UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Involucrando al humano en la toma de decisiones en sistemas autónomos. Diseño e Implementación de Servicio de Atención a Emergencias en Ciudades Inteligentes

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Ignacio Lázaro Andrés

Tutor: Joan Josep Fons Cors
Cotutor: Vicente Pelechano Ferragud

Curso 2016-2017

Agradecimientos

A Laura,
por su infinita paciencia y su constante apoyo,
por estar siempre a mi lado.

A Apa, Javi y Pablo,
por todas las respuestas a esas dudas,
por ser verdaderos amigos.

A Connor y Patri,
por ayudarme a desconectar del trabajo
y llenarme la cabeza con nuevas ideas.

A mi familia,
por estar siempre ahí sin ninguna duda.

A Miriam Gil,
por ayudarme en este trabajo
de manera desinteresada.

Y, a mi tutor, Joan,
por sus infinitos consejos y
por introducirme en un mundo de la informática
antes desconocido.

Muchas gracias

Resum

Aquest treball presenta l'anàlisi, disseny i implementació d'un sistema pensat per a l'interacció del humà amb vehicles i altres components intel·ligents trobats en ciutats intel·ligents, dins del marc del Internet de les Coses.

Per una banda, s'han definit uns modes que representen els graus de consciència que pot tindre un usuari dins del seu vehicle, respectant factor com la il·luminació, el so o la quantitat de passatgers que puga trobar-se en el vehicle.

Per altra banda, aquest sistema notifica a l'usuari de les interaccions rebudes per altres components intel·ligents a través de peticions, tenint en compte el seu grau de consciència i el tipus de petició, permitint a l'usuari responder-les.

Per tant, s'ha desenvolupat una aplicació que envolta a l'èsser humà en la comunicació entre diferents elements intel·ligents dins d'una ciutat intel·ligent.

Paraules clau: Android, Ciutat Inteligent, Interacció, Notificació, Consciència

Resumen

Este trabajo presenta el análisis, diseño e implementación de un sistema pensado para la interacción del ser humano con vehículos y componentes inteligentes hallados en ciudades inteligentes, dentro del marco del Internet de las Cosas.

Por una parte, se han definido unos modos que representan los grados de consciencia que puede tener un usuario dentro de su vehículo, respetando factores como la iluminación, el sonido o la cantidad de pasajeros que pueda hallarse en el vehículo.

Por otra parte, este sistema notifica al usuario de las interacciones recibidas por otros componentes inteligentes a través de peticiones, teniendo en cuenta su grado de consciencia y el tipo de petición, permitiendo al usuario responderlas.

Por lo tanto, se ha desarrollado una aplicación que involucra al ser humano en la comunicación entre distintos elementos inteligentes dentro de una ciudad inteligente.

Palabras clave: Android, Ciudad Inteligente, Interacción, Notificación, Consciencia

Abstract

This project develops the analysus, design and implementation of a system thought for the human interaction with intelligent vehicles and other components found in smartcities, within the ambit of Internet Of Things.

On one hand, some modes have been defined that represent the degrees of consciousness a user may have within his vehicle, respecting factors such as the light, the sound or the passengers that can be placed in the vehicle.

On the other hand, this system notifies the received interactions by other intelligent components to the user through requests, tacking in account his degree of consciousness and teh type of notification, allowing answer them to the user.

Therefore, an appliccation that involves the human in the communication among several intelligent elements within a smartcity has been developped.

Key words: Android, Smartcity, Interaction, Notification, Conscience

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En este capítulo se va a describir la motivación por la cual se ha realizado este proyecto, los objetivos que se quieren conseguir tras su desarrollo, la estructura que sigue este documento y la presentación de otros trabajos relacionados.

1.1 Motivación

Nos encontramos en un mundo de constante evolución. Se ha alcanzado un punto en que diversos dispositivos corrientes (lavadoras, televisiones, neveras etc.) han empezado a comunicarse entre ellos, en un flujo constante de información. Este hecho se denomina Internet de las Cosas[1], un término que comprende la futura comunicación de todos los dispositivos electrónicos entre sí sin necesitar la intervención del humano.

Siguiendo este marco, hay incluso ciudades que han empezado a entrar en esta evolución. Por ejemplo, Álvaro Cárdenas[2] menciona en su artículo que hay grandes ciudades que están siendo rediseñadas para ser gestionadas de una forma más sostenible e independiente y así conseguir ofrecer una mejor variedad de servicios a los ciudadanos. De hecho, poblaciones como Madrid y Valencia han comprado *software* preparado para esta evolución tecnológica.

Además, no solo evolucionan las ciudades. Componentes básicos de estos entornos como carreteras, señales de tráfico, ambulancias y otros van recibiendo pequeños prototipos y mejoras a la espera de un producto que pueda ser integrado en unas futuras ciudades inteligentes. De hecho, como escribe Érika Fernández[3] en su artículo, empresas como Google y Tesla han desarrollado vehículos inteligentes y autónomos totalmente funcionales e, incluso, han sido lanzados al mercado. Todo esto, compondrá en un futuro en una red de comunicación constante entre todos estos componentes que habilitará la gestión propia a una ciudad.

Sin embargo, es imprenscindible involucrar al humano dentro de esa red. Se necesita que reciba la información importante de una forma clara, concisa y no intrusiva, permitiéndole responder a las diferentes situaciones que se le planteen. Además, es necesario respetar el grado de consciencia que tenga en ese momento, es decir, tener en cuenta si el usuario se haya en una situación en la que pueda estar atento a todas las notificaciones que reciba o, en caso contrario, necesita solo ser completamente avisado de las más importantes. Por lo tanto, el presente trabajo de final de grado plantea llevar a cabo una aplicación que permita la participación del usuario en el bucle de información de una ciudad (*Human In The Lopp*[4]), recibiendo, como ya se ha mencionado, información en forma de peticiones de una forma no intrusiva, respetando la situación en la que se halla

y su grado de consciencia, así como que le sea permitido responder a esas peticiones, teniendo en cuenta esa respuesta en la comunicación.

1.2 Problemática

A pesar de que, inicialmente, la finalidad de este trabajo era desarrollar un sistema que gestionara el servicio de emergencias de una ciudad inteligente involucrando al humano en la toma de decisiones, se ha implementado un sistema general que permite interactuar al usuario con los dispositivos inteligentes en todo tipo de situaciones y para toda variedad de servicios que puedan hallarse en una ciudad. Introduce al humano en el bucle (*Human in the Loop*) de comunicación de las distintas máquinas que pueden hallarse en las ciudades inteligentes en un futuro

Para conseguir esa meta, como se explicará en este trabajo de fin de grado, se necesitan unos mecanismos que permitan notificar al usuario de la información necesaria y de las peticiones recibidas de forma constante, teniendo en cuenta las posibles respuestas que pueda dar y su grado de consciencia.

1.3 Objetivos

Teniendo en cuenta la problemática expuesta en el apartado anterior, este trabajo de final de grado está dirigido a personas que posean o sean usuarias de un vehículo autónomo e inteligente o pertenezcan a la asistencia de un servicio en la ciudad. Por lo tanto, se debe marcar los siguientes objetivos:

- Preparar una plataforma que permita la comunicación entre vehículos inteligentes y los diversos componentes inteligentes que puedan hallarse en una ciudad (dentro del marco del Internet de las Cosas) a través del estándar de comunicación MQTT[5].
- La plataforma ha de proveer información y notificar al usuario teniendo en cuenta su grado de consciencia, así como permitirle responder a esas notificaciones y enviar dichas respuestas a través del tópico al que esté suscrito en MQTT.

1.4 Plan de trabajo

En esta sección, se va enseñar la identificación y el desglose de tareas de este proyecto. Además, se va a explicar la metodología utilizada para el desarrollo de este proyecto. En la imagen **??**, se puede ver la descomposición de tareas que se ha hecho sobre el trabajo:

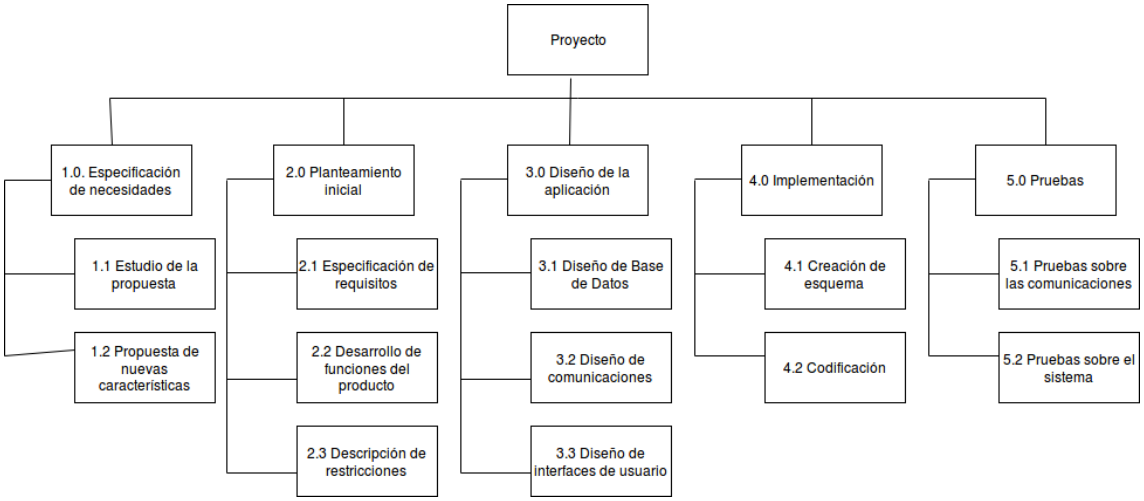

Figura 1.1: Planificación de tareas.

En cuanto la metodología utilizada, se ha utilizado un modelo en cascada, recorriendo todas las fases del proyecto de manera secuencial. Se ha escogido este modelo porque, al ser un proyecto realizado individualmente (con ayuda del tutor), era la forma más fácil de planificar y desarrollar el trabajo.

1.5 Estructura de la memoria

Este documento comenzará explicando el contexto tecnológico en el que se ha desarrollado la aplicación, es decir, los entornos, librerías y componentes utilizados en su implementación. A continuación, se realizará un planteamiento inicial del sistema, comentando las funciones del producto, seguidas de las restricciones y dependencias que contiene el proyecto, así como los requisitos del sistema desarrollado. Seguidamente se propondrá un análisis del problema, que contendrá los casos de uso de la aplicación y los correspondientes diagramas. Posteriormente, se desarrollará el diseño de la solución, teniendo en cuenta las comunicaciones e interfaces de usuario. después de lo cual se comentará la implementación, donde se explicará la arquitectura utilizada junto así como las capas implementadas. Para concluir, este documento termina con las conclusiones y con la posibles mejoras que se le pueden realizar al proyecto.

CAPÍTULO 2
Contexto tecnológico

En este capítulo, se comentará el entorno utilizado para desarrollar este trabajo de final de grado, así como una explicación de las librerías utilazadas y los diferentes componentes englobados en el proyecto.

2.1 Entornos

La aplicación ha sido desarrollada para Android[6], un sistema operativo basado en el núcleo Linux. A pesar de que, inicialmente, fue pensado para teléfonos móviles, ha llegado a distintos dispositivos como relojes, tabletas, televisiones e incluso coches. Android ha conseguido estar presente en una gran cantidad de instrumentos tecnológicos y, gracias a eso y a su facilidad de desarrollo, muchos programadores han decidido crear sus aplicaciones para este sistema operativo.

En cuanto al entorno de desarrollo usado en este proyecto, se ha optado por Android Studio[7]. Este entorno, basado en *IntelliJ IDEA*[8] de *JetBrains*, compila en el lenguaje de programación Java[9], lo que permite una integración con todas las utilidades y librerías ya desarrolladas para este lenguaje. Además, tiene importantes características como un editor gráfico de interfaces de usuario, soporte de contrucción basado en *Gradle* y, el hecho de ser el entorno oficial de desarrollo en Android, permite evolucionar al programador junto al sistema operativo.

Sin embargo, hay que mencionar que existen otras alternativas para desarrollar aplicaciones en Android. Se pueden usar los compilador como Eclipse o *Netbeans* instalando el SDK de Android. También, puedes optar por programar en un lenguaje distinto a Java. Entornos como *Xamarin* o *Unity*, usan C# para la creación de aplicaciones y juegos, respectivamente.

2.2 Librerías

Como ya se ha mencionado en la introducción de este trabajo, la aplicación utiliza el estándar de comunicación de Internet de las Cosas, MQTT. Para soportar dicha especificación, se ha utilizado la librería Eclipse Paho[10]. Esta librería, desarrollada en Java, permite la suscripción de N dispositivos a un tópico para recepción y la publicación de mensajes a través del protocolo MQTT. A pesar de estar escrito en Java, se puede utilizar tanto para el desarrollo en Android y en Phyton.

Además, podemos encontrar varios clientes de MQTT creados con la librería Eclipse Paho, como MQTT.FX[11], que facilitan y dan un mayor entendimiento sobre el funciona-

miento y la utilidad del protocolo MQTT. (MQTT.FX ha sido usado para realizar pruebas en este proyecto).

2.3 Componentes

A continuación, se procede a una descripción de los componentes más importantes utilizados en el trabajo:

- **Actividades**[12]: Son los componentes más básicos del desarrollo en Android. Definen una ventana o pantalla que puede contener una interfaz de usuario, permitiendo interactuar con ellas.
- **Servicios**[13]: Son elementos que son utilizados para realizar operaciones a largo plazo en un segundo plano, donde no puedan interactuar con el usuario.
- **Fragmentos**[14]: Estos componentes se podrían definir como una pieza de interfaz de usuario que está embebida en una actividad y que puede ser utilizada para conseguir una gran variedad de resultados. Un tipo de fragmento muy utilizado es con forma de alerta.
- **Notificaciones**[15]: Estas herramientas sirven para notificar al usuario de posibles eventos que han sido lanzados por aplicaciones. Además, permiten utilizar distintos mecanismos (vibración, sonido, uso de voz, etc.) para llevar a cabo su función.

CAPÍTULO 3
Planteamiento inicial del escenario

3.1 Introducción

3.1.1. Propósito

La meta de esta sección es establecer las diferentes características y funcionalidades que han sido desarrolladas para este producto.

Además, se definirán las posibles restricciones y dependencias que pueda tener la aplicación con elementos externos, como librerías y componentes, que han sido integrados en este proyecto.

3.2 Definición general

3.2.1. Funciones del producto

Avanzando con la redacción del trabajo, se va a continuar con una descripción de las funciones del producto a las que puede acceder un usuario que use la aplicación. Dichas funciones son:

- Marcar el grado de consciencia que posee siendo tres los posibles: Consciente (*Aware*), Semiconsciente (*SlightlyNoticeable*) e Invisible (*Invisible*).
- Recibir información a través de notificaciones que utilizarán mecanismos de notificación según el grado de consciencia definido por el usuario.
- Consultar las notificaciones ya recibidas, pudiendo visualizar los detalles de cada una.
- Contestar las notificaciones recibidas que lo requieran de distintas formas como escritura de texto o elección de opciones.

3.2.2. Restricciones

En este apartado se van a describir las limitaciones que acotan a este proyecto:

- La aplicación estará soportado por sistemas operativos Android 5.0 o superiores.
- El proyecto necesita una constante conexión a Internet para la recepción y el envío de información a través del protocolo de mensajería MQTT.

- La necesidad de poseer un vehículo inteligente o un dispositivo inteligente para que el sistema pueda funcionar (aunque para la realización del proyecto se ha usado un dispositivo móvil).

3.2.3. Dependencias

En este punto se van a explicar aquellos elementos que pueden afectar a la aplicación en caso de realización de cambios en ellos mismos.

Referente al protocolo de mensajería MQTT y la librería The Eclipse Paho Project

Teniendo en cuenta el constante cambio que está sufriendo el estándar MQTT, cualquier actualización que supusiera un cambio en la estructura o en el funcionamiento de este protocolo y que la librería utilizada no se evolucionara junto a esa nueva versión, podría suponer un problema a la hora de ejercer la comunicación en la aplicación, así como peligros para la seguridad de los usuarios.

3.3 Requisitos

3.3.1. Requisitos funcionales

Sobre la elección del grado de consciencia activo

Dado que el sistema tiene en cuenta el modo o grado de consciencia activo para la recepción de notificaciones, se permite escoger al usuario cuál usar a libre albedrío. Para realizar la elección del modo, se precisa de:

- Grados de consciencia formados por:
  - **Nombre**. Nombre del modo activo.
  - **Mecanismo de notificación**. Cada grado de consciencia utiliza unos mecanismos diferentes para notificar al usuario las peticiones recibidas.
- El usuario puede elegir que modo activar, hallándose uno en acción cada vez. Según el escogido, el sistema da unas respuestas diferentes:
  - Si es un modo diferente al ya activo, este último será apagado y el nuevo se encenderá.
  - En caso de que se intente activar un grado de consciencia en pleno uso, se avisará al usuario de que ese modo ya está siendo utilizado.

Sobre la recepción de peticiones

En este apartado se va a exponer los requisitos que deben ser cumplidos a la hora de notificar al usuario las peticiones entrantes.

- El sistema ha de notificar al usuario de las peticiones recibidas a través de notificaciones. Las peticiones y notificaciones están compuestas por los mismos atributos, que son los siguientes
  - **Identificador**. Número que identifica al objeto.
  - **Timestamp**. Momento en que se manda la petición.

  - **Init-param**.Define el tipo de notificación. Puede ser *notify*, *choose* o *ask*.
  - **Msg**.Contenido y detalles de la petición.
  - **Options**.Para el tipo de notificación *choose*, se da unas posibles respuestas a elegir por el usuario.
  - **Urgency**.Indica si la petición recibida es una urgencia.

- Además, la aplicación debe tener en cuenta el grado de consciencia activo y el tipo de petición para utilizar unos mecanismos de notificación específicos. Esos mecanismos son los que vienen a continuación:

  - Lectura por voz.
  - Vibración.
  - Tonos.
  - Aparición de símbolo en la barra de herramientas del dispositivo.
  - Alertas.
  - Uso de un *toast*.

Sobre el envío de respuestas

En este punto se procede a la descripción de los requisitos que se deben cumplir al recabar las respuestas de los usuarios y su consecuente envío.

- El sistema ha de recoger la información que proporcione el usuario como respuesta a las notificaciones y, posteriormente, enviarla. Para ello, se utilizan las alertas, que recabarán información de tres formas:

  - A través de la escritura de texto.
  - Mediante la elección de una opción entre un grupo de posibles.
  - A través de una confirmación.
  - Además, la aplicación debe permitir al usuario ignorar las peticiones o cancelarlas.

- Esa información formará un objeto de tipo respuesta que estará formado por los siguientes parámetros:

  - **Identificador**. Número que identifica al objeto.
  - **Timestamp**. Momento en que se manda la petición.
  - **Request-id**. Identificador de la petición respondida.
  - **Msg**.Respuesta dada.

3.3.2. Requisitos no funcionales

Las condiciones no funcionales del sistemas son las siguientes:

- El conjunto de contenedores virtuales desplegados en servidores usando la tecnología *Docker*[16] con imágenes que contengan el *software mosquitto*[17] para la apertura de puertos al protocolo de mensajería MQTT.

3.3.3. Interfaces de hardware

- La aplicación ha de ser ejecutada en componentes inteligentes con sistema operativo Android.

3.3.4. Interfaces de software

La máquina que posea la función de nodo central deberá tener instaladas las dependencias que siguen:

- *Docker Community Edition*, junto al siguiente contenedor virtual:
  - *Toke/docker-mosquitto*[18] con *mosquitto* instalado

Para la ejecución de la aplicación será indispensable Android con las siguientes librerías:

- Eclipse.paho.client.mqttv3:1.0.2, para usar la función de cliente en MQTT.
- Eclipse.paho.android.service:1.0.2 para activar el servicio de MQTT en Android.
- En cuanto a los componentes utilizados en interfaz gráfica:
  - android.support.v4.app.Fragment
  - android.support.v7.preference
  - android.app.notification
  - android.support.v7.app.AlertDialog[19]
  - android.support.v7.app.AlertDialog.Builder[20]
  - android.support.v7.widget.Toolbar[21]
  - android.support.design.widget.TabLayout.Tab[22]

CAPÍTULO 4
Análisis del problema

En este capítulo se van a exponer los resultados de la fase de análisis del proyecto, de forma que se expondrá el funcionamiento del sistema mediante representación con diagramas UML (Unified Modeling Language).

4.1 Diagramas de casos de uso

4.1.1. Actores del sistema

Actores encontrados en la aplicación:

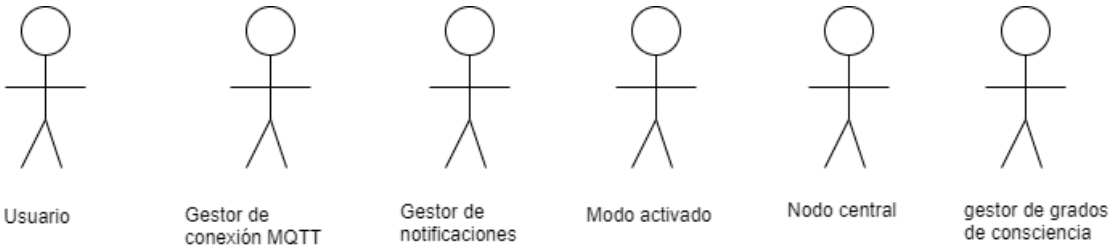

Figura 4.1: Actores del sistema

- **Usuario**: Recibe las notificaciones a través de la aplicación, puede visualizarlas y responder a dichas notificaciones.
- **Gestor de notificaciones**: Se encarga de recibir las peticiones a través de la conexión MQTT y, posteriormente, enviar las respuestas realizadas por el usuario.
- **Gestor de grados de consciencia**: Se encarga de gestionar que modo(grado de consciencia) está activo.
- **Modo abierto**: Grado de consciencia activo por el usuario.

Aparte, en el servidor encontramos el siguiente actor:

- **Nodo central**: Es el encargado de manejar las colas abiertas en sus conexiones MQTT, así como del envío de mensajes de los publicadores a los suscriptores.

4.1.2. Diagrama de contexto

El diagrama de contexto del proyecto es:

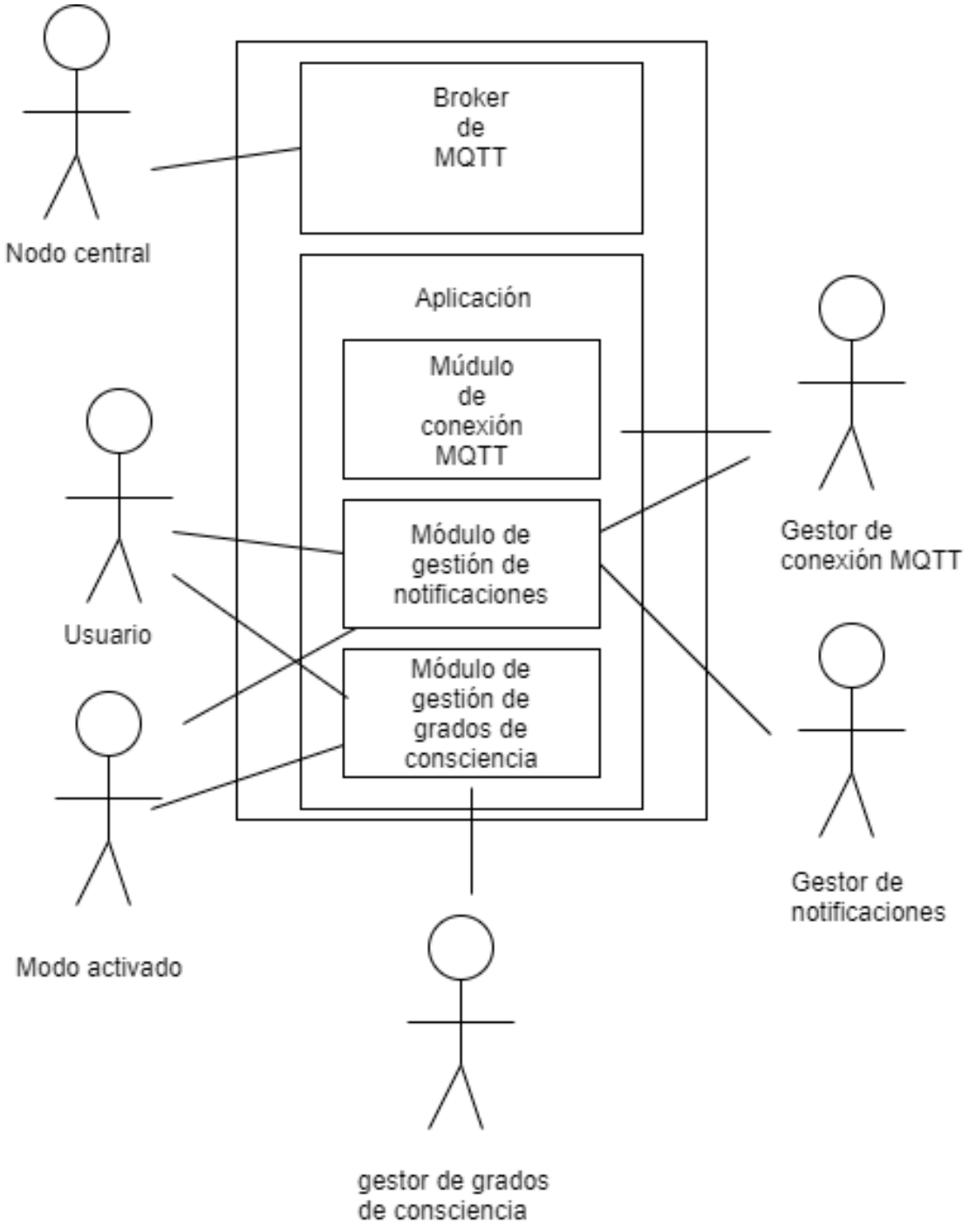

Figura 4.2: Diagrama de contexto

4.1.3. Diagrama de caso de uso inicial

Los diagramas de caso de uso inicial son los siguientes:

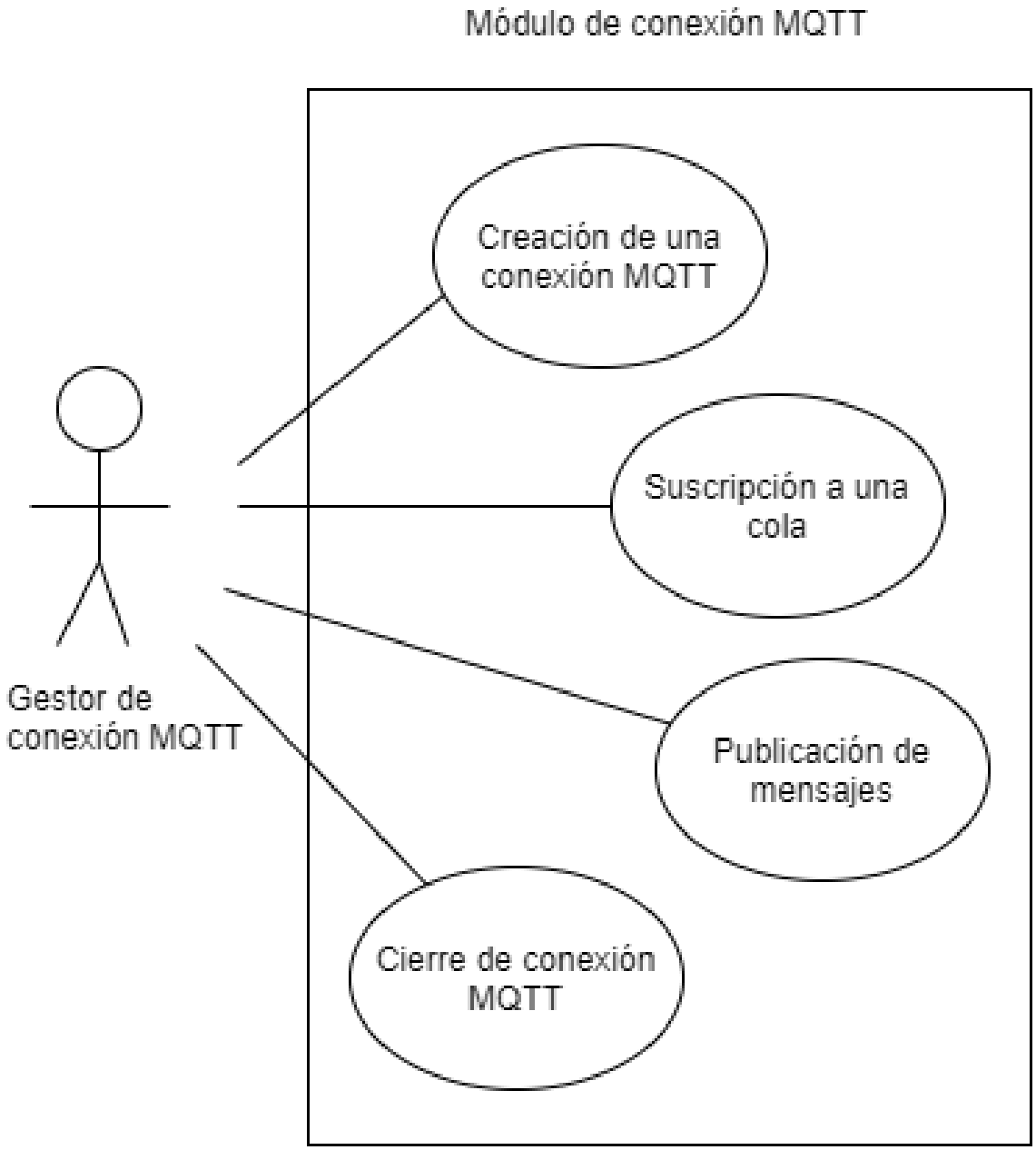

Figura 4.3: Diagrama de caso de uso inicial. Módulo de conexión MQTT.

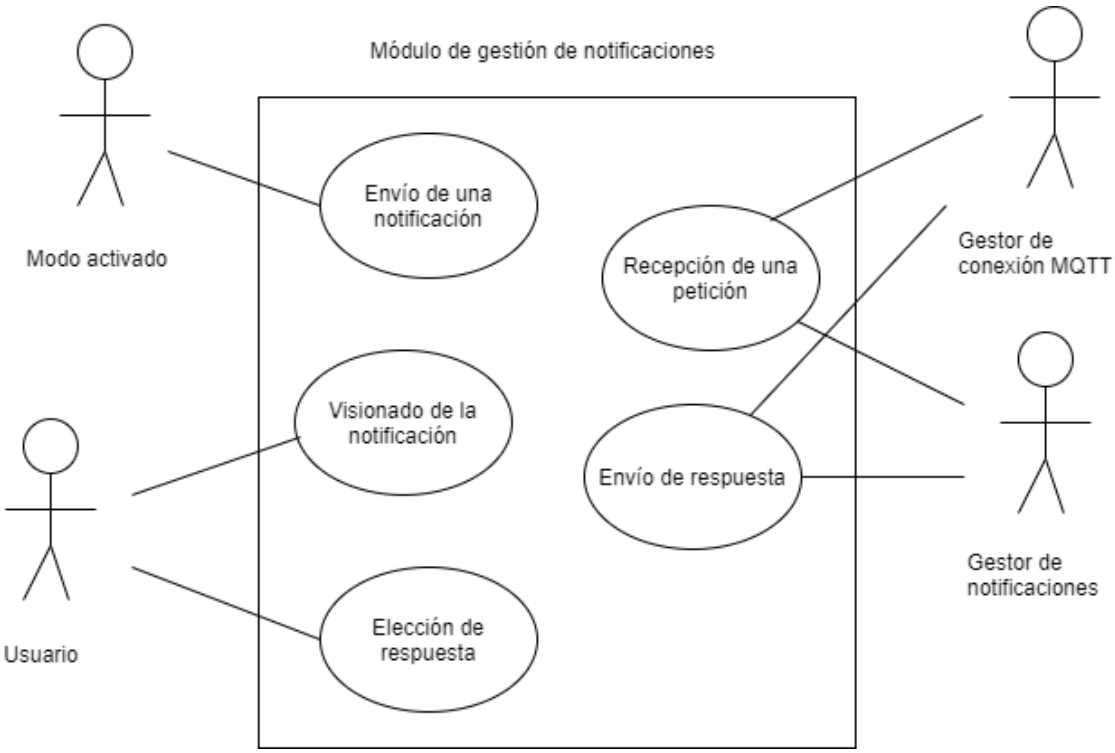

Figura 4.4: Diagrama de caso de uso inicial. Módulo de gestión de notificaciones.

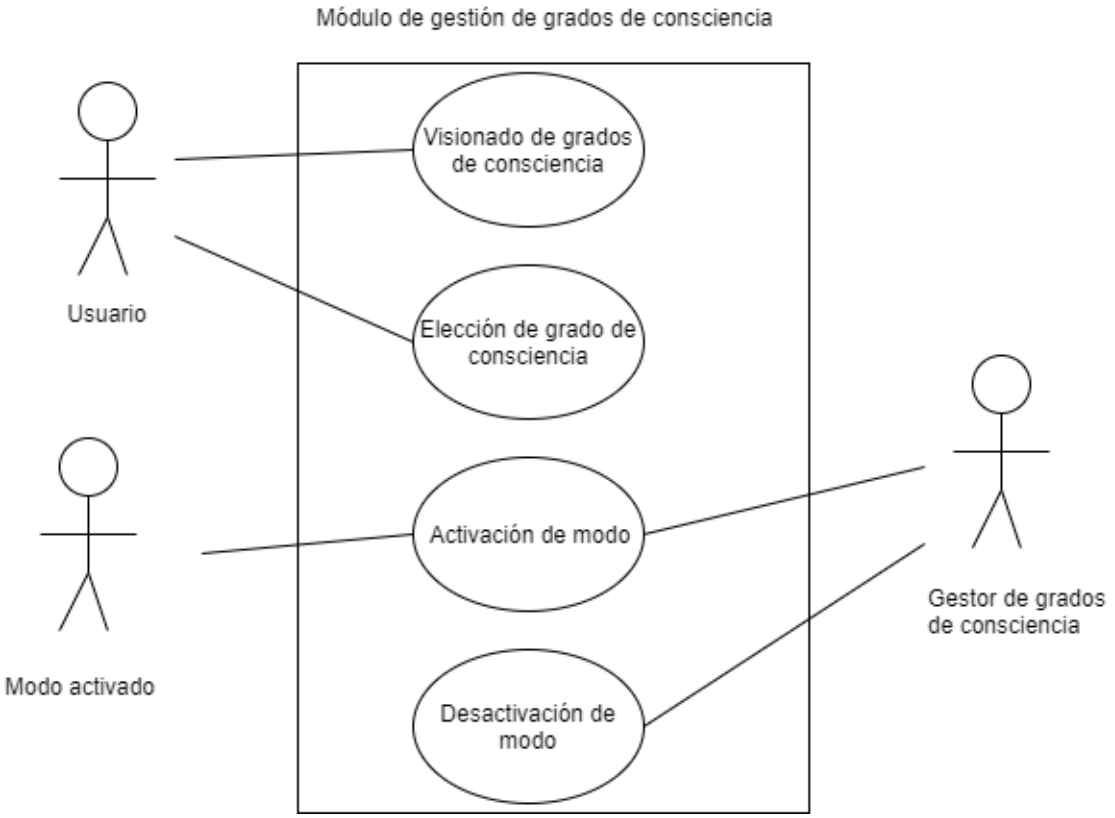

Figura 4.5: Diagrama de caso de uso inicial. Módulo de gestión de grados de consciencia.

4.1.4. Diagrama de casos de uso estructurado

Los diagramas de caso de uso estructurado de la aplicación son:

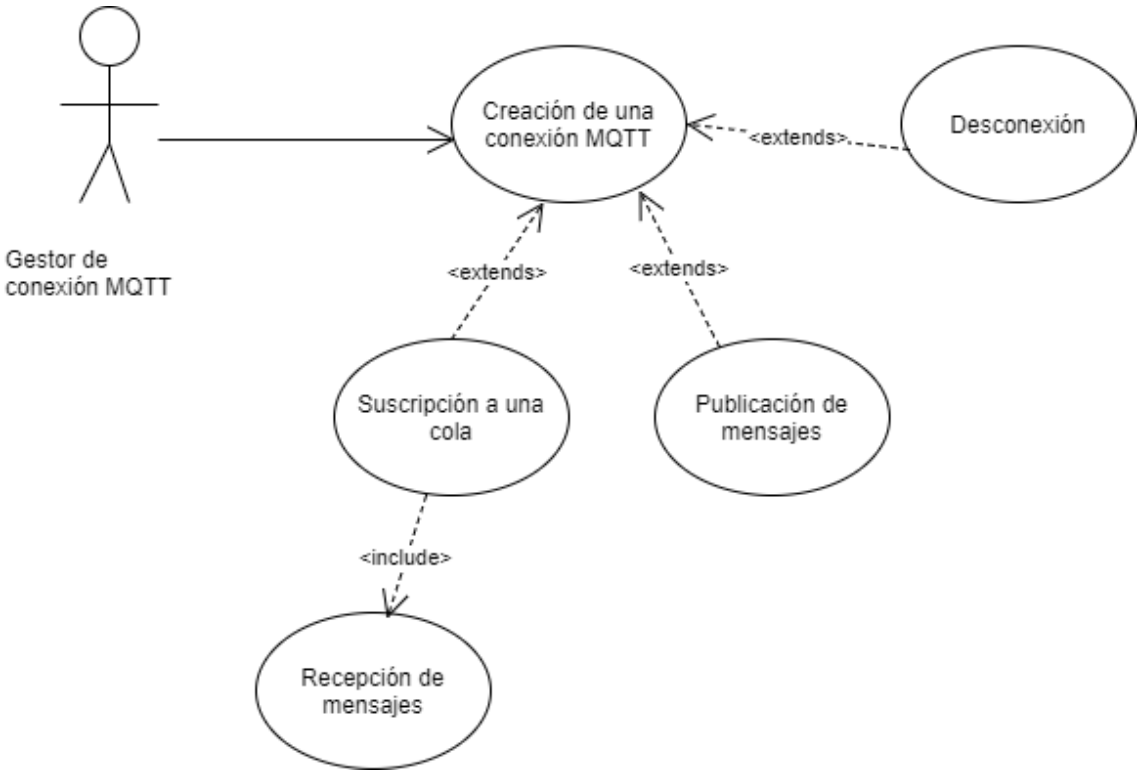

Figura 4.6: Diagrama de caso de uso estructurado. Módulo de gestión de conexión MQTT.

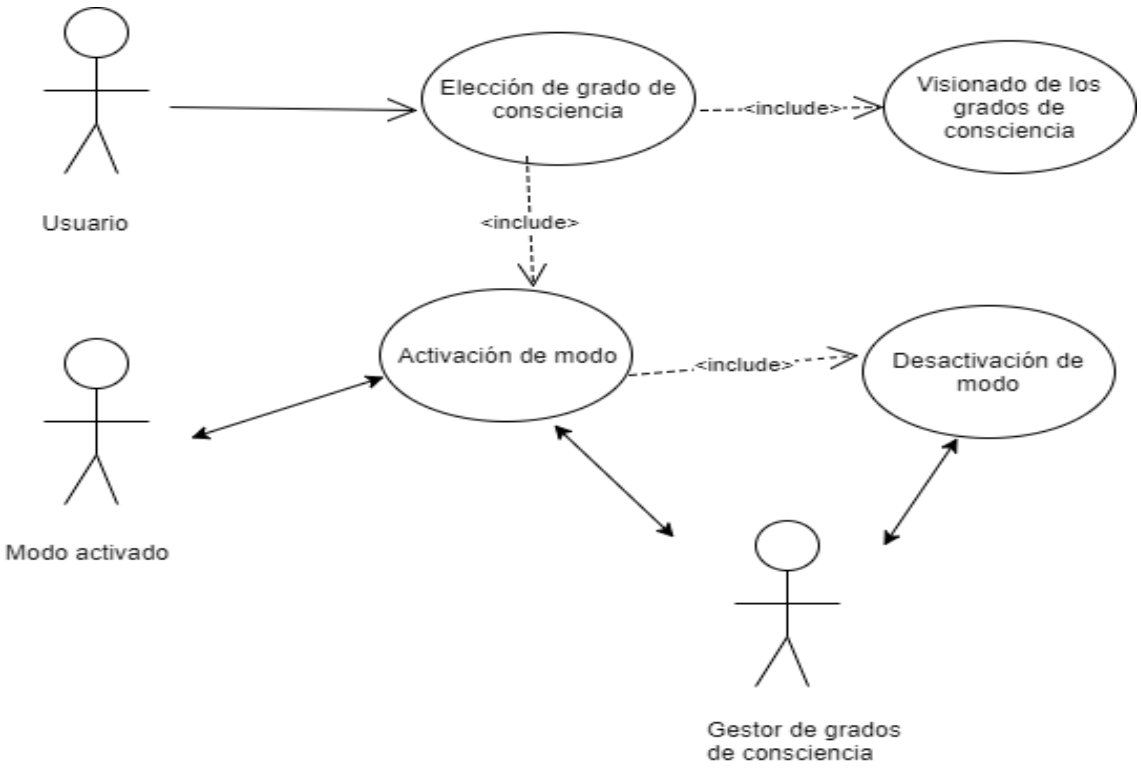

Figura 4.7: Diagrama de caso de uso estructurado. Módulo de gestión de grados de consciencia.

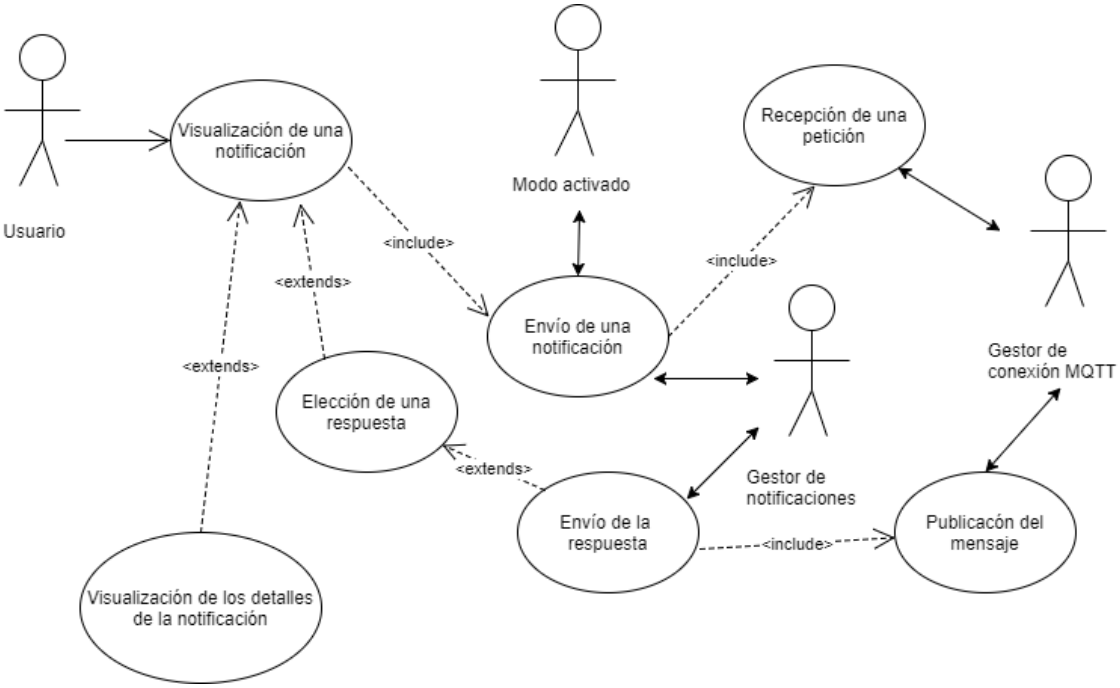

Figura 4.8: Diagrama de caso de uso estructurado. Módulo de gestión de notificaciones.

4.2 Diagramas de clases

A continuación, se presenta el diagrama de clases:

Como se puede ver en el diagrama, *ManagerService* puede recibir un número ilimitado de peticiones (que le llegan al usuario en forma de notificaciones), en el que cada petición puede recibir una respuesta. Además, gestiona los tres modos que representan los grados de consciencia (*AwareMode*, *SlightlyNoticeableMode* e *InvisibleMode*) que envían dichas peticiones con sus mecanismos de notificación correspondientes.

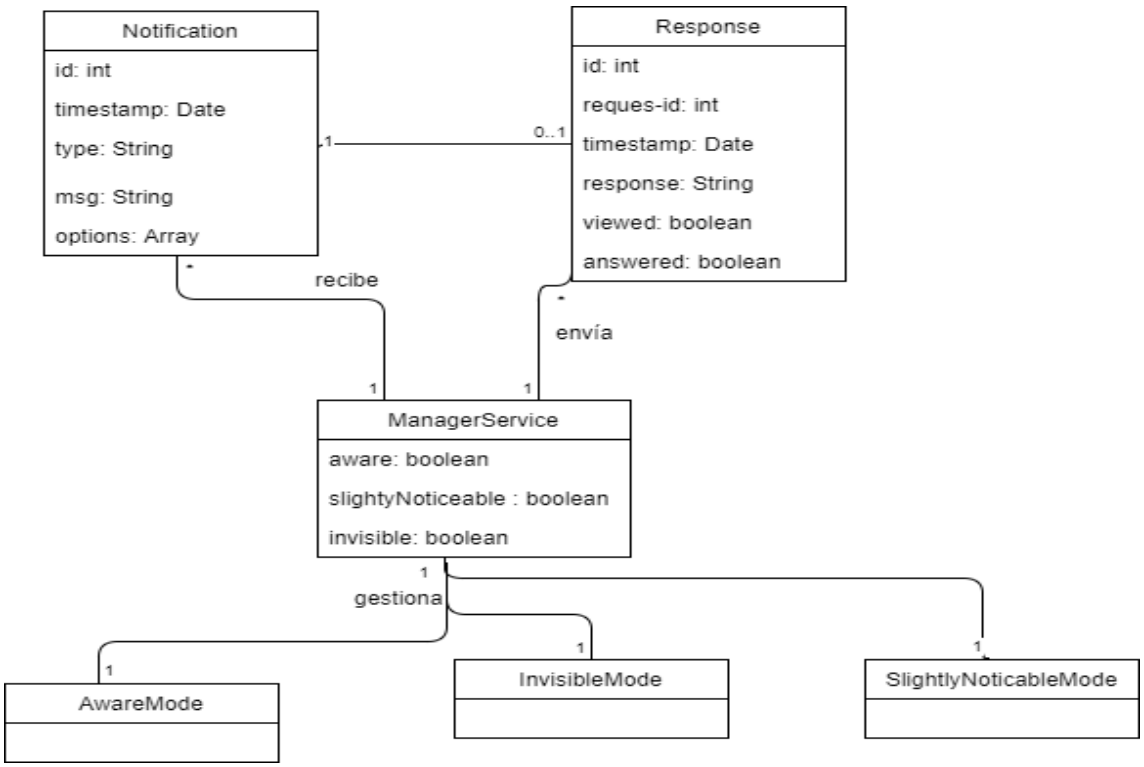

Figura 4.9: Diagrama de caso de clases.

CAPÍTULO 5
Diseño de la solución

A continuación, se procede a la explicación de los distintos módulos desarrollados, así como del diseño de las comunicaciones del proyecto y del diseño de las alertas utilizadas por la aplicación.

5.1 Modularización

Como ya se ha descrito a través en el capítulo interior, se han definido tres módulos dentro de la aplicación, siendo los siguientes:

- **Módulo de conexión MQTT:** Esta parte del proyecto se encarga de manejar la conexión con el nodo central de la conexión MQTT, suscribiéndose en la cola de recepción para recibir peticiones y publicando las respuestas en su respectiva cola.
- **Módulo de gestión de grados de consciencia:** Es el módulo que se encarga de visualizar que modo está activo y enviar las peticiones recibidas a dicho modo.
- **Módulo de gestión de notificaciones:** Es el encargado de mandar las notificaciones al usuario teniendo en cuenta el tipo de petición y el modo abierto en ese momento, además de recoger la posible respuesta que pueda dar el usuario.

5.2 Diseño de comunicaciones

5.2.1. MQTT

Como ya se ha mencionado en el punto (), en la aplicación desarrollada para este trabajo se ha utilizado para la comunicación entre los distintos componentes inteligentes el estándar MQTT. MQTT o *Message Queue Telemetry Transport* es un protocolo de mensajería orientado a la *conectividad Machine-To-Machine* (M2M). Sus principales características son su poca utilización de ancho de banda, lo que lleva a un mínimo consumo y a un menor gasto de recursos.

Este estándar está basado en la publicación y suscripción de colas o temas, donde los publicadores envían un mensaje a través de un tema que es recibido por todos los que estén suscritos a dicha cola. Para poder gestionarlo, MQTT sigue una topología en estrella, es decir, un nodo central es el que se encarga de gestionar la red y transmitir los mensajes enviados por los clientes.

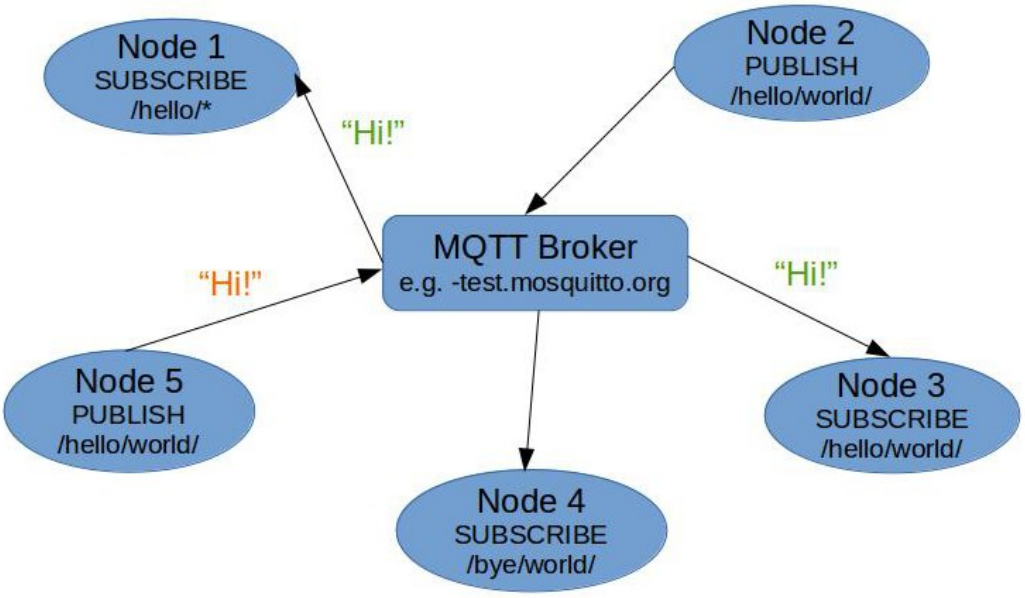

Figura 5.1: Ejemplo de conexión MQTT.

5.2.2. Nodo central

Siguiendo la arquitectura del protocolo, para el proyecto se ha escogido utilizar como software del nodo central al *broker mosquitto*. Este software de código abierto implementa las versiones 3.1 y 3.1.1 de MQTT. Destaca por su ligereza y por su capacidad de ser aplicado en un gran número de situaciones diferentes.

Para su aplicación, se puede optar por dos opciones:

- Instalarlo en una máquina física que hará directamente de nodo central.
- Usando un contenedor virtual con el *software mosquitto* ya instalado a través de *Docker*.

En este trabajo se ha escogido la última opción. Los factores decisivos de preferir trabajar con contenedores virtuales en *Docker* han sido su flexibilidad y su gran portabilidad a otros entornos, así como la escabilidad que tiene debida a su nula sobrecarga de arranque en comparación con las máquinas virtuales.

5.2.3. Colas

En cuanto a los temas utilizados, se han diferenciado las colas donde los vehículos publicarán las respuestas de los usuarios y las colas en las que recibirán las peticiones de otros componentes. Siguiendo esta explicación, las colas diseñadas son las que aparecen a continuación:

- Para la recepción de peticiones, la cola es *iot/request*.
- Para la publicación de respuestas de mensajes, la cola es *iot/response*.

Cabe destacar que, en un entorno real, esta aplicación usaría mas tipos de colas diferentes definidas para distintos servicios o situaciones (servicio sanitario, administración pública, etc.), tanto para respuesta como para recepción. Incluso se podrían crear colas que fueran definidas para comunicación entre diferentes localidades (en situaciones de emergencias nacionales, como pandemias). Sin embargo, para este trabajo de final de grado, se han especificado estas dos colas por comodidad de trabajo.

5.3 Diseño de interfaces de usuario

5.3.1. Diseño de las interfaces de las alertas

En lo referente al diseño de la forma en la que los usuarios podrán responder a las notificaciones recibidas que lo requieran, se ha optado por la utilización de alertas. Este tipo de fragmento en Android, tiene un diseño muy usable y fácil de entender para el usuario y que, además, conlleva un mínimo consumo de recursos por el hecho de ser un fragmento. Según el tipo de notificación recibida, se han creado tres tipos de alertas: alerta con texto editable, alerta con botones de radio y alerta de confirmación.

Alerta de texto editable

Este tipo de alerta tiene inflado en su interfaz un *EditText*[23] que permite al usuario escribir la respuesta a la notificación recibida, como se puede ver en la siguiente imagen:

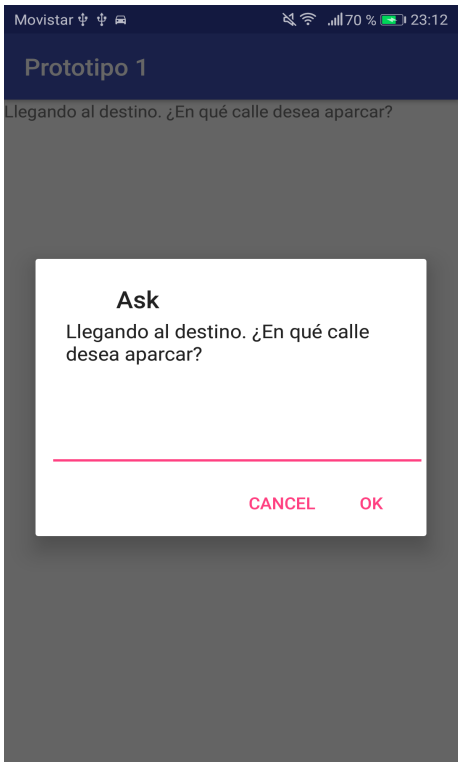

Figura 5.2: Diseño de alerta de texto editable.

Alerta de botón de radio

El modelo de alerta que se describe en este punto contiene en su interfaz un *RadioGroup*[25] compuesto por *RadioButton*[24] que permite al usuario elegir la opción que prefiera entre las recibidas por la aplicación. A continuación, se procede a la muestra de una imagen del diseño:

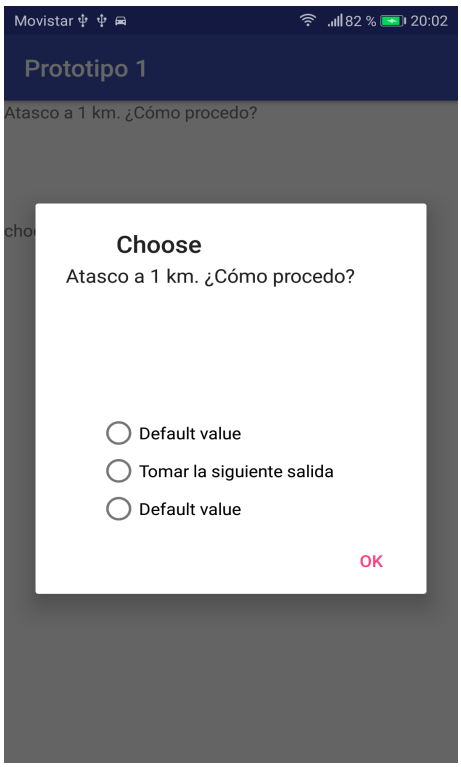

Figura 5.3: Diseño de alerta de botón de radio o elección.

Alerta de confirmación

Esta clase de alerta está compuesta por un mensaje y por dos botones de confirmación y cancelación, respectivamente. Este tipo es utilizada en casos de recepción de aviso de urgencia o una notificación informativa que requiera la visualización del usuario:

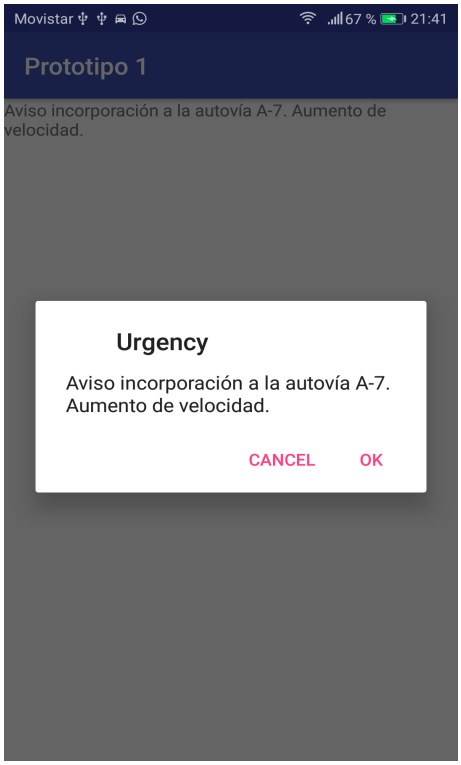

Figura 5.4: Diseño de alerta de confirmación.

En el punto 6.3.3, se hace una descripción más detallada de todos los componentes gráficos que estructuran el diseño de las alertas.

5.4 Estructura de las peticiones y las respuestas

Las peticiones y las respuestas son partes básicas de este proyecto. Las primeras son convertidas en notificaciones para que puedan llegar al usuario y las segundas representan la contestación que da el usuario a las peticiones.

Para su transmisión, se ha decidido utilizar el formato *JSON*, por su facilidad de lectura y escritura. Se han definido dos estructuras:

Para las peticiones, se han especificado estos atributos:

- **id:** Identificador de la petición.
- **timestamp:** Valor que indica el momento en que se ha emitido la petición.
- **init-param:** Tipo de petición. Los valores posibles son *notify*, *choose* y *ask*.
- **msg:** Mensaje de la petición.
- **options:** *Array* que contiene las opciones de respuesta. Solo aplicable cuando el tipo de petición es *choose*.
- **urgency:** Valor que indica si es una urgencia.

A continuación, se muestra un ejemplo de una petición en formato JSON:

```
"request":{
    "id": 1,
    "timestamp": "06/07/2017 12:18:34",
    "init-param": "choose",
    "msg": "¿Necesita asistencia tras el accidente?",
    "options": ["Pedir asistencia sanitaria", "Pedir asistencia policial", "Pedir ambas"],
    "urgency": true
}
```

Figura 5.5: Petición en formato JSON.

Para las respuestas, se han especificado estos atributos:

- **id:** Identificador de la petición.
- **timestamp:** Valor que indica el momento en que se ha respondido a la petición.
- **request-id:** Identificador de su petición.
- **response:** valor de la respuesta.

Se va a proceder a mostrar un ejemplo:

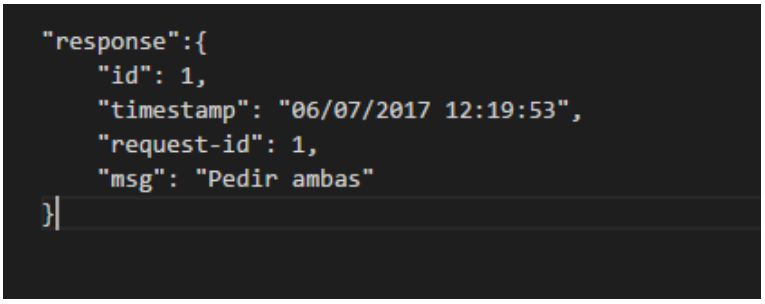

Figura 5.6: Respuesta en formato JSON.

CAPÍTULO 6
Implementación

En este capítulo, se procede a la descripción de la arquitectura utilizada en el desarrollo, así como de la implementación de las distintas capas utilizadas.

6.1 Arquitecturas usadas

6.1.1. Arquitectura de tres capas

La programación en capas es un modelo de desarrollo de software que se centra en la separación de las partes que conforman un sistema. El principal beneficio de esta arquitectura es que la actualización o cambio de una de las capas no supone una remodelación en el resto. Generalmente, se divide en tres capas: la capa de persistencia, la capa de lógica de negocio y la capa de presentación.

La capa de persistencia es aquella donde se guardan los datos y es ella misma la que accede a ellos. Está formada por gestores de bases de datos que realizan todo el almacenamiento y recuperaciones solicitados por la capa de lógica de negocio.

La capa de lógica de negocio es la que contiene todos los programas que se ejecutan. Recibe las peticiones de usuario y envía las respuestas tras el proceso. Esta capa se comunica con la de presentación donde presenta los datos al usuario y, con la de persistencia, donde realiza peticiones para almacenar y recuperar información.

La capa de presentación es la que ve el cliente ya que la componen las distintas interfaces gráficas. Recaba información del usuario y se la presenta.

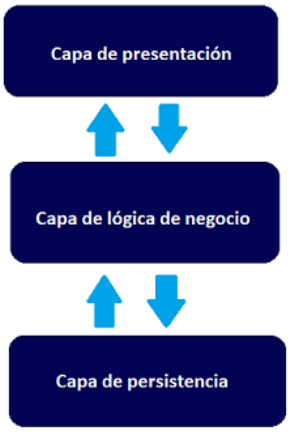

Figura 6.1: Arquitectura de tres capas.

6.2 Componentes desarrollados

En esta sección se van a describir los componentes más importantes desarrollados en el sistema.

6.2.1. Servicios

Se ha utilizado servicios tanto para la implementación de los distintos grados de consciencia definidos en la aplicación y para el desarrollo del servicio que se encarga de gestionar dicho grados, la recepción de peticiones y el envío de respuestas.

Ese servicio gestor, ***ManagerService***, se encargará de saber que grado de consciencia o modo está activo en cada momento y, por lo tanto, de enviarle la petición que él recibe a través de la conexión MQTT, para que este último la notifique al usuario. Además, se encargará de enviar las respuestas a esas peticiones a través de la cola especificada para ello.

En cuanto a los grados de consciencia del usuario, se han definido tres diferentes con un servicio cada uno. Estos tres modos son: *Aware*, definido para cuando el usuario necesita ser completamente consciente de lo todas las notificaciones que pueda recibir la aplicación sin ninguna interferencia, *Slightly Noticeable*, creado para cuando el usuario solo necesita prestar atención de ls interacciones plenamente importantes (puede ser en caso de que tenga alguna persona con él o esté realizando alguna actividad), e *Invisible*, implementado para cuando el usuario, por falta de visión en el vehículo o por otras condiciones, no necesita o no quiere estar atento a las notificaciones que le llegan (salvo en caso de notificación de urgencia, que ignora el grado de consciencia que esté seleccionado).

6.2.2. Actividades

En lo referente a las actividades usadas en el sistema, se han implementado tres: la actividad principal (*MainActivity*), la actividad de visualización de los detalles de las notificaciones (*DetailsActivity*) y la actividad de ajustes de la aplicación (*SettingsActivity*):

- **Actividad principal**. Esta actividad está compuesta por dos fragmentos gestionados por un sistema de pestañas: uno como pantalla inicial de la aplicación y otro que contiene el listado de notificaciones recibidas.
- **Actividad de visualización de detalles de notificación**. Esta actividad permite visualizar los detalles de cada notificación (mensaje, fecha de llegada, etc.)..
- **Actividad de ajustes**. Esta actividad (basada en *SettingsActivity* de Android) permite la opción de cambiar el grado de consciencia activo en ese momento por la aplicación, seleccionar la dirección IP de conexión MQTT (para concectarse al nodo central) y seleccionar el tono de notificación.

6.2.3. Fragmentos

En la aplicación desarrollada, se han utilizado fragmentos en diversos componentes del proyecto. En concreto, han sido usados tanto en la actividad principal del sistema como en las alertas para recabar la respuesta del usuario.

Por una parte, los fragmentos embebidos en las pestañas de la actividad principal son las siguientes:

- ***HomeTabFragment***. Fragmento pensado para la incorporación de información sobre el tiempo meteorológico y sobre la carretera que está siendo en ese momento recorrida.
- ***NotificationListTabFragment***. Fragmento que se encarga de mostrar el listado de notificaciones recibidas por el usuario, donde se puede seleccionar de que notificación se desea ver en detalle.

Por otra parte, los fragmentos usados en las alertas (ya descrito su diseño en el punto 5.3) son:

- ***AlertFragment***. Fragmento implementado para las alertas de confirmación..
- ***RadioGroupAlertFragment***. Fragmento creado para las alertas de elección de opciones.
- ***EditTextAlertFragment***. Fragmento pensado para permitir al usuario la escritura de una respuesta.

6.3 Capas implementadas

En este apartado se van a definir las capas implementadas en el proyecto, siguiendo la arquitectura de tres capas: capa de persistencia, capa de lógica de negocio y capa de presentación.

6.3.1. Capa de persistencia

Basándonos en el diagrama de clases del proyecto (figura **??**), se ha definido un modelo de datos relacional en el lado del cliente. La base de datos tiene la función de guardar todas ls peticiones recibidas y respuestas mandadas en sus tablas correspondientes. Para la implementación de este modelo, se ha utilizado el motor de bases de datos nativo de Android: SQLite.

Restricciones

Antes de proceder a la descripción de las tablas, se van a comentar las restricciones que se van a usar en ellas:

- **Clave primaria (PK)**: Restricción usada para indicar el identificador de cada elemento de una tabla. A su vez, contiene estas dos restricciones:
  - **No nulo**: Restricción utilizada para indicar que el campo en cuestión no puede tener valor nulo.
  - **Unicidad**: Restricción que especifica que un cada valor del campo al que es aplicado no puede ser repetido.

Tabla Requests

En esta tabla, se van a insertar todas las peticiones recibidas a través de las colas gestionadas por el nodo central.

Campo	Tipo	Valor
Id	INTEGER	Identificador de cada petición
Timestamp	TIMESTAMP	Fecha de envío de petición
Tipo	TEXT	Tipo de la petición
Msg	TEXT	Mensaje de la petición
Urgency	INTEGER	Valor que indica si es una urgencia

Tabla 6.1: Tabla *Request*.

Tabla Responses

En esta tabla, se van a insertar todas las respuestas del usuario a sus correspondientes peticiones.

Campo	Tipo	Valor
Id	INTEGER	Identificador de cada petición
Id_peticion	INTEGER	Identificador de la petición asociada a la respuesta
Timestamp	TIMESTAMP	Fecha de envío de la respuesta
Response	TEXT	Respuesta a la petición

Tabla 6.2: Tabla *Response*.

6.3.2. Capa de lógica de negocio

Siguiendo la arquitectura de tres capas, a continuación se va a explicar la capa de lógica de negocio de este proyecto.

Recepción de notificaciones

Una vez el modo activado (servicio) obtiene la petición, extrae todos sus atributos, teniendo especial relevancia el parámetro *init-param*, que define el tipo de petición, y el parámetro *msg*, que especifica el mensaje (en la imagen 6.2, se muestra como se analiza esta parte en el código). Por lo tanto, teniendo en cuenta el modo activo, se construye la notificación de la siguiente manera:

```
JSONObject jsonObject = new JSONObject(a);
int requestid = jsonObject.getInt("request-id");
String type = jsonObject.getString("init-param");
String msg = jsonObject.getString("msg");
String timestamp = jsonObject.getString("timestamp");
Boolean urgency = jsonObject.getBoolean("urgency");
JSONArray options;
String[] optionsString;
```

Figura 6.2: Extracción de atributos de la petición.

- Si el servicio activo es *AwareMode*, se utiliza un *TextSpeaker* para que sintetice a voz el mensaje de la notificación, se ejecuta el método *displayNotification()*, donde manda una notificación al móvil, utilizando mecanismos como vibración y un tono.
- Si el servicio activo es *SlightlyNoticeableMode*, se ejecuta el método *displayNotification()*, donde manda una notificación al móvil, utilizando mecanismos como vibración y un tono.
- Si el servicio activo es *InvisibleMode*, se utiliza un *TextSpeaker* para que sintetice a voz el mensaje de la notificación, se ejecuta el método *displayNotification()*, donde manda una notificación al móvil, utilizando mecanismos como vibración.

Aparte, según el tipo de petición, el tipo de alerta usado para que el usuario pueda visualizar el contenido de la notificación y pueda dar una respuesta es diferente. Así pues, atendiendo al parámetro *ïnit-param"*:

- Si su valor es *notify*, se lanzará la alerta de confirmación.

```
case "notify":
    t1.speak(msg, TextToSpeech.QUEUE_FLUSH, null, "Notify");
    displayNotification("Notify", msg);
    Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_LONG).show();
    saveState(type, msg, requestid, null);
    Intent dialogIntent = new Intent(getApplicationContext(), DetailsActivity.class);
    dialogIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    startActivity(dialogIntent);

    insertNotificationinDB(timestamp, msg, urgency, requestid, type);

    break;
```

Figura 6.3: Código de la aplicación en modo *Aware* para tipo de notificación *notify*.

- Si su valor es *choose*, se lanzará la alerta de elección.

```
case "choose":
    options = jsonObject.getJSONArray("options");
    optionsString = new String[4];

    for (int i = 0; i <= options.length() - 1; i++) {
        optionsString[i] = options.getString(i);
    }

    t1.speak(msg, TextToSpeech.QUEUE_FLUSH, null, "Choose");
    displayNotification("Choose", msg);
    Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_LONG).show();
    saveState(type, msg, requestid, optionsString);

    insertNotificationinDB(timestamp, msg, urgency, requestid, type);

    break;
```

Figura 6.4: Código de la aplicación en modo *Aware* para tipo de notificación *choose*.

- Si su valor es *ask*, se lanzará la alerta de texto editable.

```
case "ask":
    tl.speak(msg, TextToSpeech.QUEUE_FLUSH, null, "Indication");
    Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_LONG).show();
    saveState(type, msg, requestid, null);
    content = msg;
    title = "Ask";
    displayNotification(title, content);
    insertNotificationinDB(timestamp, msg, urgency, requestid, type);

    break;
```

Figura 6.5: Código de la aplicación en modo *Aware* para tipo de notificación *ask*.

Además, en caso de que en la petición el parámetro *ürgency"* tenga valor *true*, se lanzará la alerta correcta según el tipo de notificación y se utilizarán los mecanismos de vibración, uso de tono, lectura por voz y, en los dispositivos que lo soporten, luz LED, todo para que el usuario sea consciente de la llegada de una alerta. Este tipo especial de notificación es aplicado a todos los grados de consciencia.

Posteriormente, todas las peticiones recibidas son insertadas en la tabla *Request* (tabla 6.1) ejecutando el método *insertRequestInDB()*.

Obtención de respuestas

Como ya se ha descrito anteriormente, el usuario puede responder a las notificaciones recibidas a través de la alertas. Siguiendo esta explicación, los tres tipos de alerta, implementadas a utilizando fragmentos, están embebidas en la actividad *DetailsActivity*, que contiene los detalles de la notificación (se ha desarrollado de esta forma para que el usuario pueda ver más detalles de la notificación tras responderla). Todas las alertas muestran el contenido recibido por la petición como el tipo de notificación, el mensaje y dos botones, de confirmación y cancelación, respectivamente. Sin embargo, dependiendo de ese tipo, la respuesta se construirá de una forma diferente:

- **Alerta de confirmación:** Si el usuario presiona el botón de confirmar, se construirá un objeto *Response*, que será formateado a *JSON* y enviado al gestor de conexión MQTT para su publicación.
- **Alerta de elección**: Esta alerta, que contiene como posibles respuestas las opciones dadas por la petición, permitirá la selección de una de ellas y, en caso de confirmar el envío de la respuesta, generará un objeto *Response* y la enviará al gestor de conexión MQTT.
- **Alerta de texto editable**: Además del mensaje, esta alerta contiene un objeto *EditText* que permite la escritura de la respuesta al usuario. Como en el resto de alertas, en caso de confirmar el envío de la respuesta, generará un objeto *Response* y la enviará al gestor de conexión MQTT.

Cabe destacar que el usuario puede no responder a las notificaciones recibidas, pulsando sobre el botón de cancelación. En ese caso, no se publicará ninguna respuesta a través del protocolo MQTT.

Una vez creada la respuesta, se llama al método *insertResponseInDB()* para su inserción en la tabla *Response* (tabla 6.2).

Gestión de los ajustes y los grados de consciencia

Se ha utilizado la clase *Settings* de Android para la generación de los ajustes de la aplicación. En ellos, se puede cambiar el tono de las notificaciones, así como seleccionar que modo activar y determinar la dirección IP del nodo central (implementado así para facilitar las pruebas).

Para detectar los posibles cambios en sus valores, como se puede ver en la imagen 6.6, se ha utilizado un *SharedPreferencesChangedListener*, que monitoriza constantemente los posibles cambios que puedan producirse en ellos. En caso de visualizar uno, este *listener* obtiene la clave (campo que identifica el ajuste cambiado) y, según su valor, procede de la siguiente forma:

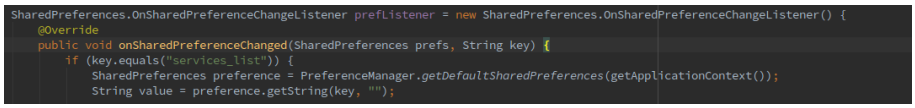

Figura 6.6: Extracción de atributos de la petición.

- Si la clave es igual a *"services"*, obtiene el valor referente al nuevo modo seleccionado y, consultando las variables globales definidas en *ManagerService*, para el servicio anterior utilizando *stopService()* y arranca el nuevo, ejecutando el método *startService()*. Se muestra un ejemplo del código:

```
case "1":
    if (ManagerService.aware) {
        Toast.makeText(getApplicationContext(), "Aware Mode is already on", Toast.LENGTH_SHORT).show();
    } else if (ManagerService.slightlyAware) {
        Intent intent = new Intent(getApplicationContext(), SlightlyAwareMode.class);
        Intent intent2 = new Intent(getApplicationContext(), AwareMode.class);
        stopService(intent);
        ManagerService.slightlyAware = false;
        startService(intent2);
        ManagerService.aware = true;
    } else if (ManagerService.invisible) {
        Intent intent = new Intent(getApplicationContext(), InvisibleMode.class);
        Intent intent2 = new Intent(getApplicationContext(), AwareMode.class);
        stopService(intent);
        ManagerService.invisible = false;
        startService(intent2);
        ManagerService.aware = true;
    } else {
        Intent intent2 = new Intent(getApplicationContext(), AwareMode.class);
        startService(intent2);
        ManagerService.aware = true;
    }
    break;
```

Figura 6.7: Inicialización del modo *Aware* usando preferencias.

- Si la clave es igual a *ïp"*, arranca el gestor de conexión MQTT, que posteriormente obtendrá su valor para crear la conexión (se explicará en el siguiente apartado).

Para el tono de las notificaciones, no se comprueba ya que solo es necesario para la construcción de las notificaciones para el grado de consciencia *Aware* y el valor se cambia automáticamente.

Conexión a MQTT

Como se describe en el apartado anterior, una vez se detecta un cambio en el valor de la preferencia asignada a la dirección IP del servidor, se arranca el gestor de conexión

MQTT. Esta parte de la aplicación es la que ha sido desarrollada usando la librería Eclipse Paho, aportando toda la funcionalidad necesaria para la conexión.

Cuando el gestor de conexión MQTT es iniciado, obtiene la dirección IP (a través de la preferencia *ïp"*) y se crea un objeto *MqttAndroidClient* especificando la dirección ip del nodo central, el puerto (por defecto, 1883) y el usuario. Una vez creado el objeto, se ejecuta el método *connect()* y, según el resultado del intento de conexión, puede llevar a dos situaciones:

- Si el intento de conexión falla, el método *onFailure()* es llamado, obteniendo una excepción especificando el motivo de fallo.
- Si el intento de conexión es exitoso, se inicia el método *onSuccess()* y, posteriormente, se suscribe a la cola de recepción de peticiones usando el método *subscribe()* y se arranca un *Mqttlistener*, que se queda escuchando la llegada de dichas peticiones.

```
final MqttAndroidClient mqttAndroidClient = new MqttAndroidClient(this, "tcp://" + ip, "usuario");
```

Figura 6.8: Creación de un objeto MQTT.

Si una petición es recibida, se llama al método *onMessageArrived()*, donde se recibe el mensaje enviado a través de la cola, se guarda en una preferencia compartida y, dependiendo de la variable que tenga valor *true* entre las tres que representan los tres grados de consciencia, la petición es enviada al servicio correspondiente.

```
@Override
public void messageArrived(String topic, MqttMessage message) throws Exception {
    String a = new String(message.getPayload());

    SharedPreferences _prefs = getSharedPreferences("request", Activity.MODE_PRIVATE);
    if (_prefs == null) return;

    SharedPreferences.Editor _prefsEditor = _prefs.edit();
    if (_prefsEditor == null) return;

    _prefsEditor.putString("request_mode", a);
    _prefsEditor.apply();
```

Figura 6.9: Recepción de una petición.

```
if (aware) {
    Intent intent1 = new Intent(getApplicationContext(), AwareMode.class);
    startService(intent1);
} else if (slightlyAware) {
    Intent intent1 = new Intent(getApplicationContext(), SlightlyAwareMode.class);
    startService(intent1);
} else {
    Intent intent1 = new Intent(getApplicationContext(), InvisibleMode.class);
    startService(intent1);
}
```

Figura 6.10: Envío de una petición al modo abierto.

Una vez explicado la creación de la conexión y la recepción de peticiones, se va a describir el proceso de envío de respuestas. Este procedimiento es más simple. Una vez una alerta ha obtenido la respuesta del usuario, ejecuta el método *startService()* sobre *ManagerService* (aunque el servicio ya está iniciado, este vuelve a ejecutar *onStartCommand()*),

comprueba que ha recibido la respuesta a través de un *SharedPreferences* en formato JSON. Cuando el gestor de conexión MQTT recibe la respuesta, ejecuta el método *publish()* especificando la cola donde será publicado el mensaje y la respuesta a enviar. Ahora, se expone una imagen del código relacionado con el envío de respuestas a través de MQTT.

```
@Override
public void onSuccess(IMqttToken asyncActionToken) {
    try {
        mqttAndroidClient.publish("iot/response", new MqttMessage(jsontostring.getBytes()));
        mqttAndroidClient.disconnect();
    } catch (MqttException ex) {
        ex.printStackTrace();
    }

}
```

Figura 6.11: Envío de una respuesta.

Cambio automático de modos según luminosidad

Antes de comenzar con la explicación de esta parte, es necesario mencionar que por falta de tiempo y problemas de implementación no se ha podido finalizar esta parte. Sin embargo, se va a describir porque se ha considerado una característica relevante que sería interesante terminar en un futuro.

Esta característica está definida en un servicio propio. Básicamente, se detectaba la presencia de un sensor de luminosidad utilizando los objetos *SensorManager* y *Sensor*. Una vez comprobada su existencia, se declaraba un *SensorEventListener* que, usando el método *onSensorChanged()*, detectaba todos los cambios de luminosidad y, a través de unos rangos establecidos y el valor obtenido, el modo activo se cambiaba.

6.3.3. Capa de presentación

Siguiendo el desarrollo explicado en el punto anterior, se va a proceder a una descripción del funcionamiento y diseño de la capa de presentación.

Vista principal

Nada más ejecutar la aplicación se muestra la interfaz gráfica de la actividad principal. En ella, se pueden ver dos pestañas, implementadas usando el componente gráfico de Android *TabLayout*, que tienen embebidas un fragmento cada una de ellas y la opción de ejecutar el menú de ajustes de la aplicación desde la barra de herramientas, utilizando el componente *MenuItem*, como se ha especificado en la subsección anterior. Esos dos fragmentos muestran:

- Vista inicial con información sobre la carretera que se está recorriendo en ese momento (no se ha podido implementar por falta de tiempo).
- Un listado de las notificaciones recibidas.

El usuario podrá deslizar con el dedo de forma horizontal para navegar entre la pestañas o pulsando sobre el nombre de cada una de ellas.

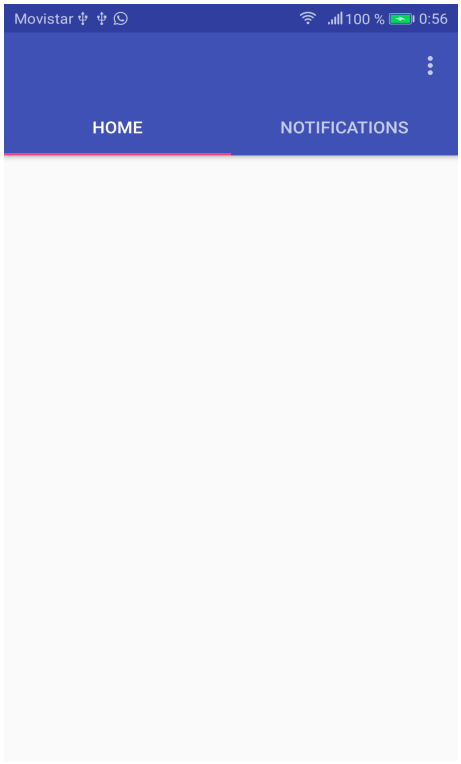

Figura 6.12: Vista principal de la aplicación.

Pestaña de consulta de notificaciones

Como se puede ver en la imagen 6.13, todas las notificaciones recibidas son insertadas en el adaptador de un *listView* para que sean mostradas en forma de listado. Además, el usuario puede seleccionar una notificación en la lista, pulsando sobre él, para poder ver los detalles de esta, como se muestra en la siguiente imagen.

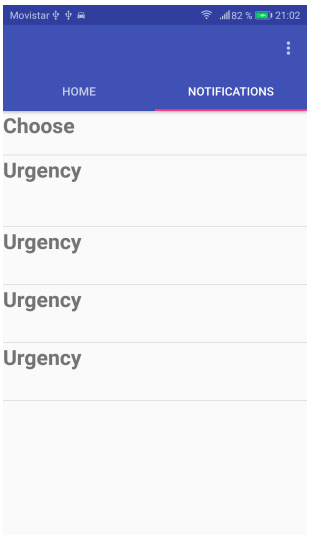

Figura 6.13: Vista de listado de notificaciones.

Vista de detalles de notificaciones

Esta vista, perteneciente a *DetailsActivity*, permite la visualización de los detalles de una notificación. A través del uso de *TextView*, se muestran los valores del mensaje, el tipo y la fecha de emisión de la petición.

Pestaña de ajustes de la aplicación

Esta vista ha sido desarrollado usando los componentes *Preferences* pertenecientes a Android, que permiten la realización de un listado de ajustes. En la imagen **??**, se puede ver como se construye una ventana de ajustes usando XML. La primera vista, muestra dos categorías: "Generalz "Notificaciones".

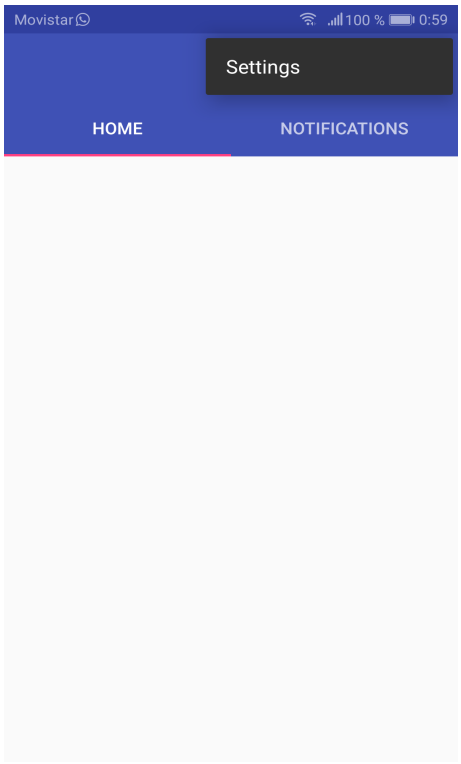

Figura 6.14: Entrada al menú de ajustes.

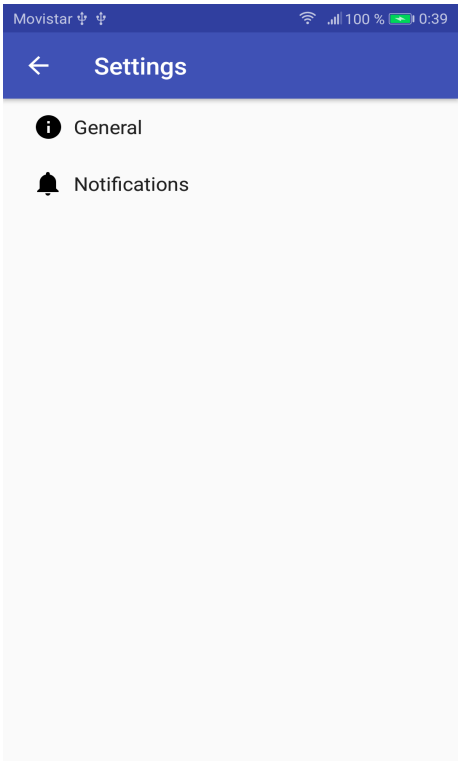

Figura 6.15: Menú de ajustes.

Dentro de "General", se pueden ver dos campos (como se puede visualizar en la siguiente imagen):

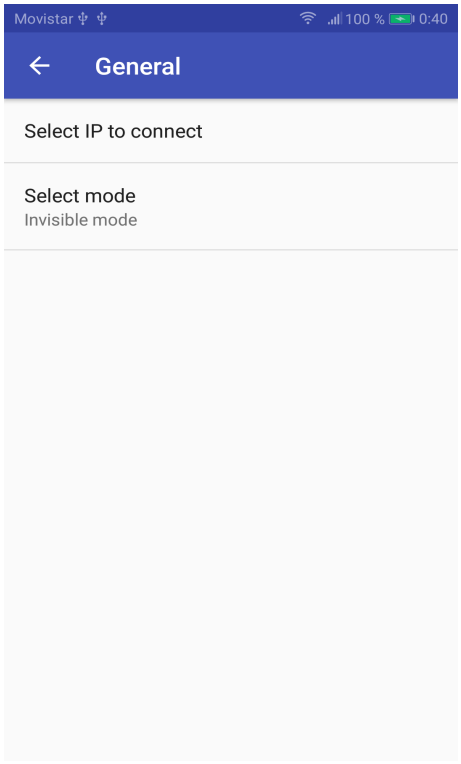

Figura 6.16: Sección "General".

El referente a la dirección IP del nodo central de la conexión MQTT, implementado a través de un *EditPreference* donde el usuario puede escribir directamente la dirección IP con el puerto de conexión:

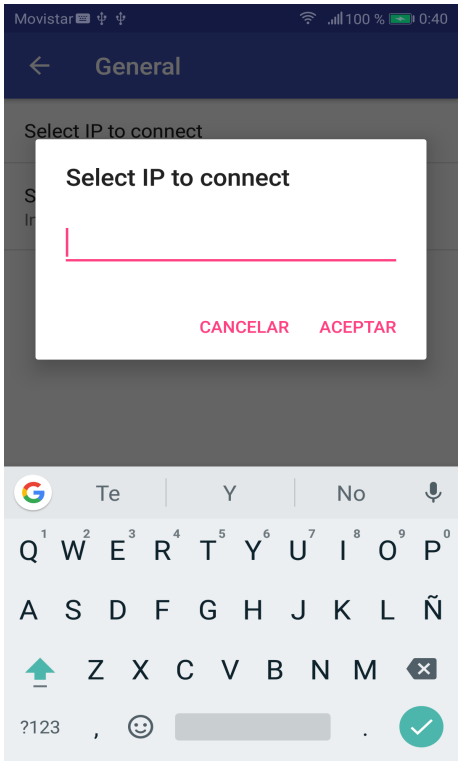

Figura 6.17: Sección de dirección IP.

Y el referente a la selección de modos o grados de consciencia, diseñado a partir de un *ListPreference*, que hace la misma función que un *RadioGroup*, donde el usuario puede activar el modo elegido (solo se puede elegir un modo).

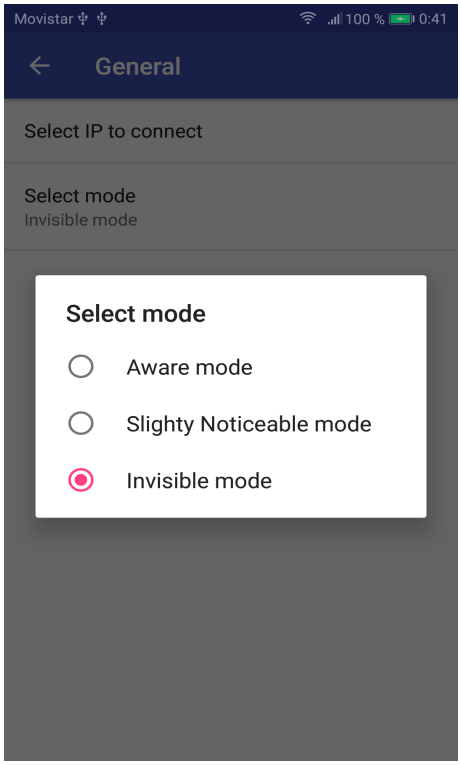

Figura 6.18: Sección de modos (grados de consciencia).

A continuación, se muestra una imagen de ejemplo con los dos valores rellenados:

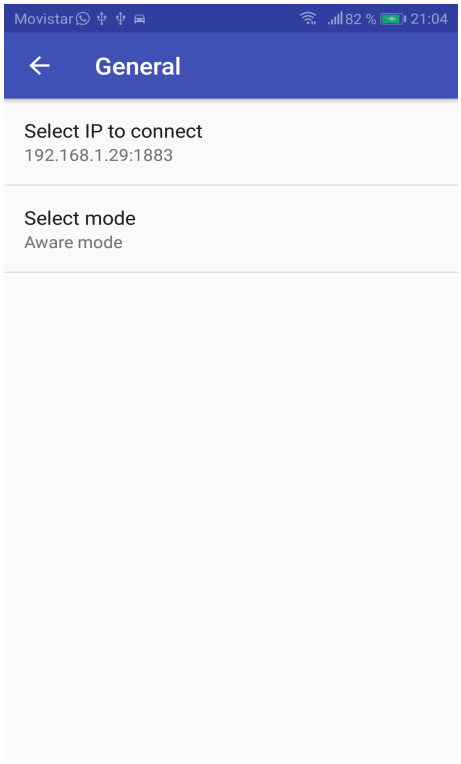

Figura 6.19: Ajustes completados.

Dentro de "Notificaciones", se visualiza una opción de selección de tono, desarrollada a partir de un *RingtonePreference* donde se accede a la biblioteca de tonos de notificaciones del dispositivo (ubicada en). También se puede ver un *SwitchPreference* que, por falta de tiempo, no se ha podido estudiar su eliminación sin afectar al trabajo, ya que su borrado implicaba el no poder habilitar la selección de tono.

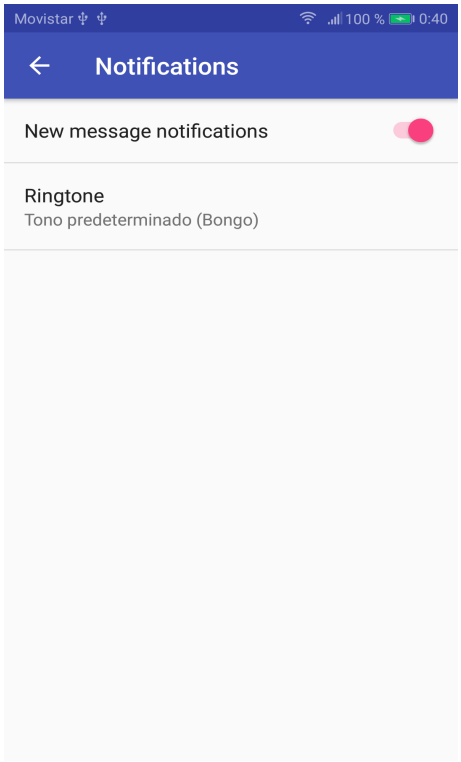

Figura 6.20: Sección de "Notificaciones".

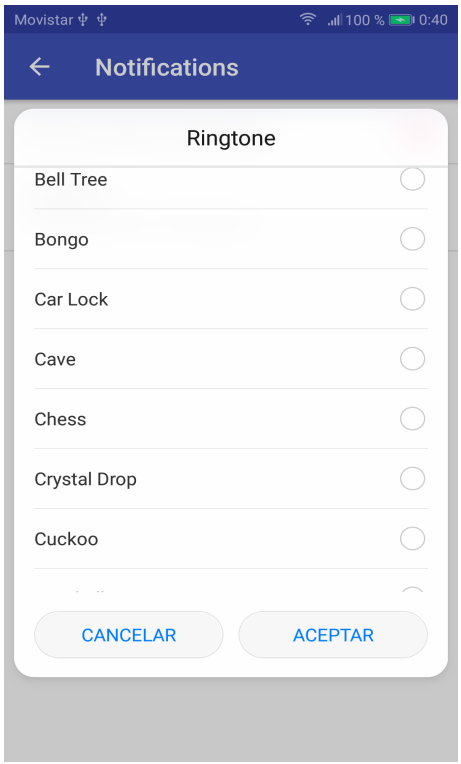

Figura 6.21: Selección de tono.

Alertas y recepción de notificaciones

Aunque ya se ha descrito el diseño de las alertas usadas por la aplicación en el punto 5.3, ahora vamos a complementar esa información de una forma más técnica.

Las alertas dependen del grado de consciencia activo y del tipo de notificación. Generalmente (salvo en un caso que será descrito posteriormente), se recibe una notificación que es indicada en la barra de tareas del propio dispositivo, apareciendo con un icono de un coche (este icono utilizado pertenece a la propia biblioteca de Android). Una vez desplegada la barra de tareas, se muestra el *ticket* de la notificación, que contiene el título (ahí se especifica el tipo de notificación) y la descripción.

Figura 6.22: Notificación .

Si el usuario pulsa en la notificación, se abrirá la vista de detalles de la notificación con la alerta correspondiente a su tipo, hallándose estas tres:

- **Alerta de confirmación:** Si el tipo de petición es *notify*, se muestra una alerta con el título y el contenido de la petición. El usuario pulsar en *OK* o *Cancelar* para confirmar que ha recibido la notificación, ya que al ser informativa, no se permite realizar una cancelación.

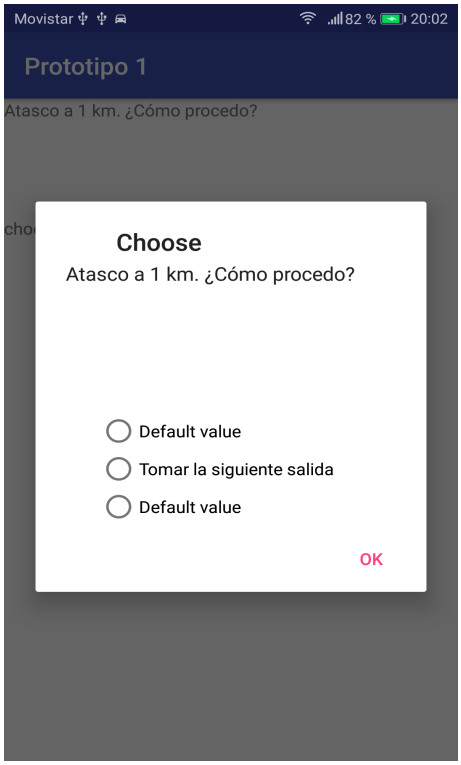

Figura 6.23: Alerta de confirmación.

- **Alerta de elección:** Si el tipo de petición es *choose*, se muestra una alerta con el título, el contenido de la petición y una serie de opciones como respuesta a dicha petición. Estas opciones se insertan utilizando un *RadioGroup* compuesto por *RadioButton*, permitiendo una única selección cada vez. El usuario puede seleccionar una de esas peticiones y pulsar en *OK* o *Cancelar*. En caso de pulsar el primero, se mandará una respuesta con la opción seleccionada por el usuario. En caso de optar la cancelación, no se construirá ni se mandará una respuesta.

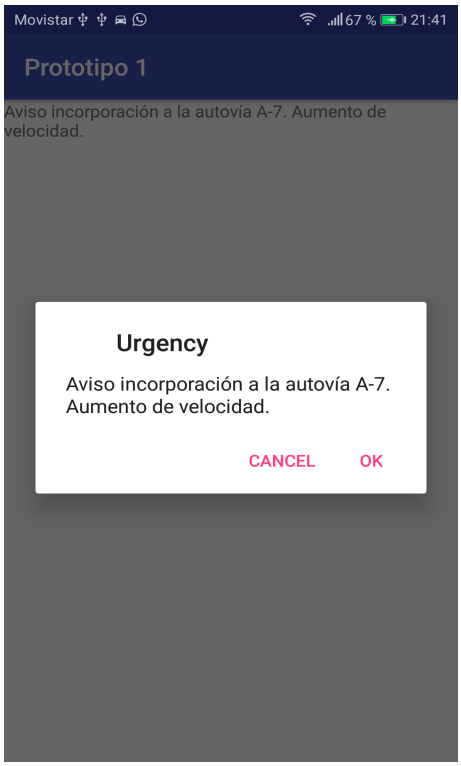

Figura 6.24: Alerta de elección.

- **Alerta de texto editable:** Si el tipo de petición es *ask*, se muestra una alerta con el título, el contenido de la petición y una opción de texto editable, diseñada con un *EditText*. El usuario podrá escribir la respuesta que desee a dicha petición y pulsar en *OK* o *Cancelar*. En caso de pulsar el primero, se mandará una respuesta con el texto escrito por el usuario. En caso de optar la cancelación, no se construirá ni se mandará una respuesta.

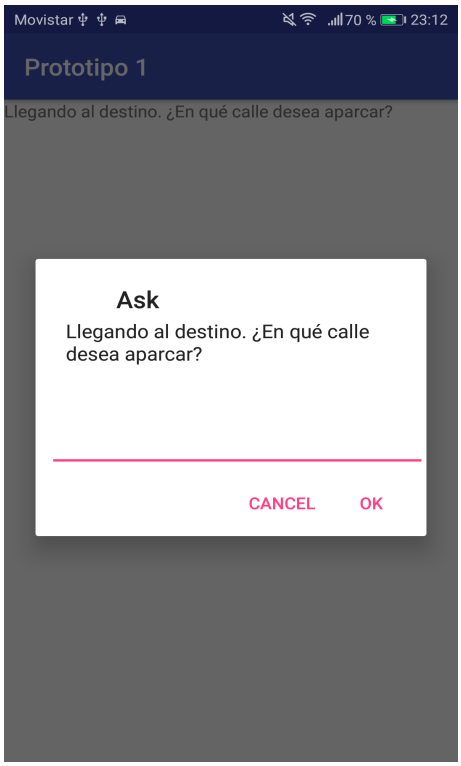

Figura 6.25: Alerta de texto editable.

En el caso de que el tipo de notificación sea *notify* y el grado de consciencia activo sea *Aware*, se mostrará, además de la alerta, un *Toast* con la descripción de la notificación, como se puede ver en la imagen 6.26.

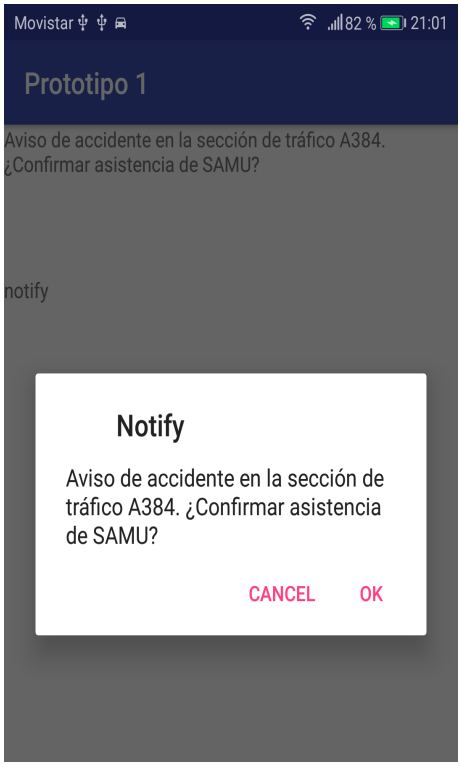

Figura 6.26: Alerta de confirmación con *Toast*.

CAPÍTULO 7
Ejemplo de uso de la aplicación

En este capítulo, se va a exponer un ejemplo de uso del sistema. A pesar de que se ha desarrollado una aplicación que funciona en todo tipo de situaciones dentro de una ciudad inteligente, se va a presentar un caso de emergencia en el que un coche ha sufrido un accidente.

Iniciamos el ejemplo con un usuario situado en su vehículo inteligente, donde decide seleccionar el modo "*Invisible*", ya que hay gente en el vehículo y no desea ser molestado con todas las notificaciones recibidas:

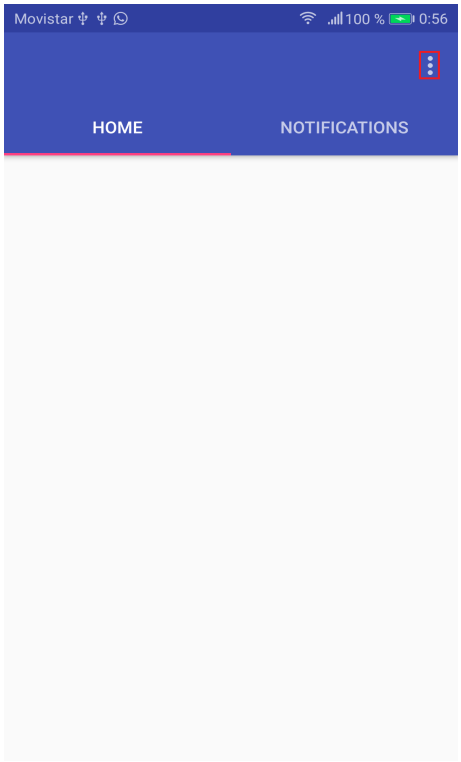

Figura 7.1

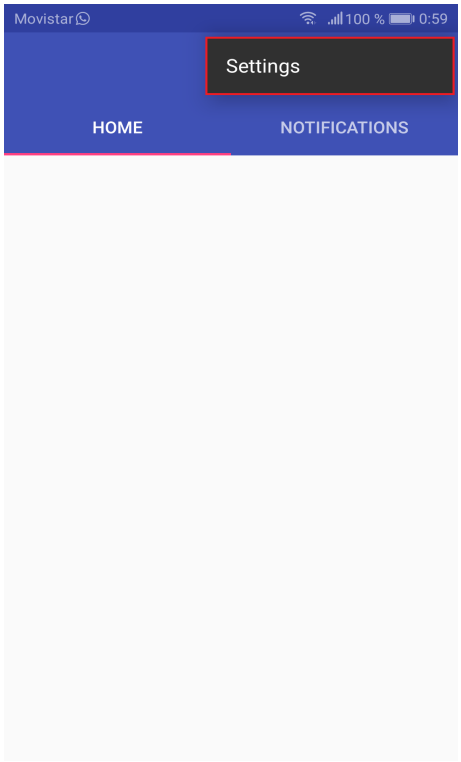

Figura 7.2

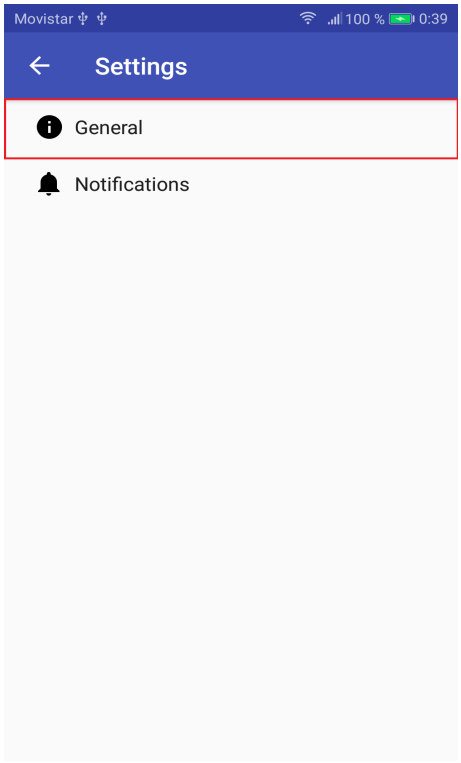

Figura 7.3

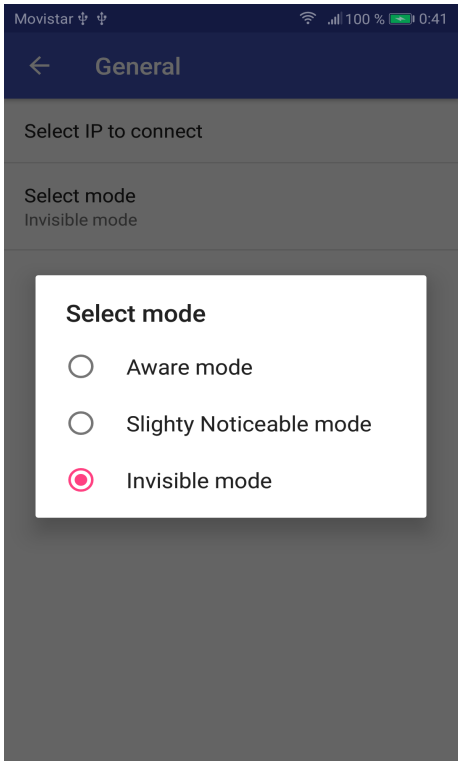

Figura 7.4

Además, decide cambiar el tono de notificación, consiguiendo uno que tiene un sonido más receptivo:

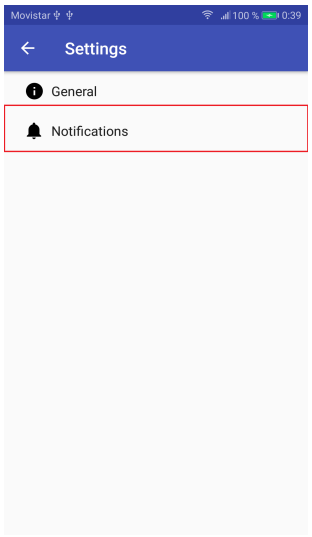

Figura 7.5

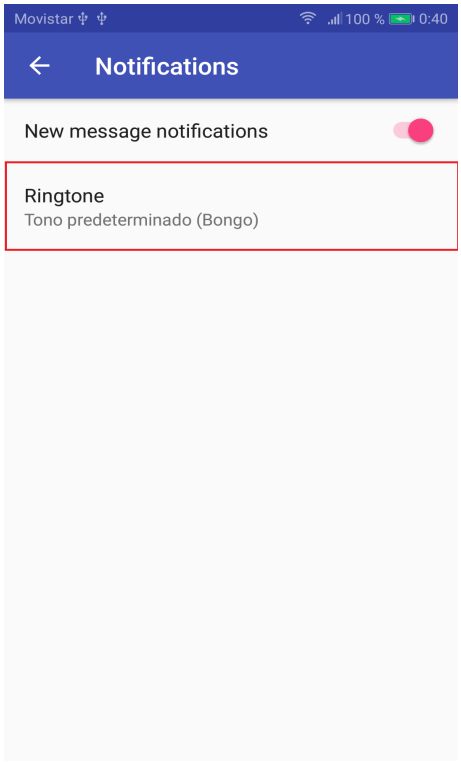

Figura 7.6

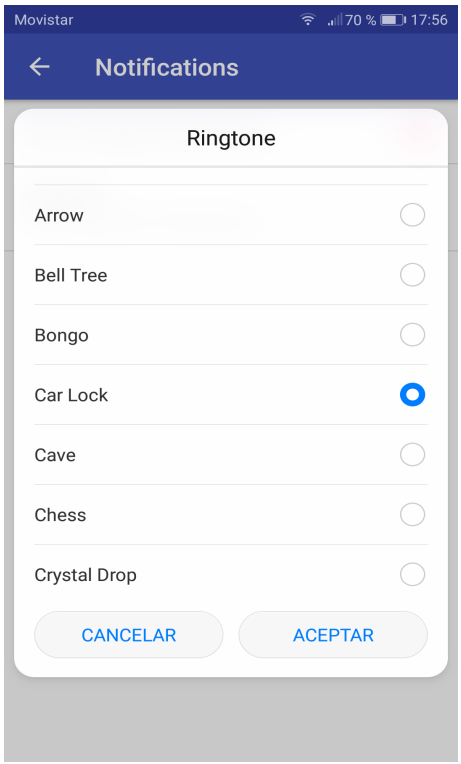

Figura 7.7

Recibe una primera petición, como se puede ver en la siguiente imagen:

Figura 7.8

Tras responder a que calle desea aparcar, el usuario va a sufrir un accidente. Tras el suceso, recibe una petición de tipo *choose*, donde se le pregunta si necesita asistencia y se le dan cuatro opciones como respuesta.

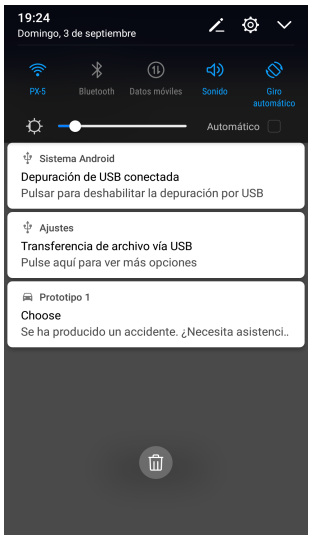

Figura 7.9

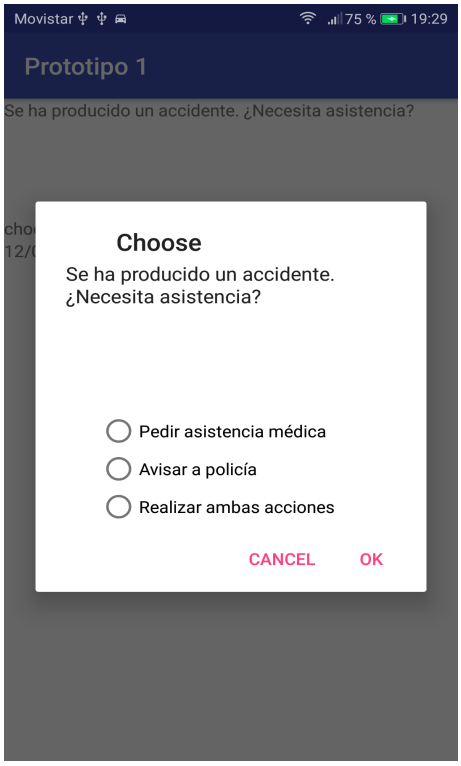

Figura 7.10

Una vez a respondido a la alerta pidiendo asistencia sanitaria y policía, el encargado de la sección de tráfico donde se ha producido el accidente, recibe una petición a raíz de la respuesta del usuario, avisando de que hay que mandar una ambulancia y una patrulla policial.

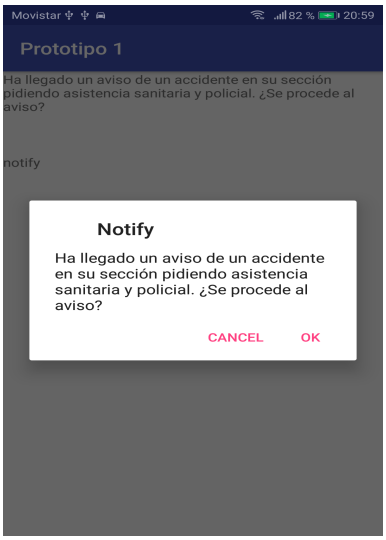

Figura 7.11

A continuación, tanto el servicio médico como la comisaría más cercana reciben notificaciones acerca del accidente y pidiendo el envío de asistencia. Una vez confirmada, acuden al lugar de los hechos.

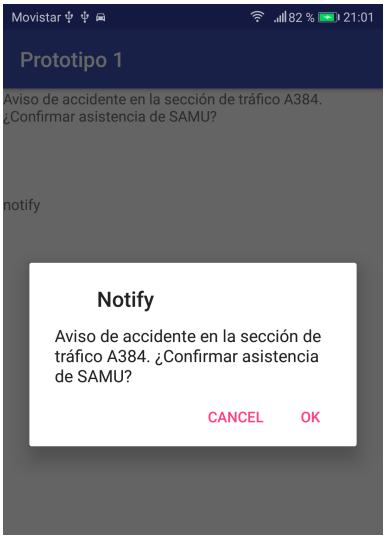

Figura 7.12

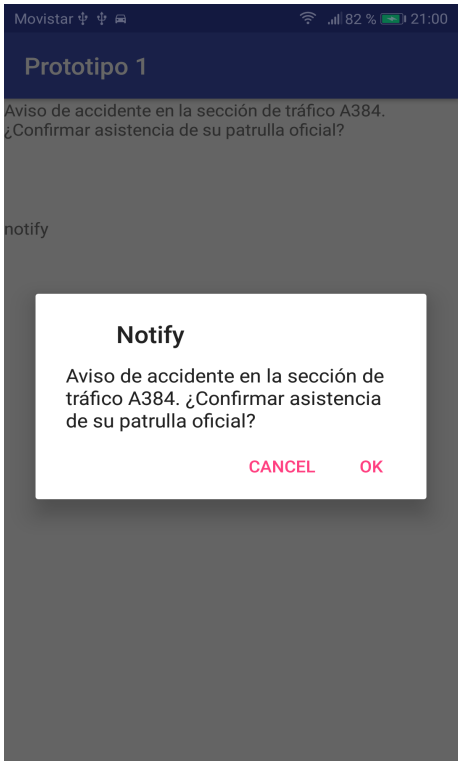

Figura 7.13

CAPÍTULO 8
Conclusión

En este capítulo se van a exponer las conclusiones alcanzadas tras el desarrollo de este trabajo de final de grado, así como las consecuentes mejores que se podrían implementar en el proyecto en un futuro.

8.1 Conclusiones

Como se ha comentado a lo largo de este documento y, especialmente, en la introducción, el usuario necesita una constante recepción de información por parte de su dispositivo inteligente, para poder ser involucrado en ese bucle de información que comparten todos los componentes inteligentes de una ciudad. Teniendo este hecho en cuenta, se ha desarrollado una aplicación que permite al usuario estar constantemente informado sobre los hechos que ocurren alrededor de una ciudad inteligente.

Además, se ha conseguido implementar un sistema que permita al usuario recibir esa constante información y otro tipo de notificaciones de forma no intrusiva, ya que se tiene en cuenta su posible grado de consciencia en todo momento. Todo esto conforma una aplicación que puede ser instalada para todo tipo de servicios y situaciones dentro de una ciudad inteligente, desde un servicio de emergencias o de regulación de tráfico hasta recogida de basuras o uso particular, siempre teniendo en cuenta la participación del humano en la comunicación.

Por lo tanto, se ha desarrollado una aplicación que puede hallarse en el futuro del Internet de las Cosas y, en concreto, en ciudades inteligentes donde gracias a su facilidad de uso, puede ayudar al futuro crecimiento de este tipo de entornos.

8.2 Posibles mejoras a realizar

En esta sección se van a presentar posibles mejores que no se han implementado por falta de tiempo o que se han encontrado tras una serie de pruebas sobre la aplicación. Dichas mejoras son:

- Mostrar información sobre la sección de carretera en la que se haya el coche actualmente, con tal de que el usuario se siente más informado.
- Utilizar una capa de seguridad a la hora del envío de mensajes a través de MQTT.
- Enseñar el listado de notificaciones de una forma más amigable para el usuario.
- Mostrar la respuesta que se haya dado a la notificación (en caso de que se haya contestad a ella).
- Permitir el cambio de voz así como el cambio de idioma con el que se leen el contenido de las peticiones.
- Bloquear las notificaciones urgentes para que el usuario no pueda ignorarlas.

Bibliografía

[1] Internet de las Cosas *¿Qué es y cómo funciona Internet de las Cosas?*. Hipertextual. https://hipertextual.com/archivo/2014/10/internet-cosas/

[2] Álvaro Cárdenas. *Smart City España: un vistazo a la situación actual de las Smart Cities*. https://secmotic.com/blog/smart-city-espana-situacion-smart-cities/, mayo, 2017.

[3] Érika Fernández. *Los coches inteligentes ya son una realidad*. Madridesnoticia. http://www.madridesnoticia.es/ciencia_y_tecnologia/nuevas_tecnologias/coches-inteligentes-ya-son-realidad, marzo, 2017.

[4] Human In The Loop (HIL) *What is Human In The Loop?*. Tallify. https://tallyfy.com/human-in-the-loop/

[5] MQTT. http://mqtt.org/documentation

[6] Android. https://www.android.com

[7] Android Developers. Documentación sobre Android Studio. https://developer.android.com/studio/intro/index.html

[8] IntelliJ IDEA. IntelliJ IDEA, compilador desarrollado por JetBrains. https://www.jetbrains.com/idea/

[9] Oracle. Documentación sobre Java. http://www.oracle.com/technetwork/java/javase/documentation/index.html

[10] The Eclipse Paho project. Documentación sobre la librería Eclipse Paho. https://wiki.eclipse.org/Paho

[11] MQTT.FX. El cliente MQTT basado en JavaFX. http://mqttfx.org

[12] Activity, publicado en Android Developers. https://developer.android.com/reference/android/app/Activity.html

[13] Service, publicado en Android Developers. https://developer.android.com/reference/android/app/Service.html

[14] Fragment, publicado en Android Developers. https://developer.android.com/reference/android/app/Fragment.html

[15] Notification, publicado en Android Developers. https://developer.android.com/reference/android/app/Notification.html

[16] Docker. Documentación sobre Docker. https://docs.docker.com

[17] Eclipse Mosquitto™. Documentación sobre Mosquitto. http://mosquitto.org/documentation/

[18] Github. Documentación sobre el contenedor virtual toke/docker-mosquitto. https://github.com/toke/docker-mosquitto

[19] Alert Dialog, publicado en Android Developers. https://developer.android.com/reference/android/app/AlertDialog.html

[20] Alert Dialog Builder, publicado en Android Developers. https://developer.android.com/reference/android/app/AlertDialog.Builder.html

[21] Toolbar, publicado en Android Developers. https://developer.android.com/training/appbar/setting-up.html?hl=es-419

[22] Tablayout, publicado en Android Developers. https://developer.android.com/reference/android/support/design/widget/TabLayout.html

[23] Edit Text, publicado en Android Developers. https://developer.android.com/reference/android/widget/EditText.html

[24] Radio Button, publicado en Android Developers. https://developer.android.com/reference/android/widget/RadioButton.html

[25] Radio Group, publicado en Android Developers. https://developer.android.com/reference/android/widget/RadioGroup.html

APÉNDICE A

Instalación y uso de Mosquitto en Ubuntu 16.04

A continuación, se van a describir los detalles de la instalación y del uso del *broker* de conexiones MQTT *mosquitto* para Ubuntu 16.04. Este software ha sido utilizado como nodo central en este proyecto debido a su facilidad de instalación y configuración

A.1 Instalación

Antes de proceder con la instalación, se recomienda actualizar los índices de los paquetes del sistema operativo, con el fin de obtener la versión de *mosquitto* más actualizada.

```
270E5EV-270E4EV:~$ service mosquitto stop
```

Figura A.1: Actualización de índices.

Una vez sincronizado el índice, se usa la instrucción mostrada a continuación de este párrafo para la instalación del programa.

```
270E5EV-270E4EV:~$ sudo apt-get install mosquitto
```

Figura A.2: Instalación de *mosquitto*.

A.2 Uso

Mosquitto no tiene complicaciones a la hora de usarlo. Sin embargo, se van a mostrar las opciones más básicas a la hora de utilizar este servicio y sus resultados.

Antes de empezar, se recomienda parar el servicio que es arrancado por defecto en el sistema, ya que no siempre permite la conexión al puerto abierto por defecto.

```
270E5EV-270E4EV:~$ service mosquitto stop
```

Figura A.3: Para servicio de *mosquitto*.

Una vez parado el servicio, tenemos dos opciones:

- Se utiliza el comando por defecto, que arranca el software utilizando el puerto por defecto de MQTT, el 1883.

```
270E5EV-270E4EV:~$ mosquitto
```

Figura A.4: Arrancando *mosquitto*.

Una vez ejecutado, obtenemos este resultado:

```
1504391282: mosquitto version 1.4.14 (build date Tue, 11 Jul 2017 00:29:26 +0100) starting
1504391282: Using default config.
1504391282: Opening ipv4 listen socket on port 1883.
1504391282: Opening ipv6 listen socket on port 1883.
```

Figura A.5: Resultado de ejecutar *mosquitto*.

- Se puede especificar el puerto en concreto que quiere abrir para recibir las conexiones, como se muestra en la siguiente imagen:

  Con este comando, conseguimos estos detalles:

```
270E5EV-270E4EV:~$ mosquitto -p 1885
```

Figura A.6: Arrancando *mosquitto* en el puerto 1885.

```
1504391600: mosquitto version 1.4.14 (build date Tue, 11 Jul 2017 00:29:26 +0100) starting
1504391600: Using default config.
1504391600: Opening ipv4 listen socket on port 1885.
1504391600: Opening ipv6 listen socket on port 1885.
```

Figura A.7: Resultado de ejecutar *mosquitto* en el puerto 1885.

Cuando ya se ha arrancado el nodo central, permitiendo a otros componentes conectarse a él y publicar contenido o suscribirse a una cola. Por lo tanto, en caso de que se conecte un elemento externo, en terminal se muestra estos datos:

```
1504392008: mosquitto version 1.4.14 (build date Tue, 11 Jul 2017 00:29:26 +0100) starting
1504392008: Using default config.
1504392008: Opening ipv4 listen socket on port 1883.
1504392008: Opening ipv6 listen socket on port 1883.
1504392075: New connection from 127.0.0.1 on port 1883.
1504392075: New client connected from 127.0.0.1 as MQTT_FX_Client (c1, k60).
```

Figura A.8: Cliente conectado a *mosquitto*.

Sin embargo, en caso de desconexión, se muestra el usuario del cliente que se ha desconectado, como se puede ver en la imagen A.9:

Figura A.9: Cliente desconectado de *mosquitto*.

APÉNDICE B
Uso de MQTT.FX

En este apéndice, se va a explicar como funciona el cliente de MQTT usado para la pruebas de la aplicación en este trabajo. Como ya se mencionó en el punto –, este cliente está desarrollado con la librería Eclipse Paho en Java y es compatible tanto con Linux, Windows y OS.

Cuando se abre el programa, se puede visualizar la pantalla principal. En ella se puede ver las opciones de publicar, suscribirse y conectarse.

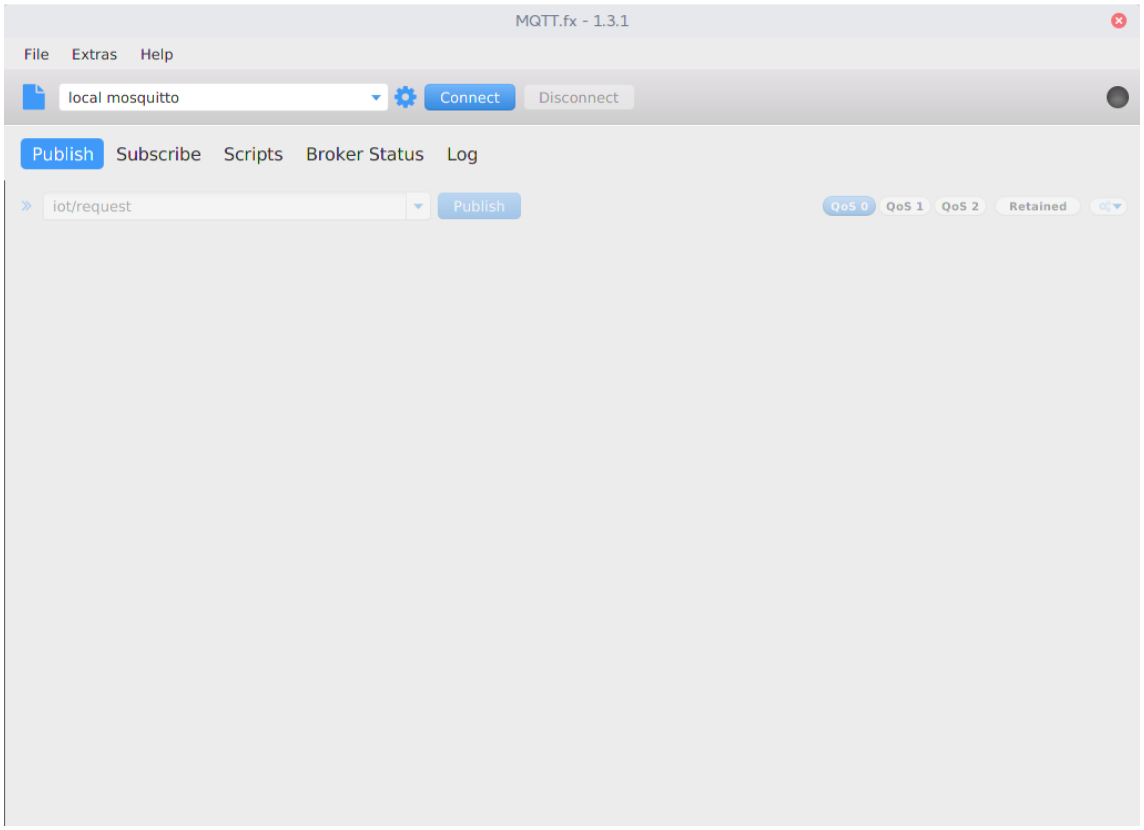

Figura B.1: Vista inicial de MQTT.FX.

En caso de querer crear una conexión al nodo central, clickando en el botón de ajustes se muestra un pequeño formulario. En él se puede ver que ya tiene unos valores por defectos escritos para realizar una conexión en local al puerto 1883.

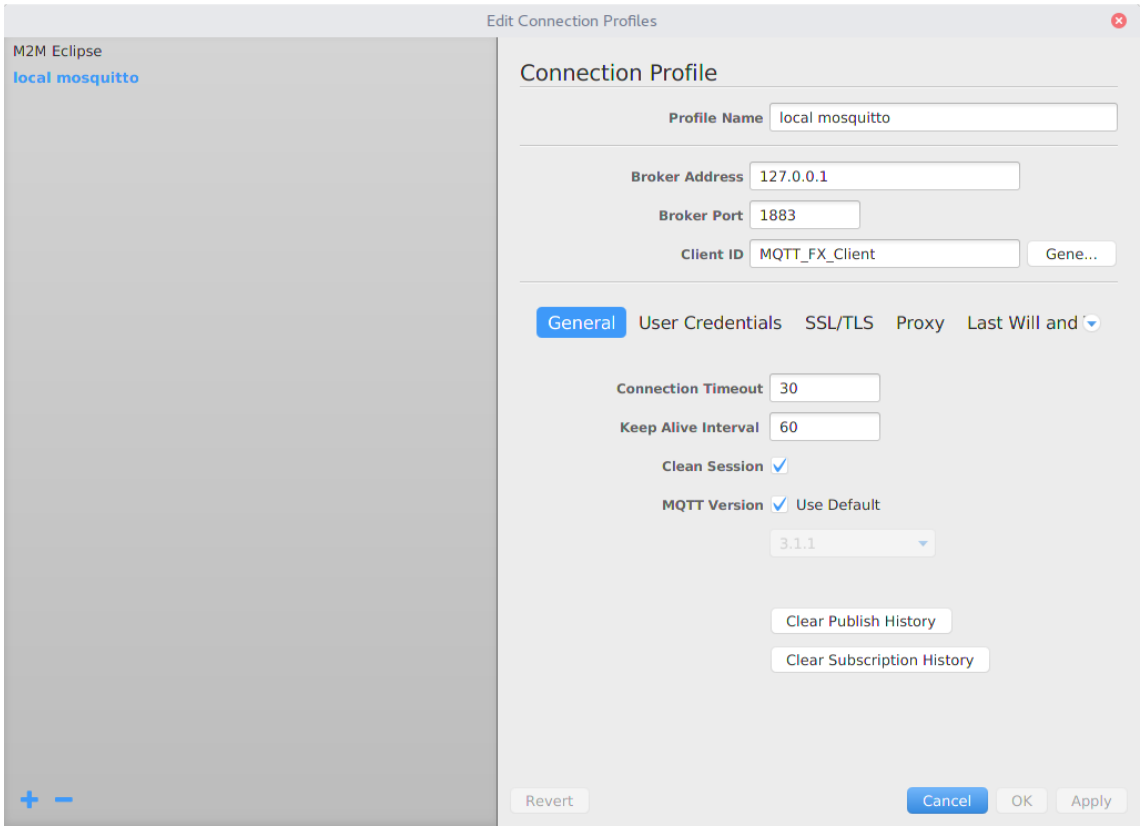

Figura B.2: Configuración de MQTT.FX.

En caso de querer publicar un mensaje, escribimos la cola donde queremos enviar la publicación y presionamos sobre el botón "*Publish*.

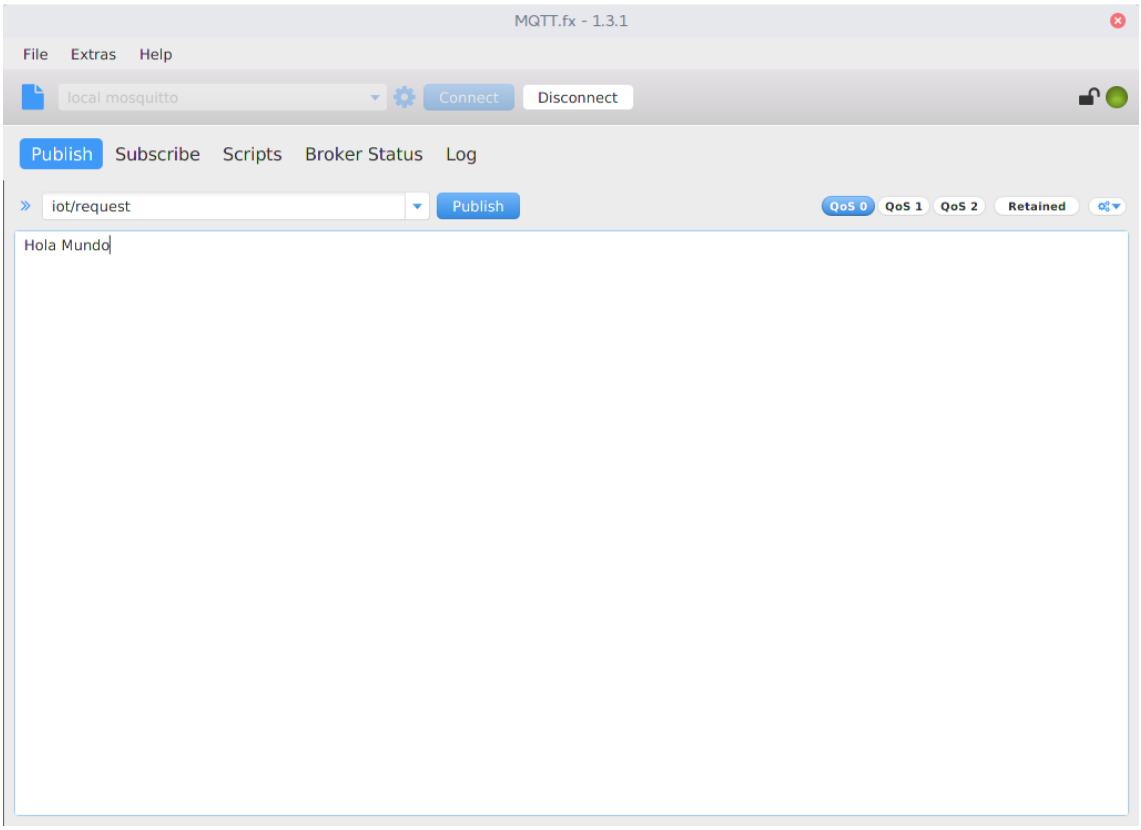

Figura B.3: Publicación de mensajes en MQTT.FX.

En caso de querer una suscripción, se clicka en el apartado "*Subscribe*", donde se permite escribir sobre que cola queremos realizar la acción. Una vez suscritos, se pueden visualizar los mensajes recibidos y a través de que cola, como se puede ver en la imagen B.4.

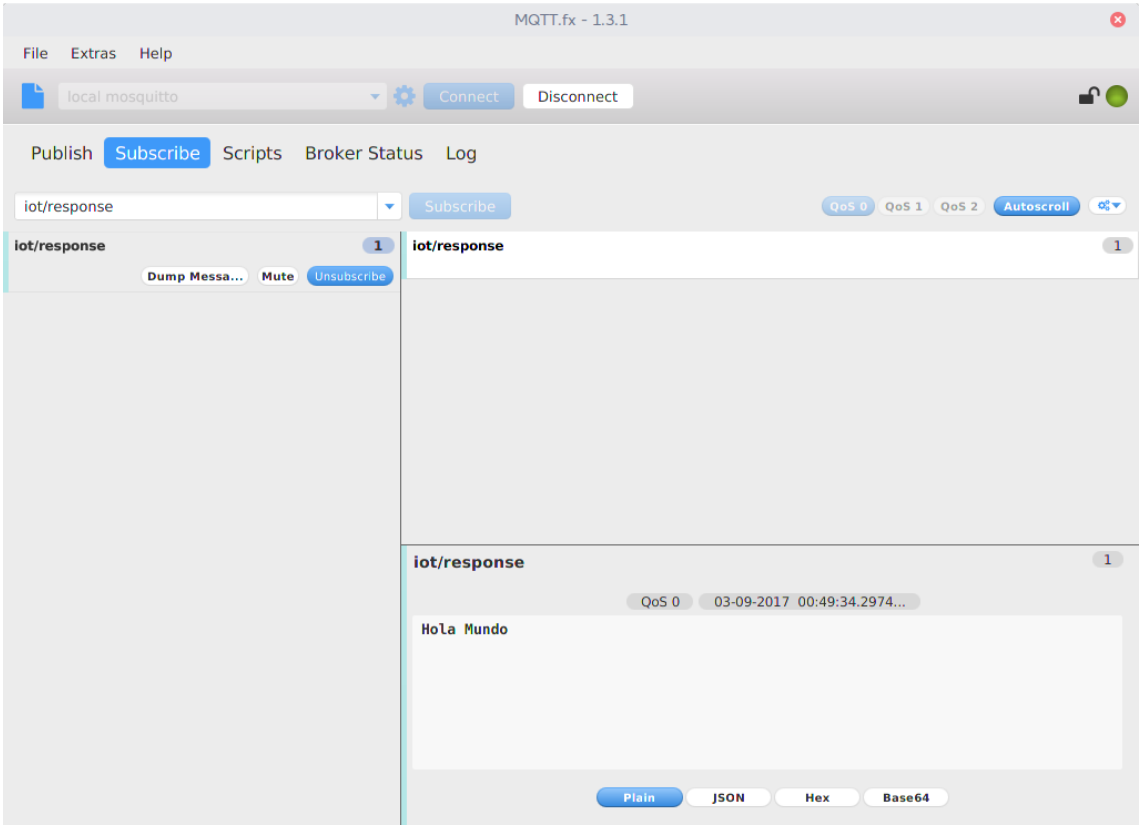

Figura B.4: Suscripción y recepción de mensajes en MQTT.FX.