripción general de la aplicación a desarrollar con sus funcionalidades, las características de sus usuarios y las restricciones. Analizado lo anterior se procederá a determinar los requisitos de la aplicación, exponer su diagrama de clases y mostrar los casos de uso.

4.2 Contexto de la aplicación

La aplicación se desenvuelve dentro del marco del macroproyecto ecoMobility. Este macroproyecto busca a partir de las ciudades inteligentes y el uso de las tecnologías basadas en el internet de las cosas desarrollar soluciones que ayuden a la sostenibilidad de las ciudades. Según este contexto la aplicación está pensada para desenvolverse dentro de ciudades inteligentes que dispongan la información tal y como el proyecto ecoMobility lo plantee.

Expuesto este hecho se procede a mostrar mediante la siguiente figura el ecosistema por donde la aplicación de comunicará.

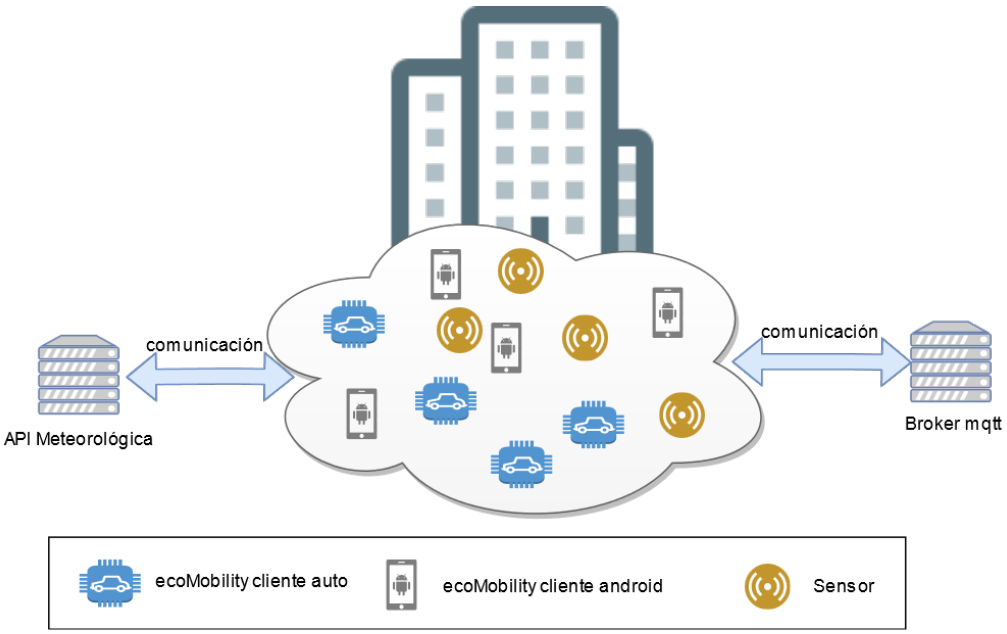

Figura 4.1: Ecosistema del proyecto.

Observando la imágen anterior se localizan diferentes elementos que están conectados entre ellos. Estos elementos son:

- **Clientes con la aplicación:** Móviles y automóviles con la aplicación ecoMobility instalada.
- **Sensores:** Son todos los sensores de que dispone la ciudad para conocer su situación.
- **Servidores Brokkers MQTT:** Servidores encargados de gestionar las comunicaciones con los sensores de la ciudad inteligente.
- **Gestor de notificaciones:** Servicio de la administración que informa de las diferentes alertas o noticias relacionadas con su zona.
- **Servidor de servicio del tiempo:** servidor encargado de almacenar y suministrar mediante su api la información meteorológica.

A partir de estos elementos se concluye que la aplicación estará en constante comunicación con dos de los elementos los servidores MQTT Brokkers y el servicio meteorológico.

4.3 Caso de estudio

Se quiere desarrollar un programa capaz de conectarse con la ciudad inteligente donde se encuentre su usuario y que además este programa facilite información relacionada con el medio ambiente y la calidad del aire. Esta información se reparte por un lado en datos legibles y por otro en información representada en un mapa que cargará la aplicación. En lo referente a los datos legibles se necesitará mostrar información de:

- Ciudad actual.
- Tiempo.
- Polución.
- Tráfico.

Esta información la provee la ciudad inteligente por sensores conectados (polución y tráfico) y por servicios (tiempo). El dato de la ciudad actual se recoge gracias al sensor del propio dispositivo de geolocalización.

En el otro lado tendremos los datos plasmados en el mapa de la aplicación, estos datos se dividen en zonas de polución y una capa activa de la información del tráfico por calles.

Se quiere que la aplicación móvil permita a su usuario disponer de un conjunto de información relacionada con la movilidad y el medio ambiente relacionadas con la zona por donde este pase. Esta aplicación aporta una visión más intuitiva y accesible de la información y pretende mejorar la conciencia sobre la movilidad sostenible.

4.4 Descripción general

4.4.1. Funciones de la aplicación

Siguiendo con la exposición del desarrollo de la aplicación, a continuación se procede a describir las funciones que un usuario de la aplicación puede realizar.

- Consultar la localidad en la que se encuentra en todo momento.
- Consultar la situación del tráfico en tiempo real.
- Consultar el estado de la contaminación en tiempo real.
- Consultar el estado del tiempo de su zona en tiempo real.
- Disponer de un mapa completo de la zona, con la información representada de la polución por zonas e integrado con la capa de situación del tráfico todo ello en tiempo real.

4.4.2. Características del usuario objetivo

El usuario que descargue e instale la aplicación podrá usar directamente la aplicación y podrá conocer de primera mano la información de la ciudad.

Una vez iniciada la aplicación el usuario obtendrá un resumen de la información del estado de la zona en la que se encuentra pudiendo navegar por el mapa y obtener una representación de la polución y tráfico de la localidad donde se encuentre.

Por otro lado el usuario podrá consultar todas las notificaciones que la administración esté enviando para informar a las personas de la zona de los diferentes sucesos que están ocurriendo en ese preciso momento.

Además los usuarios disponen de un modo conducción que activaría la función GPS[1] de la aplicación consiguiendo una navegabilidad automática del mapa mientras se esta conduciendo.

4.4.3. Restricciones

En este subapartado expondremos las limitaciones de uso que limitan la funcionalidad del proyecto.

La aplicación necesita:

- Disponer de un dispositivo inteligente o vehículo inteligente con Android.
- Conexión constante a Internet para poder comunicarse con la ciudad inteligente mediante el protocolo de mensajería MQTT o los servicios de la misma.
- Que el dispositivo donde esté instalada disponga de GPS y además habilitado.
- Que el dispositivo disponga de una versión de Android igual a 6.0 o superior.

4.5 Requisitos

En esta sección se comentará a continuación los requisitos funcionales y no funcionales de la aplicación

4.5.1. Requisitos de software

- La aplicación solo funcionara bajo sistema operativo Android.
- La aplicación solo funcionara correctamente si dispone de las siguientes dependencias:
  - android.support:appcompat-v7
  - android.support.constraint:constraint-layout
  - android.support:design
  - android.support:support-v4
  - google.android.gms:play-services
  - eclipse.paho.android.service:1.0.2

4.5.2. Requisitos hardware

- La aplicación tiene que arrancarse en dispositivos inteligentes y con pantallas táctiles donde representar la información y controlar las interacciones.

[1]GPS:Global Positioning System, sistema por el cual los sensores GPS funcionan

4.5.3. Requisitos funcionales

- La aplicación debe suministrar la información global de la situación de la zona donde el usuario se sitúe.
- La aplicación debe disponer de modo conducción.
- La aplicación debe permitir consultar los avisos de la administración.
- La aplicación debe permitir modificar si se muestran o no y con qué grado las notificaciones en su dispositivo.
- La aplicación mientras no esté en modo conducción debe permitir dejar al usuario consultar el mapa.

4.5.4. Requisitos no funcionales

- La aplicación debe poder conectarse a cualquier ciudad inteligente.
- Los servidores deberan tener implementados software mosquitto[2] para la gestión de los mensajes entre la aplicación y los sensores de la ciudad inteligente.

4.6 Casos de uso

En esta sección se describirán las tareas que se completarán por parte del usuario para conseguir realizar las diferentes funcionalidades de la aplicación. Para poder representarlo se hará uso de diagramas UML, un tipo de diagramas usados para describir procesos y que ayudará a entender los procesos de la aplicación. Los casos de uso tienen como protagonistas uno o más actores que interactúan con el sistema, en nuestro caso sólo disponemos de un actor que es el usuario. Esto es así por que la aplicación no incorpora ningún sistema de privilegios o usuarios diferenciados.

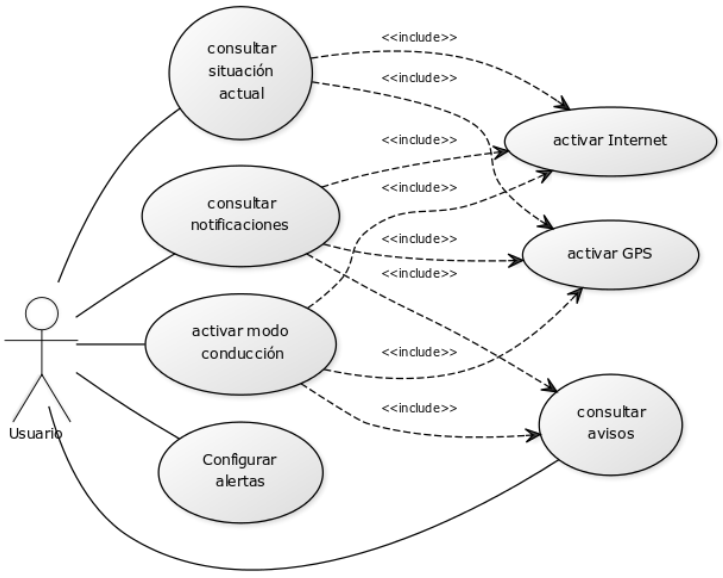

Figura 4.2: Caso de uso.

[2]Página del software Eclipse Mosquitto: mosquitto.org

4.7 Diagramas de clases

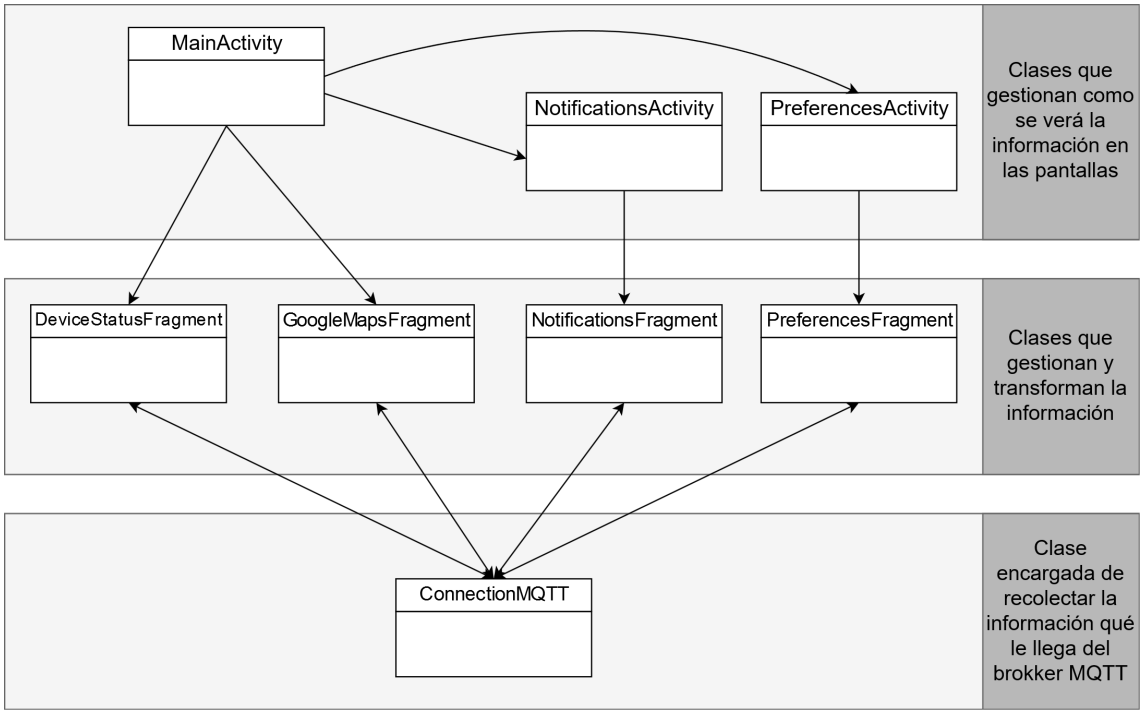

Figura 4.3: Diagrama de clases.

En el anterior diagrama de clases se distinguen tres tipos de clases.

Las que gestionan la información en las pantallas y que se encargan de cargar sus respectivos layouts, además como coordinadora de todas estas clases la principal es la clase MainActivity, estas clases especiales de Android (Actividades) se coordinarán con la lógica y la interfaz para conseguir así poder mostrar la información al usuario.

El segundo tipo de clases son los fragmentos, estas clases preparan la información para que sea gestionada por las actividades.

Y el tercer tipo de clases, pero en este caso solo una clase, es la encargada de gestionar las conexiones con el servidor MQTT.

CAPÍTULO 5
Diseño de la solución

En el capítulo de diseño de la solución se explicarán las diferentes secciones o partes del diseño de lógica, comunicaciones e interfaz.

5.1 Diseño de interfaces

En esta sección del proyecto explicaremos el porqué de las diferentes interfaces de usuario desarrolladas para la presentación de los datos obtenidos. Además se expondrá su evolución desde los primeros prototipos hasta el prototipo final de cada pantalla.

5.1.1. Prototipo Pantalla principal

Figura 5.1: Primera iteración del diseño de la pantalla principal.

En la anterior figura se muestra el primer prototipo de lo que podría haber sido la pantalla de la aplicación, pero una vez planteado esta solución de diseño, se observaron diferentes problemas de usabilidad como:

- Una mala distribución de los botones que en el caso de que el usuario estuviera conduciendo dificultad a su acceso o podría incluso provocar un accidente.
- Abuso del uso del texto para representar la información, lo cual no facilita una rápida comprensión de la información.

Por lo tanto se concluyo que esta un era una solución al diseño de la interfaz y se procedió ha realizar una segunda iteración al diseño para conseguir mejorar estos aspectos y detectar otras posibles carencias.

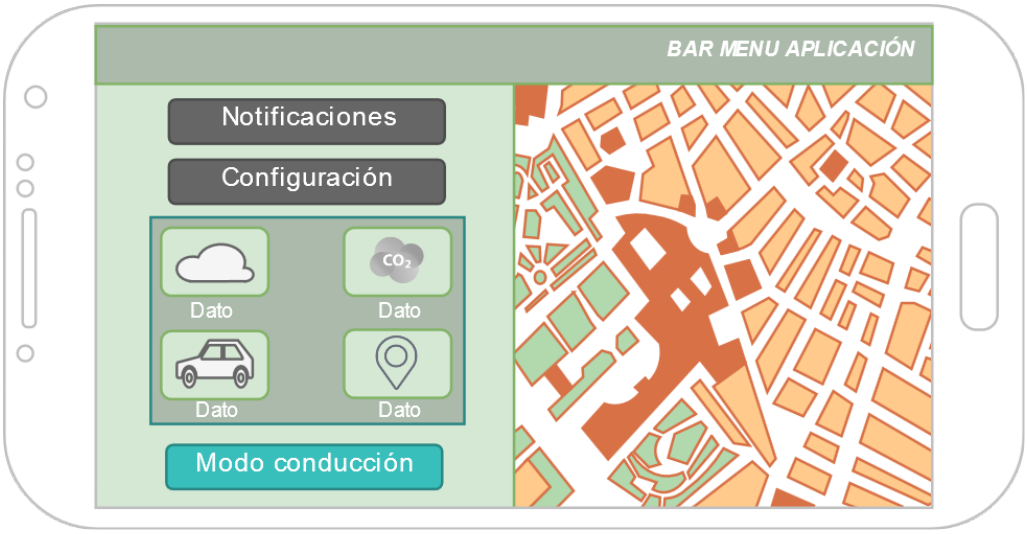

Figura 5.2: Segunda iteración del diseño de la pantalla principal.

Siguiendo el procedimiento iterativo se busca la mejora de la primera versión prototipada, en la figura anterior disponemos de un prototipo con una nueva disposición de los elementos, destacando el uso de una menor cantidad de texto y la inclusión de cuatro iconos representativos en su lugar, por otro lado se cambia la disposición del mapa para que ahora la accesibilidad de los botones sea óptima durante el ejercicio de la conducción. Sin embargo esta nueva iteración del diseño mostró una carencia importante y es la falta de visibilidad del mapa.

Por lo tanto se decidió realizar una última iteración al diseño de la interfaz.

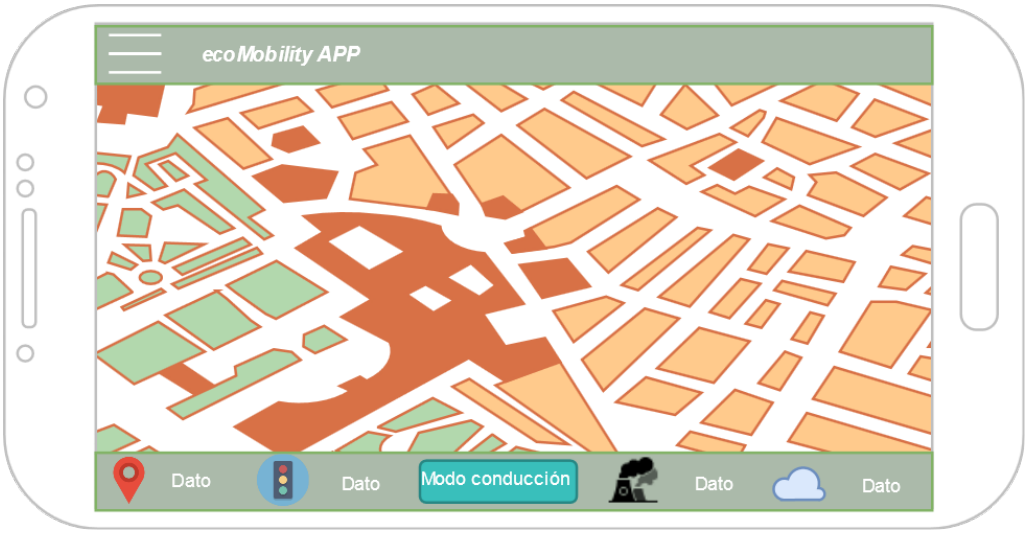

Figura 5.3: Tercera iteración del diseño de la pantalla principal y final.

Figura 5.4: Tercera iteración del diseño de la pantalla principal y final con menú oculto desplegado.

Para poder realizar esta tercera y última iteración se debía solventar el problema del espacio en pantalla, para poder solucionarlo se optó por la utilización de un menú oculto deslizante, el cual contendría la funcionalidad de los botones de notificación y configuración y además aporta a la aplicación la capacidad de ampliación de funcionalidades. Por otro lado la adición de este nuevo elemento aportaba una mejora estética de la aplicación al tener este un aspecto más simplificado y más orientado al standard de Material Design[15].

El siguiente problema a solucionar fue el desplazamiento de los iconos representativos. La decisión tomada fue de colocarlos en la zona inferior de la pantalla junto con sus datos, al hacerlo se decidió por distribuirlos horizontalmente y conseguir de esta forma una pantalla principal más simple y usable.

5.1.2. Prototipo Pantalla notificaciones

El siguiente prototipo realizado es la pantalla de información de las diferentes notificaciones que irán llegando a la aplicación. Para su desarrollo se aplicó también un sistema iterativo para optimizar su usabilidad y simplicidad.

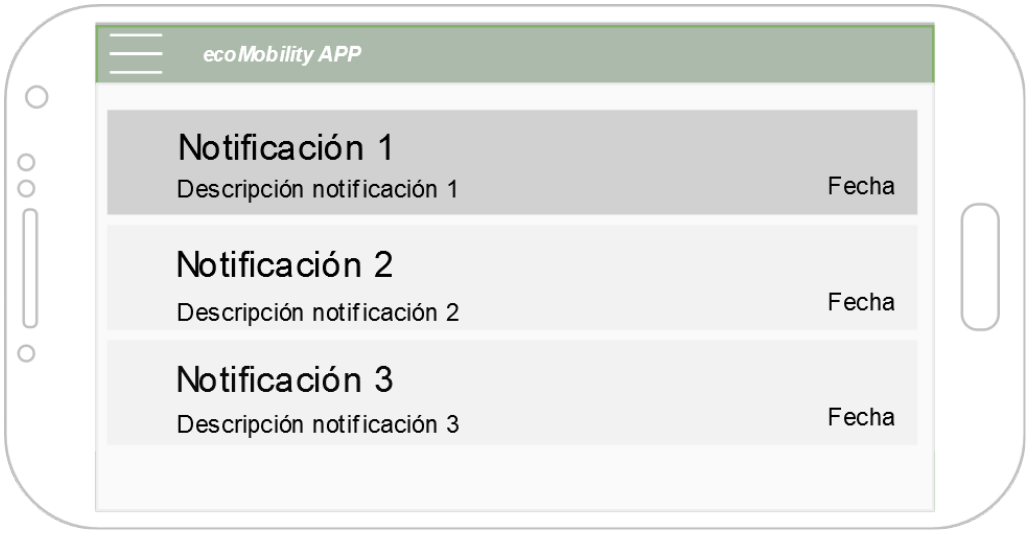

Figura 5.5: Primera iteración del diseño de la pantalla notificaciones.

Al realizar este prototipo se observó que se abusaba del uso del texto, por lo tanto se optó por una solución similar a la de la pantalla principal, incluir el uso de iconos representativos. Estos iconos a diferencia de los de la pantalla principal servirán como una mera presentación del ámbito de la notificación ayudando al usuario a interpretar el tipo de información antes de que comience a leer la misma en la descripción de la notificación.

Figura 5.6: Segunda iteración y final del diseño de la pantalla notificaciones.

Ahora con el siguiente desarrollo se ha conseguido una pantalla de notificaciones visualmente más atractiva y usable para el usuario objetivo.

5.1.3. Prototipo Pantalla configuración

Para el diseño de la pantalla de configuración se eligió usar el diseño de la pantalla estándar de android "preferences". Las pantallas que usan preferences son pantallas que adoptan el standard de material design a un esquema de configuración.Por lo tanto nuestro esquema para la aplicación ha sido el siguiente.

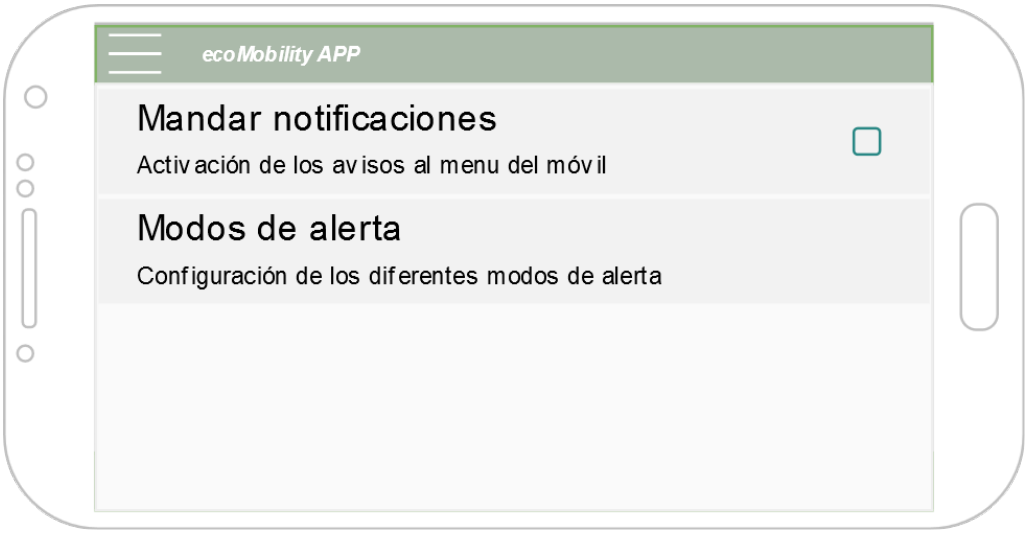

Figura 5.7: Primera iteración y final del diseño de la pantalla configuración.

5.2 Diseño de lógica

Para el diseño de la solución en el contexto de la lógica se ha dividido la aplicación en diferentes módulos con diferentes funcionalidades, dependiendo del origen de los datos y el cómo tienen que ser tratados. Estas partes son:

- **deviceStatus:** Parte encargada de gestionar el GPS del dispositivo móvil. Además se encarga de gestionar la información recibida por otras partes de la aplicación relacionadas con la geolocalización del dispositivo.
- **maps:** Encargada de cargar el mapa a partir de la información de otros módulos del programa, será capaz de construir el mapa y sus diferentes capas (polución y tráfico).
- **notifications:** Encargado de montar las notificaciones y gestionarlas.
- **preferences:** Encargado de gestionar la configuración del usuario en lo referente a las notificaciones.
- **connectionMQTT:** Módulo encargado de gestionar la suscripción del dispositivo a la ciudad inteligente y de gestionar la información que esté emitiendo la ciudad.

Descritos estos módulos y a partir del diagrama de casos de uso se procede a plantear los diferentes diagramas de secuencias que representan el funcionamiento interno de la aplicación para cada uno de los casos de uso que el usuario podrá realizar en la aplicación.

En el diagrama de secuencias siguiente se muestra las diferentes interacciones internas para resolver la necesidad de consultar la situación actual. Para ello la aplicación, a

partir de sus diferente módulos, nada más iniciarse preguntará al módulo de "deviceStatus" por su situación (coordenadas y localidad). Este le devolverá esa información y la aplicación mediante con esa información pedirá un mapa de su ubicación al módulo de "Maps" el cual devolverá inicialmente el mapa con la localización actual. Seguidamente y mientras se carga el mapa, el módulo "deviceStatus" pedirá la información al módulo "connectionsMQTT" respecto la polución y el tráfico de la zona, enviando a este las coordenadas que previamente había conseguido. Una vez hecho esto, la interfaz de la aplicación mostrará la información obtenida por los módulos.

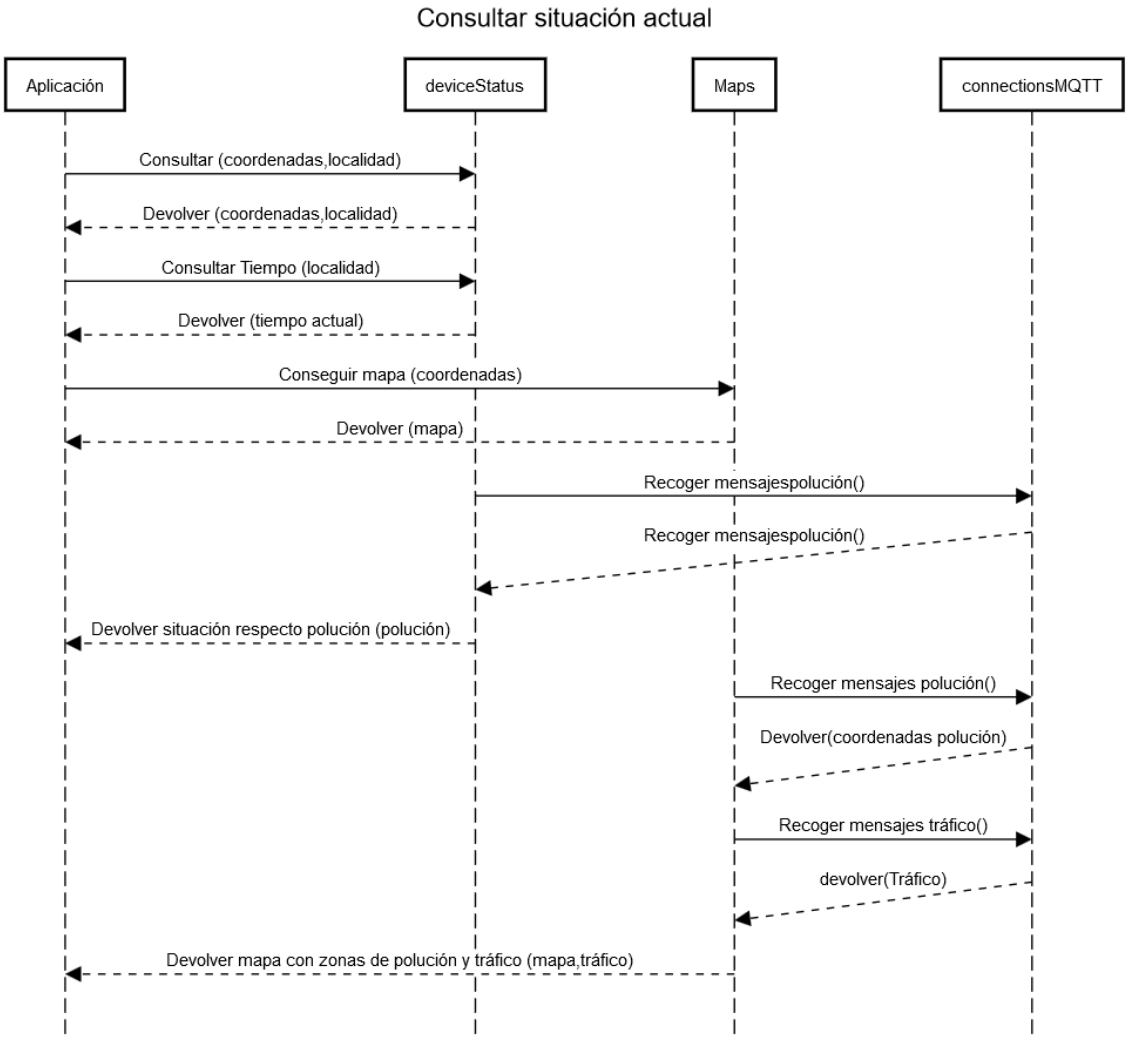

Figura 5.8: Esquema secuencial del caso: consultar situación actual.

En el siguiente diagrama se expone el diseño de cómo se solventa la gestión de las notificaciones. Para hacerlo la aplicación se vale del módulo de "Notifications", "deviceStatus" y "ConnectionsMQTT". El proceso de interacción entre ellos es: Primeramente que el módulo de notificaciones conozca el contexto de la aplicación "localidad" esto se hace recogiendo la información obtenida anteriormente por el módulo "deviceStatus". Seguidamente y por último notifications se comunica con el módulo "connectionsMQTT" para gestionar toda aquella información clasificada como notificaciones y disponerla para ser visualizada por la interfaz.

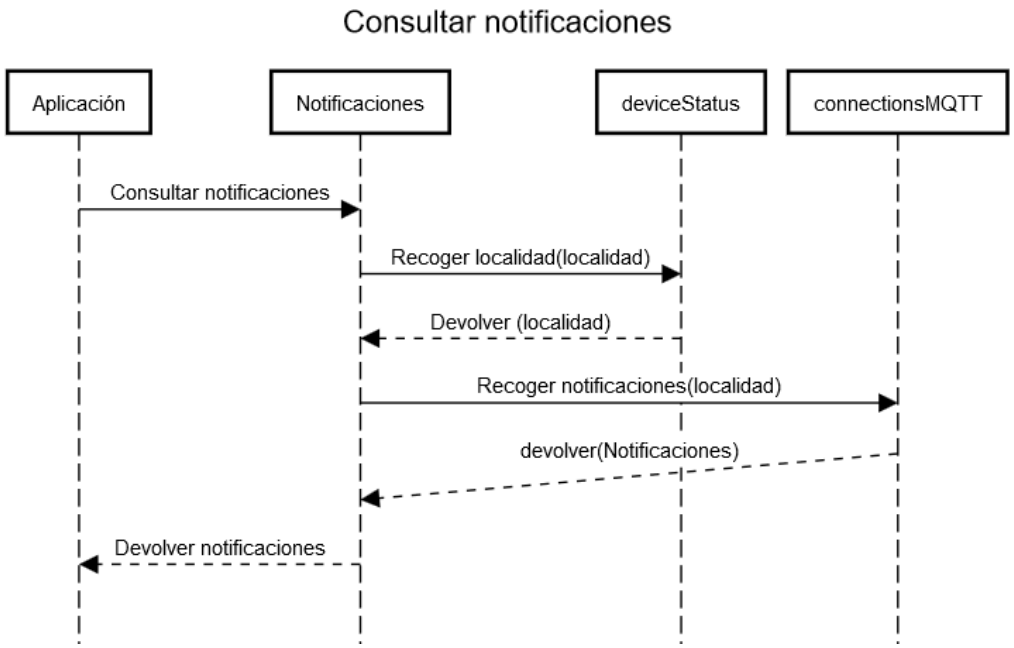

Figura 5.9: Esquema secuencial del caso:consultar notificaciones.

Para diseñar la solución del caso "cambiar modo conducción" se necesita la interacción de los módulos "Maps", "deviceStatus" y "connectionsMQTT". En el siguiente diagrama de secuencia se expone el caso:

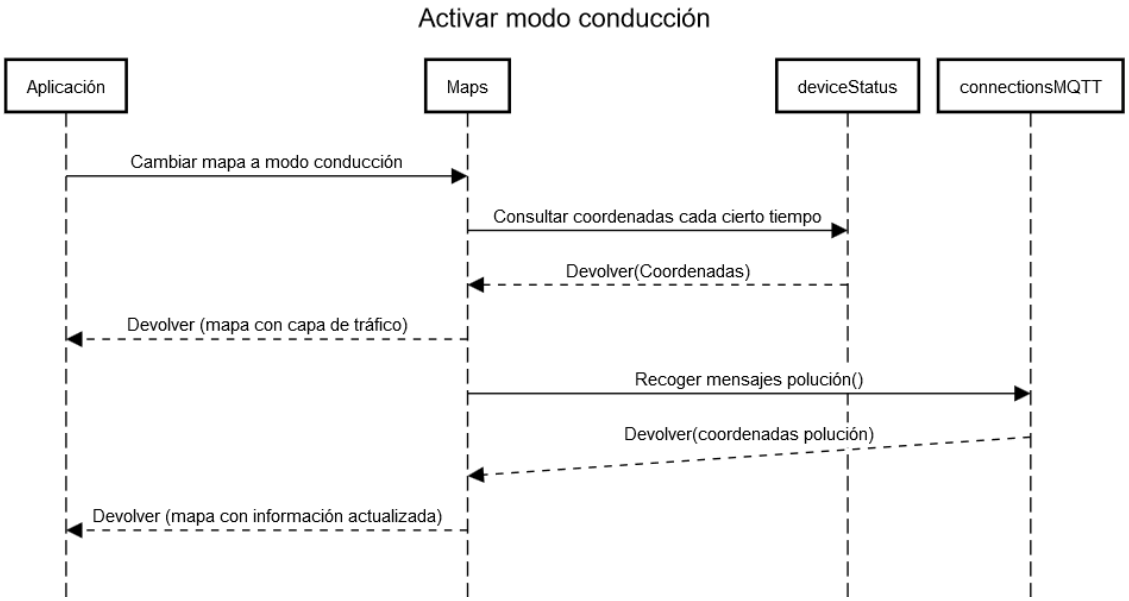

Figura 5.10: Esquema secuencial del caso:cambiar al modo conducción.

El funcionamiento consiste en que el usuario, al activar este modo, generará que la aplicación solicite al módulo "Maps" la información del mapa con la capa de tráfico y además le fije la cámara del mapa para que esta se mueva con el movimiento del dispositivo, mientras este esté conduciendo. Además el módulo del mapa necesitará, para llevar a cabo la tarea, consultar periódicamente la ubicación actual del dispositivo y su situación. Para ello se valdrá del módulo "deviceStatus" el cual le suministrará esa información y por otro lado el módulo "connectionsMQTT" que le proveerá de la información

respecto de la zona por donde pase ya que el trabajo de los módulos de “deviceStatus” y “connectionsMQTT” afectarán al mapa cada cierto tiempo.

Por último en la siguiente figura se muestra el diseño de la lógica para el caso “cambiar configuración alertas”. Este diagrama muestra cómo la comunicación es directa con el módulo de “configuración alertas” el cual se encargará de gestionar los diferentes grados de alertas y si se habilitan o no.

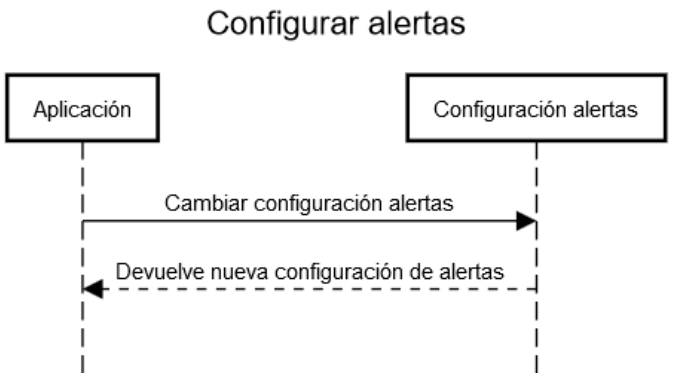

Figura 5.11: Esquema secuencial del caso:cambiar configuración de alertas.

5.3 Diseño de comunicaciones

Para poder integrar la aplicación móvil al entorno y poder conocer los datos que nos proporciona la ciudad inteligente por donde pase se necesita diseñar cómo serán las comunicaciones. En este apartado procederemos a diseñar cómo van a ser estas comunicaciones de la aplicación con la ciudad y el contexto de las mismas.

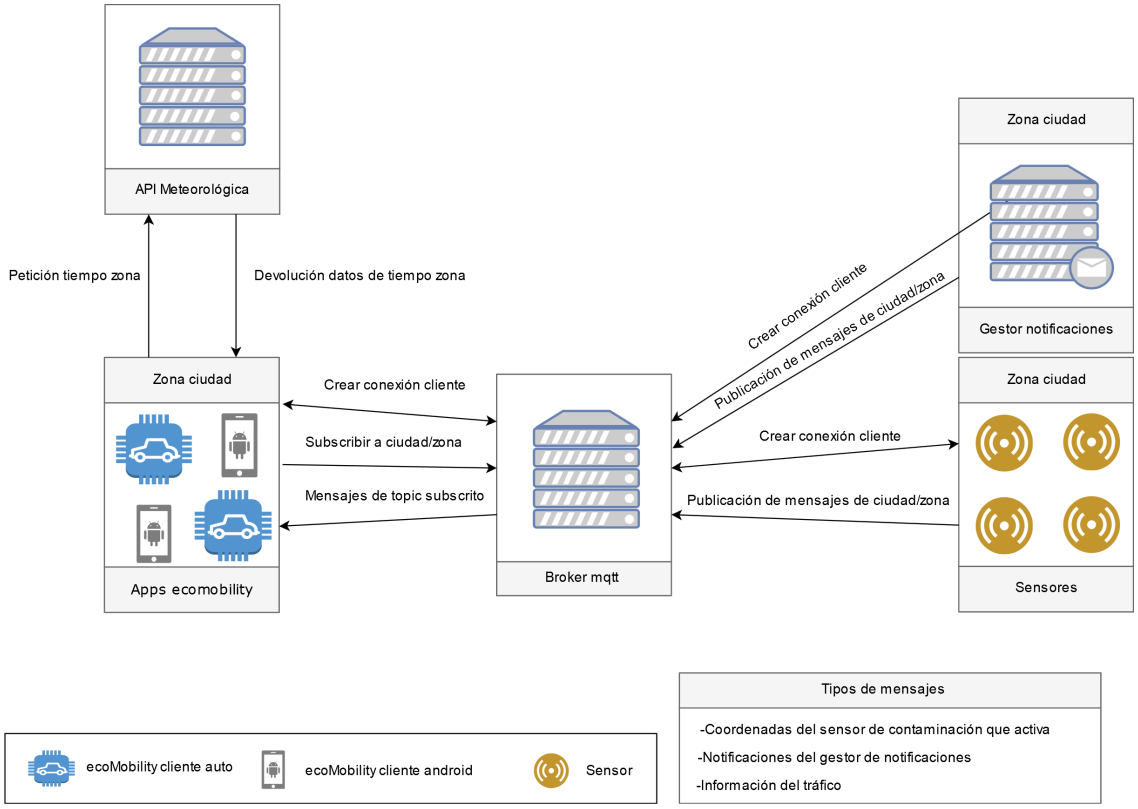

Figura 5.12: Diagrama de diseño de comunicaciones.

En la anterior figura se expone como es el contexto de todas las comunicaciones que involucran a la aplicación y además se exponen las comunicaciones del servidor brokker MQTT con los sensores y gestor de notificaciones para visualizar la interacción de la ciudad inteligente.

La aplicación dispondrá de dos flujos de información:

- **Servidor con API del tiempo:** Este servidor ofrece los datos meteorológicos relacionados con la localidad que le envíe la aplicación.
- **Servidor brokker MQTT:**Mediante este servidor la aplicación se suscribirá a diferentes topics relacionados con su zona. De esta forma recibirá información del tráfico, notificaciones y polución.

CAPÍTULO 6
Implementación

En el siguiente capítulo se procede a explicar las diferentes fases de la implementación. Explicando las estrategias utilizadas, la arquitectura implantada y el proceso de desarrollo del proyecto.

6.1 Estrategias utilizadas

Para la implementación del proyecto se ha decidido por usar una metodología de trabajo incremental.

Esta metodología ayuda a agrupar tareas en pequeñas etapas repetitivas. Cada una de estas etapas buscará cumplir los requerimientos solicitados para la aplicación, comenzando por implementar el esqueleto de la aplicación , que sea funcional y más prioritario y continuando con la adición del resto de requerimientos y funcionalidades en iteraciones pequeñas.[8]

La ventaja de un desarrollo de estas características es que el desarrollador o desarrolladores que aplican esta metodología obtienen un conocimiento del proyecto mayor en cada iteración, además de que permite separar la complejidad del proyecto gracias a la repartición de los requerimientos o funcionalidades en pequeñas etapas iterativas del desarrollo.

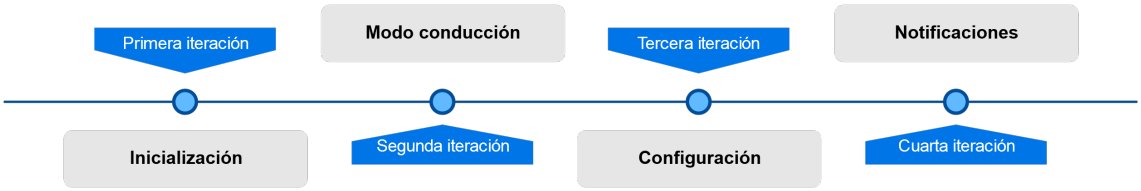

Figura 6.1: Iteraciones del proyecto.

6.2 Arquitectura de la aplicación

La aplicación se estructura usando el modelo de diseño de software basado en la arquitectura cliente-servidor o también conocido como arquitectura a dos capas.

Este modelo de diseño reparte las tareas entre dos actores principales, el servidor y el cliente. El servidor se encarga de repartir unos servicios o recursos y el cliente realiza peticiones al servidor el cual le facilita la información que necesita. Ahora es el cliente el que la gestiona y transforma la información para ser consumida por el usuario.[16]

Se elige esta arquitectura, ya que la aplicación solo tiene como funciones solicitar y recibir la información suministrada por diferentes servidores. Además ser capaz de gestionarla y transformarla para que sea entendible por su usuario. Por otro lado al no disponer disponer de una base de datos o fichero que almacene datos del usuario en la aplicación hace más claro el uso de la arquitectura de cliente/servidor para este proyecto.

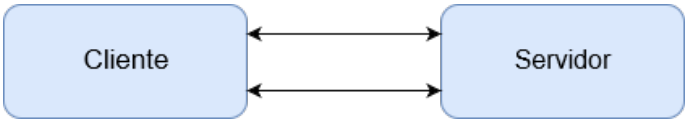

Figura 6.2: Esquema de la arquitectura.

Dado que el desarrollo del proyecto se basa en el lado del cliente se procede a dividir esta capa en dos, capa de presentación y capa de lógica.

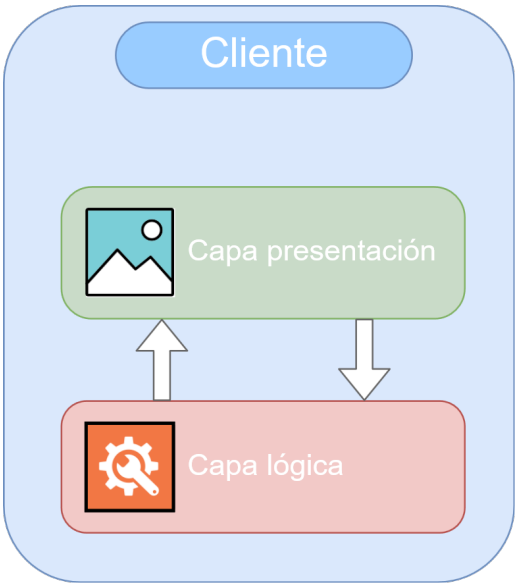

Figura 6.3: Esquema de la arquitectura del cliente.

6.3 Desarrollo de la aplicación

A partir de los dos puntos anteriores se procede a explicar el desarrollo del proyecto comenzando por explicar cada una de las etapas del desarrollo y finalizando con un resumen del conjunto de componentes desarrollados.

6.3.1. Formación

Para abordar el proyecto con solvencia se necesitó de una preparación previa sobre las diferentes tecnologías que se iban a implementar en el desarrollo, ya que estas no se han visto previamente durante la formación en la carrera. Para ello se ampliaron conocimientos de Android mediante cursos como los de la plataforma edX[1]. Mediante esta plataforma se realizó el curso de desarrollo en android impartido por la UPV[17], impartido por el Dr. Jesús Tomás Gironés.

[1]Página web de la plataforma de cursos edX: www.edx.org

Seguidamente se ampliaron los conocimientos sobre el funcionamiento de MQTT y tratamientos de datos JSON. Mediante pruebas, cursos y documentación de la red.

6.3.2. Preparación del entorno de trabajo

Para el desarrollo de la aplicación se ha utilizado un puesto de trabajo con las siguientes características:

Características	
Procesador	AMD RYZEN 7 1700, 3GHz, 8 cores
RAM	16 GB DDR4
Sistema Operativo	Windows 10

Tabla 6.1: Información del equipo usado para desarrollar el proyecto.

El entorno de pruebas ha sido un dispositivo android con las siguientes características:

Características	
Modelo	Xiaomi MI A1
Procesador	Snapdragon 625 , 2,2GHz, 8 cores
RAM	4GB
Sistema Operativo	Android 8

Tabla 6.2: Información del dispositivo de pruebas.

Como IDE de desarrollo principal se ha usado Android studio en su versión 3.1.2 y como controlador de versiones se ha utilizado Github.

Por otro lado para emular un servidor brokker MQTT se ha optado por implementar un servidor mosquitto, ejecutándose bajo una maquina virtual Ubuntu 18.04 LTS además se ha hecho uso de un Dynamic DNS Update Client (DUC)[2] para poder conectarnos desde fuera de la red local con la aplicación desarrollada.

6.3.3. Primera iteración: Inicialización

En esta etapa se implementa la funcionalidad mínima, pero que sirva de esqueleto para el resto de funcionalidades especificadas en las etapas anteriores del proyecto y que se añadirán en las siguientes iteraciones.

Para ello se decide que durante la creación del proyecto se desarrollen los siguientes aspectos:

- AndroidManifest
- MainActivity
- DeviceStatusFragment(DeviceStatus)
- GoogleMapsFragment(Maps)
- connectionMQTT
- Primera interfaz usable basada en los prototipos

[2]Página del software DUC: www.noip.com

El objetivo es crear una primera versión de la aplicación donde se pueda interactuar con el mapa y sitúe la situación a nivel de posición del cliente.

Ahora se procederá a explicar el desarrollo de cada uno de ellos:

AndroidManifest

Toda aplicación nativa de Android tiene un fichero denominado AndroidManifest.xml este fichero es un manifiesto de la aplicación. Su funcionalidad es aportar información esencial de la aplicación al sistema Android.[18] La información que puede contener es:

- Nombre del paquete Java de la aplicación que sirve como identificador de la aplicación.
- Describe los componentes de la aplicación y las condiciones para su lanzamiento.
- Identifica los procesos que alojan a los componentes.
- Declara que permisos se necesitará la aplicación para interactuar con los diferentes sensores o partes protegidas de la API de Android.
- Declaración del nivel mínimo de Android API que se necesita.
- Enumera las bibliotecas con las que tiene que estar vinculada la aplicación.

En el presente proyecto el fichero AndroidManifest.xml quedó de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    package="gps.ecomobility.com.tfg">

    <!-- acceso al GPS,COBERTURA y WIFI -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />

    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />

    <uses-feature android:name="android.hardware.location.gps" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:screenOrientation="landscape">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <meta-data
            android:name="com.google.android.geo.API_KEY"
```

```
34              android:value="KEY" />
35          <service android:name="org.eclipse.paho.android.service.MqttService" />
36      </application>
37
38  </manifest>
```

MainActivity

Para el desarrollo de esta primera versión creamos la actividad principal o MainActivity. Este componente servirá a la aplicación como invocador de otros componentes como actividades secundarias o fragmentos. Además se encarga de gestionar el funcionamiento del layaout o capa de presentación de la aplicación.

En nuestro caso el MainActivity en esta primera iteración invocará a dos fragmentos que contendrán parte de la lógica y funcionalidad de la aplicación.

- DeviceStatusFragment
- GoogleMapsFragment

Por otro lado hara la gestión de crear el cliente MQTT llamando a la clase encargada de las conexiones.

- ConnectionMQTT

Además dado que muchas de las clases que se implementan van recibiendo datos de forma constante a través de agentes externos como la ciudad inteligente o los sensores del dispositivo y esto genera que cambie el estado de la información de las clases se procedió a solventar este problema implementando el patrón observador para gestionar de una forma eficiente y fácil la información interna de la aplicación.

El patrón observador o observer es un patrón de diseño que genera una dependencia de uno a muchos generando que si un objeto cambia de estado avisa a todos los observadores.

En la aplicación se gestiona de la siguiente forma:

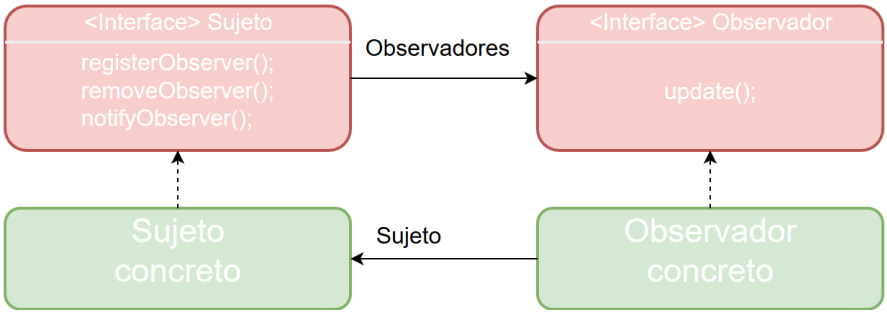

Figura 6.4: Esquema del funcionamiento del patrón observer.

DeviceStatusFragment

Este fragment o fragmento es un componente del cual la aplicación obtendrá datos como la posición del dispositivo,la localidad y comprobar si está en una zona con polución elevada.

Para ello se hará uso del sensor del dispositivo GPS y la información de localización del proveedor de internet.Para poder tener acceso como hemos especificado antes en el manifest añadimos que se necesitan los permisos para el GPS e Internet. Seguidamente este fragmento tendrá que implementar las siguientes características:

- Implementación del LocationListener: Encargado de escuchar las modificaciones de estado del objeto location.
- Adición del objeto cliente FusedLocationProviderClient: Encargado de obtener la posición del GPS, controlar la precisión del mismo y su consumo.
- Añadir el objeto LocationRequest: Para determinar las características de las peticiones sobre el objeto location.

Al implementar el LocationListener se nos añadirán los siguiente métodos que podrán ser sobreescritos para utilizar su funcionalidad:

- onLocationChanged(Location location).
- onProviderDisabled(String provider).
- onStatusChanged(String provider, int status, Bundle extras).
- onProviderEnabled(String provider).

Seguidamente para poder hacer uso de la información de localización se hará uso de la herramienta que nos provee la api de android denominada FusedLocationProviderClient. Esta herramienta nos proveerá la posición del GPS y permite controlar la precisión y el consumo de energía cuando está en uso. Para poder hacerlo necesitaremos también usar la objeto LocationRequest. Esta objeto nos permitirá definir las características de actualización de datos o dicho de otra forma los criterios por los cuales se enviaran las peticiones para obtener información de localización.

El codigo del cliente (FusedLocationProviderClient) y criterios de petición (LocationRequest) quedaría así:

```
1  mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY); // se
       determina el nivel de precisión
2  mLocationRequest.setInterval(UPDATE_INTERVAL); // se determina el el
       periodo de actualización
3  mLocationRequest.setFastestInterval(FASTEST_INTERVAL);// se determina el
       periodo rapido de actualización
4  LocationSettingsRequest.Builder builder = new LocationSettingsRequest.
       Builder();// se crea el objeto de opciones de petición
5  builder.addLocationRequest(mLocationRequest);// se aplican las opciones al
       nuevo objeto
6  LocationSettingsRequest locationSettingsRequest = builder.build();
7  client.requestLocationUpdates(mLocationRequest, new LocationCallback() {
8    @Override
9    public void onLocationResult(LocationResult locationResult) {
10     // do work here
11     onLocationChanged(locationResult.getLastLocation());
12   }
13 },Looper.myLooper());
```

Una vez implementado este código en el contexto de la aplicación, el método del listener onLocationChanged, nombrado anteriormente, se encargará de realizar todas las tareas de:

- Comprobación de coordenadas.
- Extracción de localidad mediante geocoder.
- Extracción de datos del tiempo mediante conexión a la api de OpenWeatherMap[3].
- Detectar si estamos en una zona de contaminación a partir de la información obtenida de la ciudad inteligente.

GoogleMapsFragment

GoogleMapsFragment es un fragmento que tiene dos tareas fundamentales en esta primera iteración del proyecto. Generar un mapa ubicándose en las coordenadas del dispositivo y precargar las zonas con polución de la localidad donde nos situemos.

Para ello primeramente haremos la generación del mapa, esta tarea puede ser una tarea costosa y se recomienda hacerlo mediante una tarea asíncrona. Además, en cuanto esté cargado se necesita mostrar el mapa inmediatamente. Para ello se ha servido del código siguiente. insertado dentro del momento de creación del fragmento dentro del ciclo de vida de la aplicación.

```
1  public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
       savedInstanceState) {
2          View rootView = inflater.inflate(R.layout.fragment_googlemaps,
               container, false);
3          mMapView = (MapView) rootView.findViewById(R.id.mapView);
4          mMapView.onCreate(savedInstanceState);
5          mMapView.onResume(); // Mostrara el mapa inmediatamente cuando se cree
               el fragmento en el momento que arranque la aplicación
6          try {
7              MapsInitializer.initialize(getActivity().getApplicationContext());
8          } catch (Exception e) {
9              e.printStackTrace();
10         }
11         mMapView.getMapAsync(new OnMapReadyCallback() {
12             @SuppressLint("MissingPermission")
13             @Override
14             public void onMapReady(GoogleMap mMap) {
15                 mapa = mMap;
16                 mapa.getUiSettings().setCompassEnabled(true);
17                 //Posición actual
18                 mapa.setMyLocationEnabled(true);
19                 //Ultima posición guardada
20                 if(myLastLatitude!=0.0){
21                     LatLng latLng = new LatLng(myLastLatitude, myLastLongitude)
                           ;
22                     mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng,
                           17));
23                 }else{
24                     LatLng latLng = new LatLng(myLatitude, myLongitude);
25                     mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng,
                           17));
26                 }
27             }
28         });
29         return rootView;
30     }
```

por otro lado la siguiente tarea necesita de la información que le suministre la ciudad inteligente a la cual estaremos conectados mediante la clase connectionMQTT que

[3]Página del servicio API OpenWeatherMap: openweathermap.org

nombraremos más adelante. A partir de la información que obtenga ira dando de alta las zonas con polución elevada, dibujando áreas en el mapa. Para ello se ha hecho uso del método addCircle del objeto mapa de google maps que hemos creado previamente. El método que se encarga de esta tarea es el siguiente:

```
1  public void setCoordinatePollution(double latitude,double longitude, int
       radius){
2      LatLng latLng = new LatLng(latitude, longitude); //Creación del objeto
           LatLng
3      Circle circle = mapa.addCircle(new CircleOptions() //Adición del area
           contaminada en el mapa previamente cargado
4              .center(latLng)//centro del area
5              .radius(radius)// radio del area
6              .strokeColor(edge) // color del filo del area
7              .fillColor(fill)); // color de relleno dela area
8   }
```

ConnectionMQTT

Esta clase se encarga de gestionar las diferentes conexiones con el servidor brokker MQTT, para realizar esta función y obtener los datos.

Para ello sus tareas son primeramente crear los objetos clientes MQTT que se conectaran al servidor y que se suscribirán a un topic determinado por su posición actual.

Una vez realizada la suscripción su tarea es difundir al resto de clases mediante el patrón observable la información recibida por parte del servidor , pero antes se encargará de transformar esa información para que sea entendible para los diferentes componentes de la aplicación.

La conexión se realiza de la siguiente forma:

```
1  public void getMqttClient(Context context, final String topic) throws
       MqttException {
2      client = new MqttAndroidClient(context, brokerUrl, clientId);
3      Log.d("I", "Success");
4      client.setCallback(this);
5      try {
6          IMqttToken token = client.connect();// prueba conexión
7          token.setActionCallback(new IMqttActionListener() {
8              @Override
9              public void onSuccess(IMqttToken asyncActionToken) {
10                 // Estamos suscritos
11                 Log.d("I", "onSuccess");
12                 try {
13                     subscribe(client,topic,0);
14                 } catch (MqttException e) {
15                     e.printStackTrace();
16                 }
17             }
18             @Override
19             public void onFailure(IMqttToken asyncActionToken, Throwable
                   exception) {
20                 Log.d("I", "onFailure");
21             }
22         });
23     } catch (MqttException e) {
24         e.printStackTrace();
25     }
26  }
```

Después de realizar el desarrollo de las clases anteriores se procedió a montar el layout del MainActivity la cual se encargará de mostrar al usuario todas las funcionalidades de la iteración inicial. Su aspecto es el siguiente:

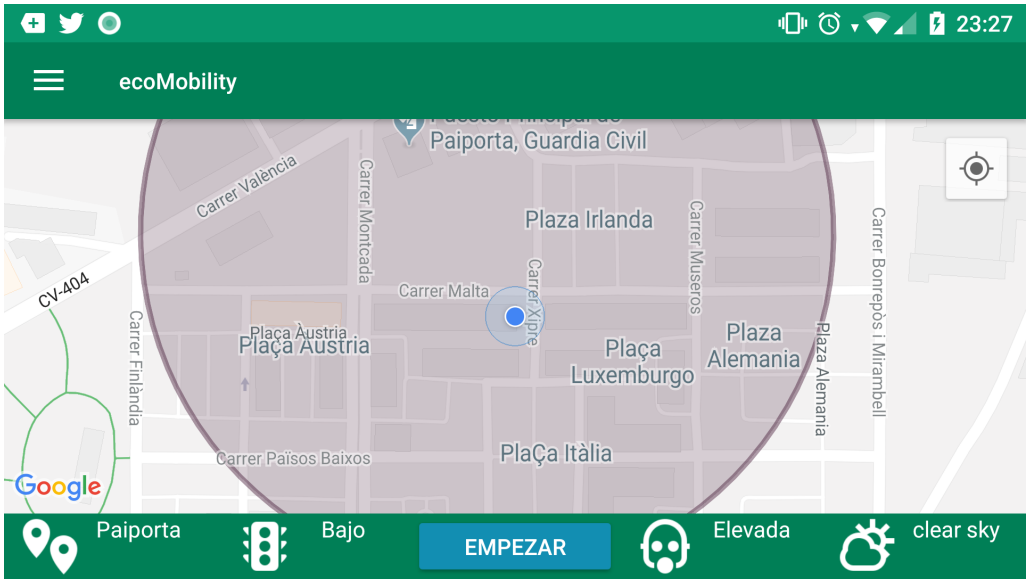

Figura 6.5: Pantalla principal de la aplicación con los datos de las funcionaldiades aplicadas: Tiempo, polución, visualización del mapa, tráfico y visualización de zonas contaminadas.

6.3.4. Segunda iteración: Modo conducción

En esta segunda iteración se busca implementar una nueva funcionalidad de las nombradas anteriormente, el modo conducción. Para ello añadiremos la funcionalidad del foco de la cámara se mueva con cada actualización de posición del dispositivo y además se añadirá la capa de tráfico. Como se explicó anteriormente esta es la primera iteración con adición de una sola funcionalidad aplicando así la metodología incremental.

Como vamos a trabajar sobre el objeto mapa obtenido en la clase GoogleMaps implementada anteriormente en la iteración de inicialización y esta se encarga de transformarlo. Esta clase será la encargada de cargar con esta nueva funcionalidad. En esta iteración para añadir la funcionalidad se centrará en dos características:

- Cámara.
- Capa de tráfico.

En la característica de cámara lo que necesitamos implementar es que cambie el foco de posición con cada actualización de la posición del dispositivo, para ello nos valdremos del patrón observable implementado anteriormente, en este caso la clase GoogleMaps ser un observador de deviceStatus el cual le avisara que las posición del dispositivo ha cambiado y se llamará al siguiente método que ejecuta el cambio de foco con las nuevas coordenadas.

```
1    public void setCoordinates(double latitude,double longitude){
2        LatLng latLng = new LatLng(latitude, longitude);//se crea el objeto
             latitud longitud
3        mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, 17)); //
             movemos la camara con la nueva posición de latitud longitud
4    }
```

Para implementar la capa de tráfico lo que necesitamos es añadir las siguientes líneas de código que lo que se encargarán es de decirle a la api de GoogleMaps que nos habilite la capa de tráfico en el mapa cargado.

```
1     public void setCoordinates(double latitude,double longitude){
2         LatLng latLng = new LatLng(latitude, longitude);//se crea el objeto
              latitud longitud
3         mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, 17)); //
              movemos la camara con la nueva posición de latitud longitud
4         if(firstCall){ // si es la primera vez que se ejecuta el método ejecuta
               el siguiente código
5             mapa.setTrafficEnabled(true);
6             firstCall=false;
7         }
8     }
```

Por último se añadirá el controlador en la clase MainActivity. Este controlador se encargará de activar esta nueva funcionalidad mediante el evento de onClick.

6.3.5. Tercera iteración: Configuración

En esta tercera iteración del desarrollo se implementó la funcionalidad de configuración de las notificaciones, para ello se hizo uso del recurso preferences que la API de Android nos provee. Las ventajas son diversas pero las más destacadas es por un lado que las preferencias de la aplicación compartirán diseño con el resto de aplicaciones de este sistema, mejorando así la experiencia de usuario y por otro lado hace la implementación mucho más sencilla.

En esta aplicación para implementarlo se ha hecho uso de una actividad secundaria que será llamada desde el mainActivity. Esta actividad secundaria tendrá como nombre PreferencesActivity y estará compuesto por un fragmentActivity el cual llamará a un fichero XML. Este fichero XML definiera las preferencias que necesitemos y los de sus datos.

El fichero XML es el siguiente.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
3      android:key="preferencias_principal" >
4      <CheckBoxPreference
5          android:key="notificaciones"
6          android:title="Mandar notificaciones"
7          android:summary="Notificar si estamos cerca de un lugar"/>
8
9      <ListPreference
10         android:key="orden"
11         android:title="Modos de alerta"
12         android:summary="Diferentes estados de aviso de notificaciones"
13         android:entries="@array/tiposOrden"
14         android:entryValues="@array/tiposOrdenValores"
15         android:defaultValue="1"/>
16 </PreferenceScreen>
```

El resultado es el siguiente:

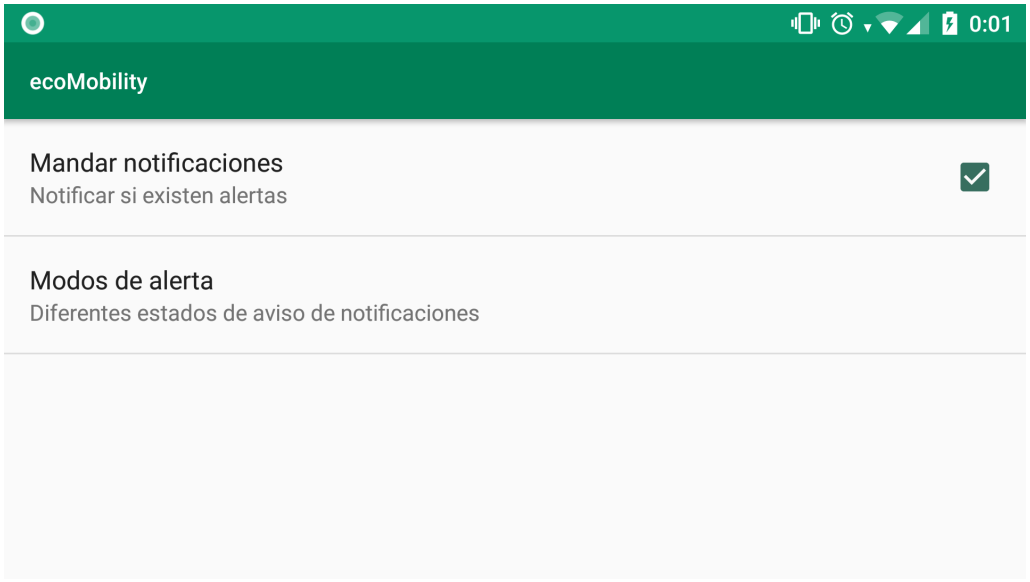

Figura 6.6: Pantalla de configuración donde podremos determinar la configuración de las notificaciones.

6.3.6. Cuarta iteración: Notificaciones

Para esta cuarta y última iteración del proyecto hemos implementado la funcionalidad de las notificaciones. En este apartado se desarrollaron las notificaciones en dos vertientes.

Por un lado para que sean visualizadas desde la aplicación mediante una actividad para esa tarea y por otro lado para que aparezcan desde la barra de notificaciones del dispositivo Android.

En la primera vertiente se hizo uso de uso de la funcionalidad recycler view en Android la cual nos permitirá reciclar la funcionalidad de crear una notificación y replicarla cada vez que ésta sea invocada reutilizando las notificaciones creadas anteriormente que no sean visibles.

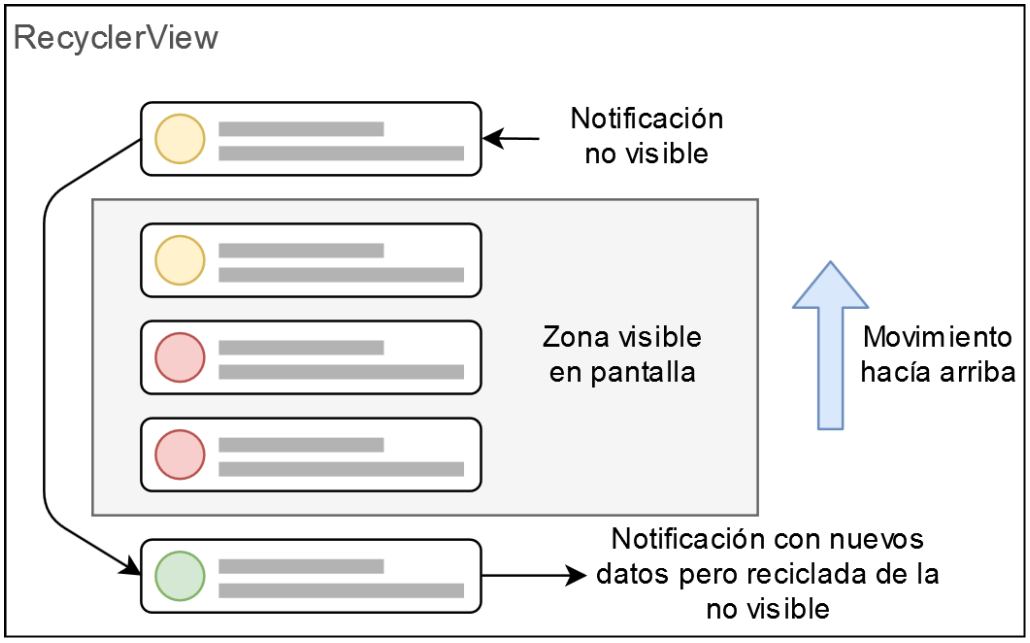

Figura 6.7: Funcionamiento del recyclerView.

Con el recyclerview ganamos en rendimiento en la aplicación gracias a su funcionamiento basado en el reciclaje, por otro lado para que funcione adecuadamente en el contexto de la aplicación necesitaremos de dos elementos más, el adaptador de notificaciones y el POJO con el cual tendremos acceso a los datos de las notificaciones que nos lleguen.[19]

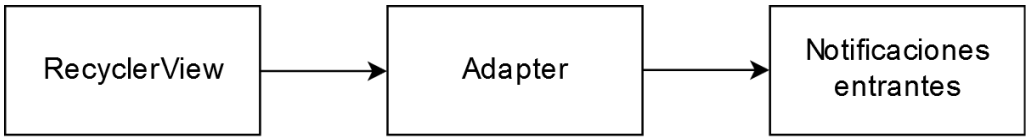

Figura 6.8: Funcionamiento global del recyclerView en la aplicación.

El resultado en la aplicación es el siguiente:

Figura 6.9: Pantalla de notificaciones donde se aprecia el funcionamiento.

En la otra vertiente se implementa para que estas notificaciones también aparecen abreviadas en la barra de notificaciones del dispositivo Android cada vez que sobrepasamos una área o llegue una notificación. Para eso implementaremos una clase denominada notificationsUtils el cual extiende de ContextWraper. Esta clase nos ayudará conforme marcan las nuevas pautas de desarrollo en android los diferentes canales para las notificaciones a nivel interno de la aplicación y nos ayudar aa gestionar como queremos que avise al usuario dependiendo del canal por el que llegue la notificación. Por ejemplo podríamos definir el color del led de notificaciones, si queremos vibración e incluso añadir si queremos que la alerta sea visible o no desde la pantalla de bloqueo.

El resultado es el siguiente:

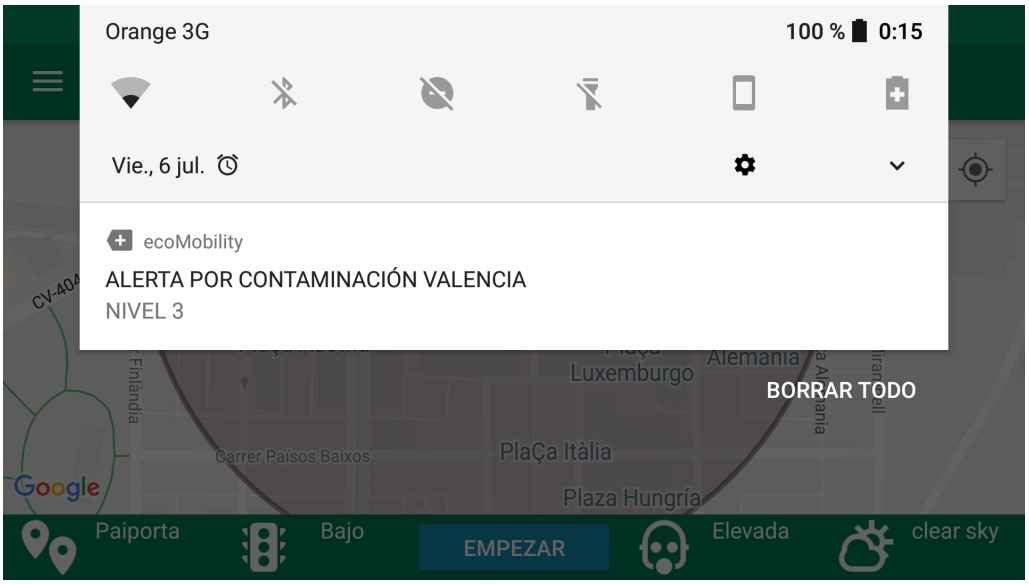

Figura 6.10: Visualización de las notificaciones en la barra de notificaciones.

6.3.7. Conjunto de componentes desarrollados

Durante la fase de desarrollo en el proyecto hemos implementado los siguientes componentes:

- MainActivity como gestionador de toda la aplicación y encargado de invocar al resto de actividades de la aplicación.
- ConnectionMQTT como gestionador de conexiones usando protocolo MQTT con el servidor MQTT brokker mosquitto.
- DeviceStatusFragment como gestionador de la situación de la aplicación (posición , si está o no en zona con alerta, etc).
- GoogleMapsFragment como gestionador del mapa.
- Notifications, NotificationsActivity, NotificationsAdapter y NotificationsUtils clases encargadas de las notificaciones.
- PreferencesActivity y PreferencesFragment como gestionadoras de cargar y establecer la configuración.

También se ha hecho uso de la metodología incremental que nos ha ayudado durante el desarrollo a repartir y reducir la carga de trabajo de forma que la adición de nuevas funcionalidades sea una tarea más sencilla, además esta metodología propicia a que si en un futuro se quiere mejorar el proyecto se pueda hacer añadiendo poco a poco más funcionalidades.

Además cabe resaltar el uso del patrón observable el cual nos ha ayudado a lo largo del desarrollo para reducir la complejidad del código y evitar en la medida de los posible el acoplamiento entre componentes.

CAPÍTULO 7
Testing

En este capítulo del proyecto se han realizado diferentes pruebas a la aplicación implementada. Estas pruebas se dividen en pruebas funcionales, de usabilidad y de rendimiento.

7.1 Pruebas funcionales

En pruebas funcionales hemos realizado pruebas a la aplicación para comprobar si todas las funcionalidades requeridas en la fase de análisis del proyecto verdaderamente las cumple.

Para ello se han realizado pruebas manuales a la aplicación a lo largo del desarrollo que han ayudado a probar una a una todas las implementaciones aplicadas iterativamente y viendo que en su conjunto funcionan correctamente. Por otro lado se generó un servidor de pruebas para generar mensajes ficticios de avisos y poluciones.

Muchas de estas pruebas consisten en:

- Recepción de los datos que envía el brokker MQTT al cliente suscrito.
- Desplazamiento del dispositivo móvil físicamente por zonas marcadas por el mapa como contaminadas y ver que detecta que entra en una de ellas.
- Recepción de notificaciones que puedan ser mostradas en la app.

7.2 Pruebas de usabilidad

Antes de la realización d ela implantación ya se habían realizado pruebas a los prototipos pensando en la calidad del producto final antes de ponerse a desarrollarlo, una vez hecho se probó la aplicación con usuarios reales de dos perfiles de usuarios: el perfil objetivo y un perfil experto.

De estas pruebas, en el desarrollo realizado, se determinaron cambios y mejoras leves de iconos utilizados y paleta de colores a usar.

7.3 Pruebas de rendimiento

Gracias a Android studio podemos ver estadísticas en tiempo real del rendimiento de la aplicación sobre nuestro dispositivo de pruebas y mediante un testeo de un periodo de uso medio de sesiones de 10 minutos. Se han extraído los siguientes datos.

Características dispositivo	
Modelo	Xiaomi MI A1
Procesador	Snapdragon 625 , 2,2GHz, 8 cores
RAM	4GB
Sistema Operativo	Android 8

Tabla 7.1: Información del dispositivo de pruebas.

Datos de rendimiento de la aplicación durante sesiones de 10 min	
Máximo de consumo de CPU	51 %
Máximo de consumo de RAM	225MB

Tabla 7.2: Datos de rendimiento obtenidos durante las pruebas.

CAPÍTULO 8
Manual de usuario

En este capítulo se desarrollará el manual de la aplicación en base a lo desarrollado explicando cada uno de los apartados y cómo se deberían de interpretar los datos expuestos.

8.1 Pantalla principal

En la pantalla principal de la aplicación tendremos acceso a la información relevante de la situación de la zona donde nosotros estemos ubicados.

Para ello la aplicación estructura la pantalla en tres zonas:

- Acceso menú.
- Mapa con previsualización de datos dependiendo de si estamos o no estamos en modo conducción nos mostrará una información u otra.
- Acceso a datos de nuestra posición: Localidad, tráfico, contaminación y tiempo. Además del acceso al botón de activación modo conducción.

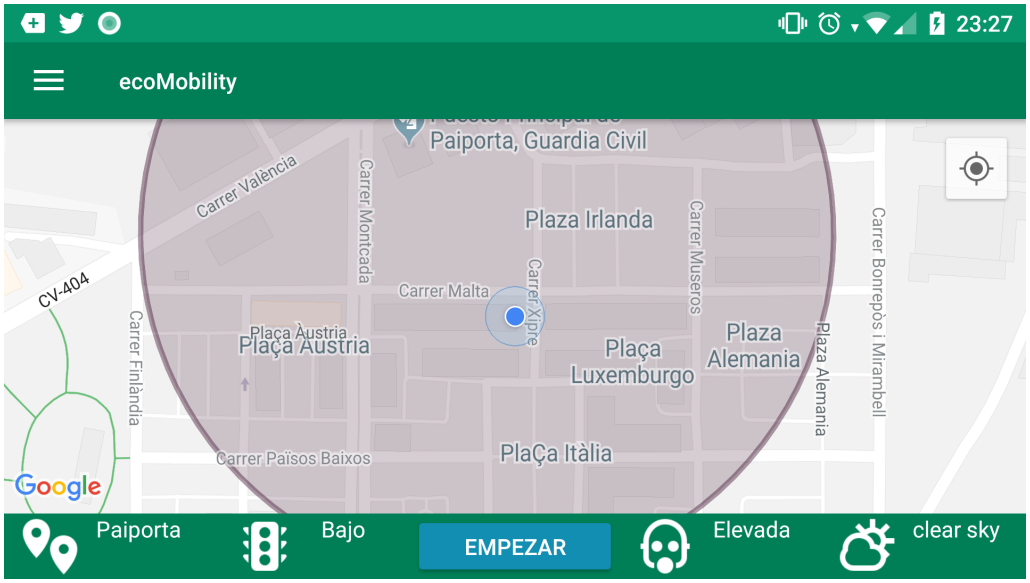

Figura 8.1: Pantalla principal de la aplicación en la cual podemos consumir los datos que nos provee la ciudad inteligente.

8.1.1. Botón menú

Botón de acceso al menú, este botón nos permitirá acceder a las pantallas de configuración y notificaciones.

Figura 8.2: Botón menú de la pantalla principal.

8.1.2. Mapa

En esta parte de la pantalla principal podremos consultar el mapa de nuestra zona actual. La información que nos provee nada má arrancar la aplicación es la polución de la zona actual, representada en áreas de mayor o menor medida dependiendo de la cantidad de sensores que disparan el aviso.

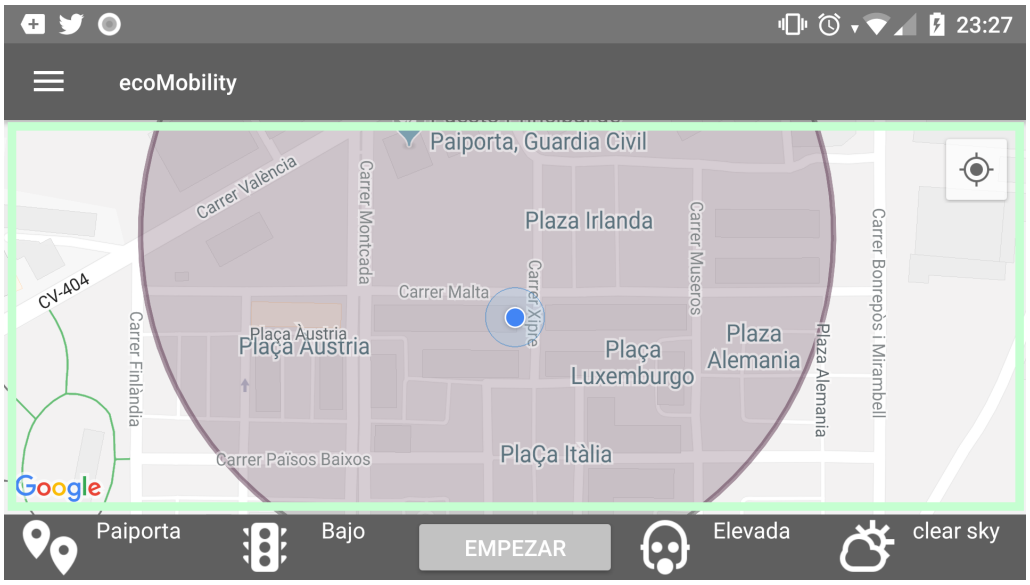

Figura 8.3: Mapa de la pantalla principal con información de polución.

8.1.3. Información de localidad

En esta zona de la pantalla principal y representado con un icono de chincheta se podrá consultar la información de en qué localidad está situado el dispositivo.

Figura 8.4: Datos sobre la localidad o zona donde esta situado el usuario.

8.1.4. Información tráfico

En esta zona de la pantalla principal resaltada en la figura y representado con un icono de semáforo se podrá consultar la información de tráfico de la zona. Los diferentes estados de tráfico son:

- Bajo
- Medio
- Alto

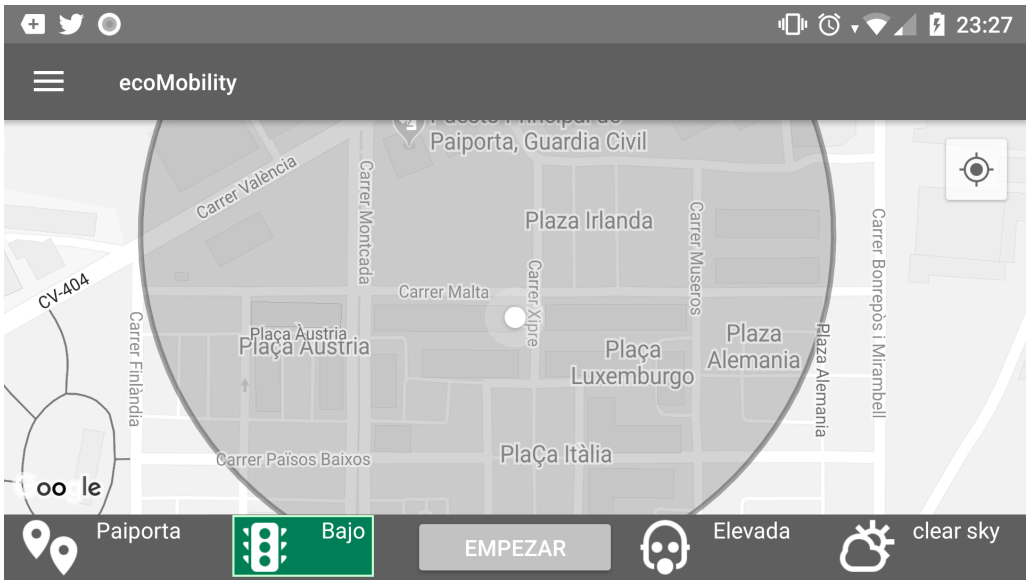

Figura 8.5: Datos sobre el tráfico de la zona donde esta situado el usuario.

8.1.5. Información contaminación

En esta zona de la pantalla principal resaltada en la figura y representado con un icono de mascarilla de gas se podrá consultar la información de contaminación de la zona. Los diferentes estados de contaminación son:

- Bajo
- Medio
- Elevado
- Muy elevado
- Extremo

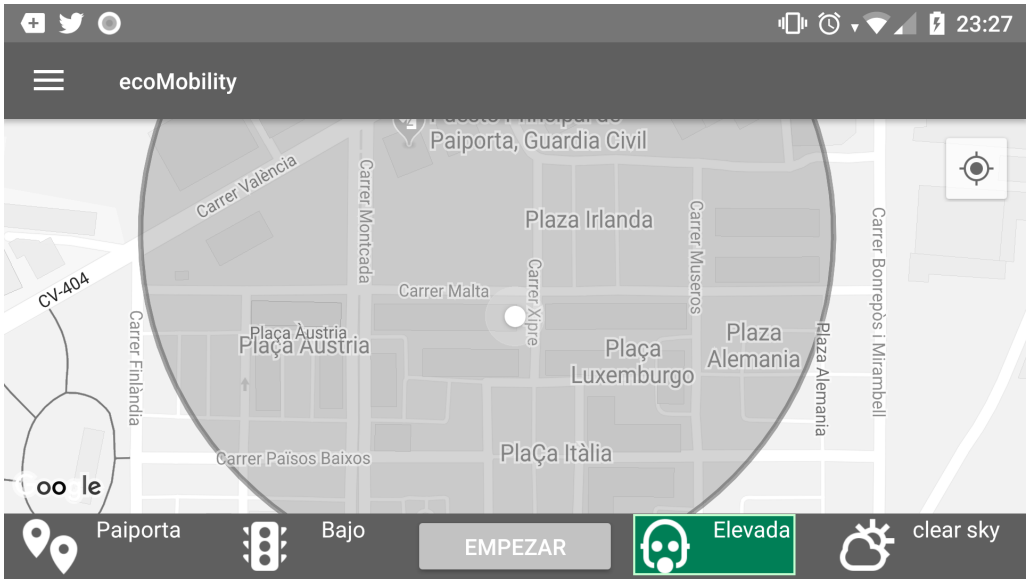

Figura 8.6: Datos sobre la contaminación de la zona donde esta situado el usuario.

8.1.6. Información meteorológica

En esta zona de la pantalla principal resaltada en la figura y representado con un icono de sol y nubes se podrá consultar la información meteorológica de la zona. Los diferentes estados del tiempo dependen de los valores de la API OPENWEATHER, pero los más usuales son:

- Despejado
- Parcialmente despejado
- Nublado
- Lluvioso

Figura 8.7: Datos sobre el tiempo de la zona donde esta situado el usuario.

8.1.7. Botón modo conducción

En este botón activaremos el modo conducción que su funcionalidad se detalla más adelante.

Figura 8.8: Botón de activación de modo conducción.

8.2 Menú

En esta parte del programa presentamos el menú, este menú da acceso a las pantallas de configuración y notificaciones del programa.

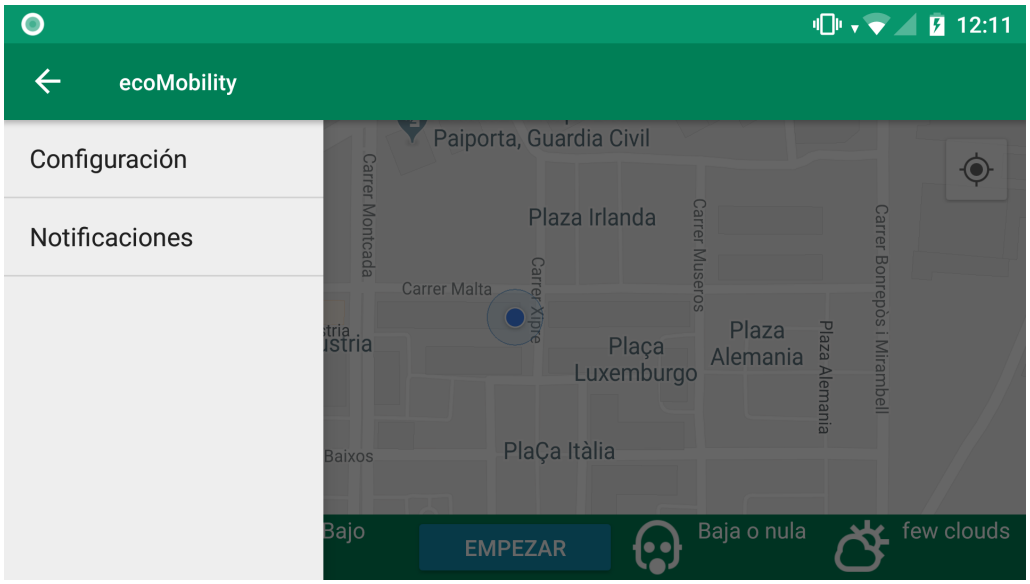

Figura 8.9: Menú del programa.

En la siguiente imagen se muestra como volver al estado anterior y volver a la pantalla principal mediante el botón de vuelta atrás o back. Por otro lado destacar que si se selecciona cualquier parte externa del menú también se vuelve al menú principal.

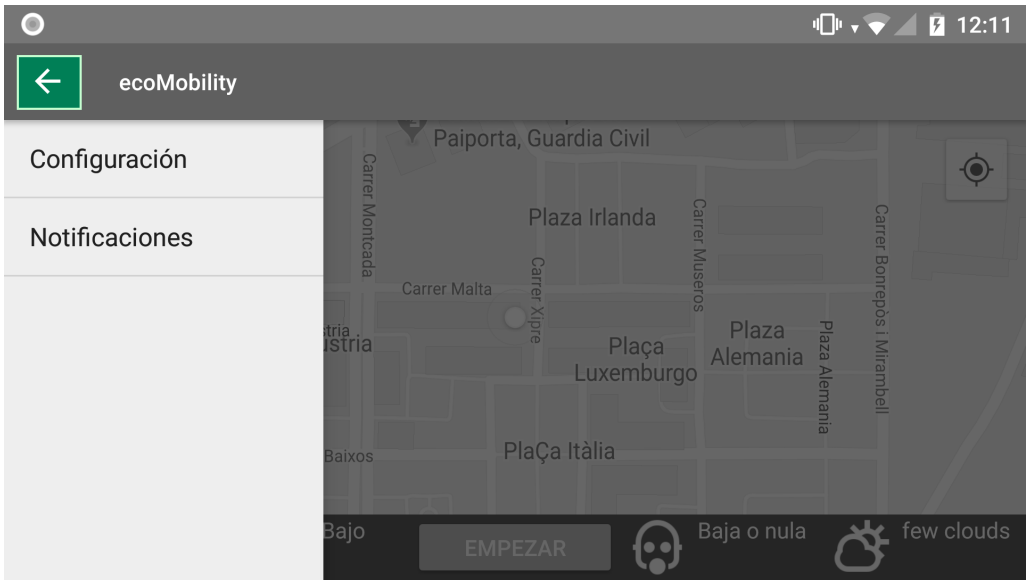

Figura 8.10: Botón para volver a la pantalla principal.

8.3 Pantalla Configuración

En la siguiente figura se presenta la pantalla de configuración, esta pantalla nos permite habilitar o deshabilitar las notificaciones y determinar cómo se quiere que se nos notifique.

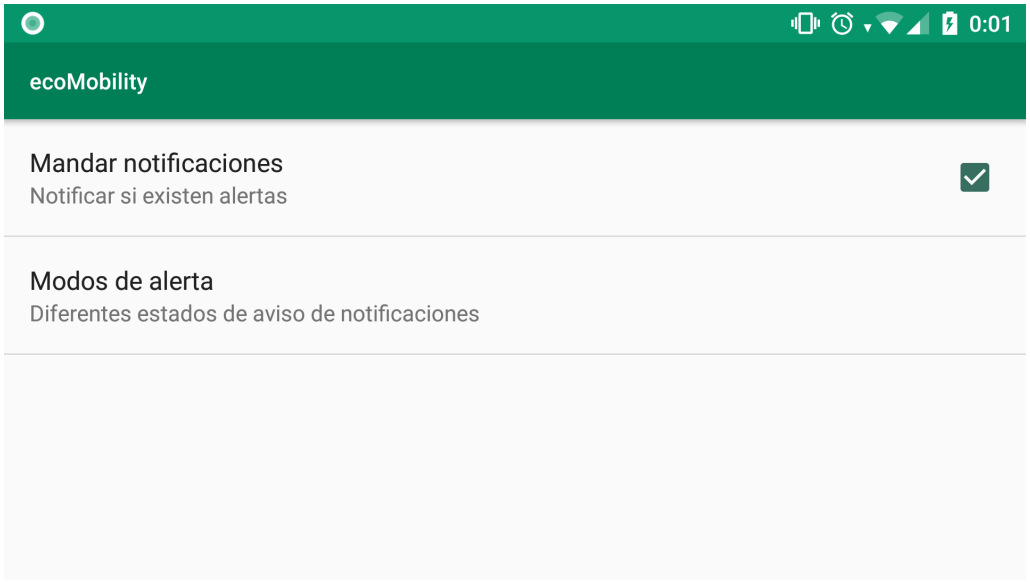

Figura 8.11: Pantalla de configuración donde podremos determinar la configuración de las notificaciones.

Los modos de alerta son:

- **Máximo:** Alerta de todas las notificaciones entrantes.
- **Intermedio:** Las alertas por barra de estado del dispositivo no llegan solo se podrán ver las notificaciones en la aplicación.
- **Mínimo:** Las notificaciones que llegan a la aplicación y solo a la aplicación son solo las de mayor gravedad , las de gravedad en nivel bajo no avisan.

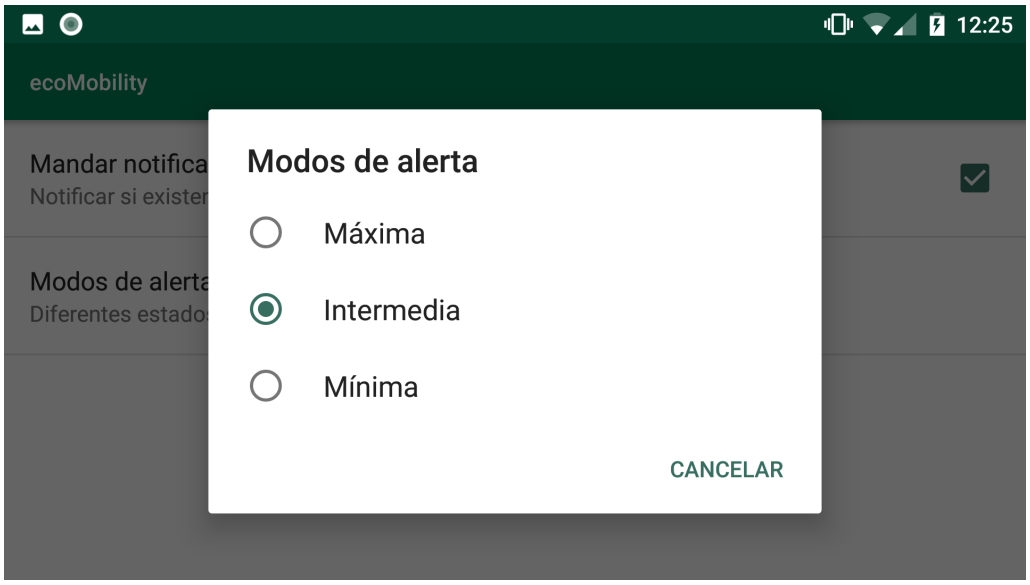

Figura 8.12: Diferentes modos de configuración de alertas.

8.4 Pantalla Notificaciones

En la pantalla de notificaciones que se puede acceder desde el menú de la pantalla principal dispondremos de la información de las alertas que nos suministra la ciudad inteligente. Por otro lado las alertas pueden ser de diferentes niveles y mediante el título de la notificación nos informa la ciudad inteligente de lo que está pasando.

En la siguiente figura se muestra un ejemplo de la información que puede llegar.

Figura 8.13: Pantalla de notificaciones donde se pueden consultar las notificaciones que nos van llegando desde la ciudad inteligente.

CAPÍTULO 9
Conclusión

En este último capítulo se expondrá un resumen de lo desarrollado y expuesto en este proyecto final de grado, además de mencionar todas las conclusiones que se han determinado durante el desarrollo y explicar la relación del trabajo realizado con los estudios cursados. Por otro lado se comentarán posibles mejoras para la aplicación desarrollada.

9.1 Conclusiones

Durante la realización de este proyecto final de grado, hemos analizado las necesidades que justifican la realización del proyecto. Además se ha expuesto el contexto actual el cual ayuda a cualquiera que lea el proyecto a entender la situación y el porqué de su desarrollo.

También, se ha desarrollado una aplicación escalable, capaz de conectarse con las ciudades inteligentes y de gestionar toda la información que recibe para que esta sea notificada al usuario. Por otro lado y no menos importante se han planteado diferentes interfaces o pantallas durante el transcurso del desarrollo, las cuales en cada una de las iteraciones se ha mejorado para así conseguir un objetivo inherente en cualquier desarrollo, la usabilidad y la experiencia de usuario.

En definitiva, se ha conseguido desarrollar un módulo de comunicaciones con la ciudad inteligente, dentro del marco del macroproyecto ecoMobility, es por eso que este módulo sirve como claro ejemplo de que el futuro de las aplicaciones conectadas con la ciudad, para mejorar la vida de las personas, está cada vez más cerca.

9.2 Relación del trabajo desarrollado con los estudios cursados

Para poder realizar el presente trabajo final de grado se han necesitado diversos conocimientos adquiridos a lo largo del periodo cursado en la Universidad Politécnica de Valencia. De entre todos los conocimientos adquiridos cabe resaltar, los que provienen de las asignaturas de programación: "introducción a la programación y programación". Ya que sin ellos no se hubiese podido realizar la aplicación. Tampoco de disponer de la suficiente soltura como para poder embarcarse en este proyecto. Por otro lado y no menos importante, tenemos que mencionar asignaturas como:

- **Desarrollo centrado en el usuario:** Que gracias a sus conocimientos se ha podido diseñar una aplicación bonita y usable, ya que aplicando los conocimientos sobre el uso de prototipos, para obtener mediante diferentes pruebas iterativas a usuarios

objetivos, el mejor diseño de prototipo de interfaz de la aplicación antes de ponerse a desarrollarla. Ahorrando volumen de trabajo.

- **Gestión de proyectos:** Sin esta asignatura es posible que el proyecto aún estuviera en fase de desarrollo ya que la planificación y el cómo se ha abordado la realización del presente proyecto bebe de los conocimientos adquiridos en esta asignatura.

- **Integración de aplicaciones:** Gracias a esta asignatura cursada ha sido más sencillo poder comprender cómo encajar las diferentes fuentes de datos provenientes de diferentes servicios o fuentes. Además con los conocimientos adquiridos de esta asignatura se pudo saber por dónde orientar el proyecto, cómo enfocar las comunicaciones y transformar la información para la aplicación.

En definitiva seguramente me olvide de relacionar algún conocimiento adquirido de alguna asignatura con el proyecto, dado que muchos detalles y formas de trabajar se han ido adquiriendo durante el desarrollo de la carrera y todas las asignaturas han sumado su grano de arena.

Por otro lado durante el periodo de formación en la carrera se forjaron diferentes competencias transversales, que también se han aplicado en el proyecto. Como por ejemplo:

- **Comprensión e integración:** En el presente proyecto esta competencia transversal ha sido clave para el desarrollo ya que gracias a ella se han podido asimilar nuevos conocimientos e ideas relacionadas con la forma de trabajar con aplicaciones Android, ya que estos conocimientos no los di durante el periodo de formación pero que gracias a la adquisición de esta competencia transversal se han podido asimilar estos nuevos conocimientos.

- **Análisis y resolución de problemas:** Esta competencia transversal ha sido clave en el proyecto ya que ha servido para determinar toda la etapa de análisis y diseño de este proyecto.

- **Innovación creatividad y emprendimiento:** Esta competencia transversal hace su aparición en el presente proyecto en la sección del diseño de los prototipos ya que gracias a la creatividad e iniciativa de realización de iteraciones para mejorar, se ha conseguido disponer de una mejor aplicación.

- **Diseño y proyecto:** Durante el transcurso de las primeras etapas del proyecto esta competencia ha sido clave para contextualizar la aplicación a realizar con la realidad tanto social, logística y tecnológica.

- **Aplicación y pensamiento práctico:** Gracias a esta competencia transversal se han podido resolver situaciones complicadas durante el desarrollo gracias a conocimientos previamente adquiridos durante la carrera.

- **Pensamiento crítico:** Esta competencia transversal está presente en el proyecto ya que se decidió aplicar una metodología iterativa, pensando ya en mejorar constantemente lo desarrollado teniendo como máxima que todo es mejorable.

- **Aprendizaje permanente:** Para la realización del proyecto se ha tenido que adquirir nuevos conocimientos formándome de forma autodidacta mediante cursos y buscando información por la red de redes, internet.

9.3 Mejoras futuras

Durante el desarrollo también han surgido nuevas ideas de mejora de la aplicación que por motivos de tiempo no se han podido implementar. Además también al ser usada la aplicación por diferentes usuarios durante la fase de testing, estos han aportado nuevas ideas que le podrían venir bien a un módulo de comunicaciones entre los usuarios y la ciudad inteligente que se ha desarrollado. En este apartado enumeramos todas ellas.

Las mejoras son las siguientes:

- Mejora y optimización de la aplicación dentro del entorno Android Auto.
- Establecer un sistema de información extendida de las notificaciones.
- Añadir alertas de polen para avisar a los usuarios con alergias.
- Añadir modo nocturno a la aplicación para las horas con menos luz del día.
- Añadir opción de ruta a la aplicación para saber si esta cruza una zona contaminada o con incidencias.
- Recibir notificaciones aunque la aplicación esté apagada si esta está configurada para notificar de todo.

Bibliografía

[1] Comunicado de prensa de la Organización Mundial de la Salud, emitido el 25 de marzo de 2014. Consultado en http://www.who.int/mediacentre/news/releases/2014/air-pollution/es/.

[2] Infografía del IESE Business School, Universidad de Navarra, emitido en 2016. Consultado en https://www.iese.edu/Aplicaciones/upload/ESIESECitiesInMotion2016.pdf.

[3] Ciudad inteligente, emitido el 30 junio 2018. Consultado en https://es.wikipedia.org/wiki/Ciudad_inteligente.

[4] ¿Qué es y cómo funciona el Internet de las cosas?, emitido el 20 octubre 2014. Consultado en https://hipertextual.com/archivo/2014/10/internet-cosas/.

[5] Mapa de la Red Española de Ciudades Inteligentes, emitido en 2017. Consultado en http://www.redciudadesinteligentes.es/images/municipios/mapa-ciudades/miembros-reci.pdf.

[6] Ley 27/2006, de 18 de julio, por la que se regulan los derechos de acceso a la información, de participación pública y de acceso a la justicia en materia de medio ambiente (incorpora las Directivas 2003/4/CE y 2003/35/CE), emitido el 19 de julio de 2006. Consultado en https://www.boe.es/buscar/doc.php?id=BOE-A-2006-13010.

[7] Qué es REST, emitido el 25 de abril de 2014. Consultado en https://desarrolloweb.com/articulos/que-es-rest-caracteristicas-sistemas.html.

[8] Desarrollo iterativo e incremental Consultado en https://proyectosagiles.org/desarrollo-iterativo-incremental/.

[9] Consulta los últimos datos de Kantar Worldpanel sobre cuota de mercado de smartphones en España, emitido el 15 de junio de 2017. Consultado en https://es.kantar.com/tech/m%C3%B3vil/2017/junio-2017-cuota-de-mercado-de-smartphones-en-espa%C3%B1a-2017/.

[10] ¿Qué es la tecnología Java y para qué la necesito? , Consultado en https://www.java.com/es/download/faq/whatis_java.xml.

[11] ¿Qué es Kotlin?, emitido el 17 mayo 2017. Consultado en https://platzi.com/blog/que-es-kotlin/.

[12] XML Technology, Consultado en https://www.w3.org/standards/xml/.

[13] Actividades, Consultado en https://developer.android.com/guide/components/activities?hl=es-419.

[14] Fragmentos, Consultado en https://developer.android.com/guide/components/fragments?hl=es-419.

[15] Material Guidelines, Consultado en https://material.io/design/.

[16] Cliente-servidor, emitido el 2 junio 2018. Consultado en https://es.wikipedia.org/wiki/Cliente-servidor.

[17] Android: Introducción a la Programación, Consultado en https://www.edx.org/course/android-introduccion-a-la-programacion.

[18] ¿Qué es el Android manifest?, Consultado en http://www.tuprogramacion.com/glosario/que-es-el-android-manifest/.

[19] Un vistazo rápido al nuevo RecyclerView , emitido el 9 octubre 2014. Consultado en http://androcode.es/2014/10/un-vistazo-rapido-al-nuevo-recyclerview/.