UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

DESARROLLO DE UNA APLICACIÓN ANDROID PARA LA COMUNICACIÓN ENTRE EQUIPOS DE BOMBEROS

DEVELOPMENT OF AN ANDROID APPLICATION FOR COMMUNICATION BETWEEN TEAMS OF FIREFIGHTERS

Realizado por
Rubén Puga Roldán

Tutorizado por
Eduardo Guzmán De los Riscos

Departamento
Departamento de Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, NOVIEMBRE DE 2019

Fecha defensa:

Fdo. El/la Secretario/a del Tribunal

D. Rubén Puga Roldán, con DNI 76639855-Z, estudiante del Grado/Máster en Ingeniería del Software, de la Universidad de Málaga.

DECLARO QUE:

El Trabajo Fin de Grado/Máster denominado: DESARROLLO DE UNA APLICACIÓN ANDROID PARA LA COMUNICACIÓN ENTRE EQUIPOS DE BOMBEROS

es de mi autoría, inédito (no ha sido difundido por ningún medio, incluyendo internet) y original (no es copia ni adaptación de otra), no habiendo sido presentado anteriormente por mí ni por ningún otro autor ni en parte ni en su totalidad. Así mismo, se ha desarrollado respetando los derechos intelectuales de terceros, para lo cual se han indicado las citas necesarias en las páginas donde se usan, y sus fuentes originales se han incorporado en la bibliografía. Igualmente se han respetado los derechos de propiedad industrial o intelectual que pudiesen afectar a cualquier institución o empresa.

Para que así conste, firmo la presente declaración en Málaga, a 12 de Noviembre de 2019.

Fdo.: D. Rubén Puga Roldán

Resumen

Englobado dentro del ámbito del desarrollo de software, este trabajo fin de grado se ha llevado a cabo con el objetivo de realizar una aplicación móvil Android con la finalidad de ser utilizada para la comunicación interna de los equipos de bomberos cuando están desplegados en una zona para atender a cierta incidencia. No obstante la aplicación desarrollada también puede satisfacer las necesidades de otros cuerpos de emergencia, para la gestión de equipos e incidencias.

Para cubrir ese objetivo, en la aplicación los usuarios administradores podrán dar de alta, modificar o borrar cuentas de administradores y equipos, así como crear, modificar y archivar incidencias dando la posibilidad de asignarle los equipos que intervendrán y de indicar con precisión el lugar donde se produce tanto añadiendo la dirección como marcándolo sobre un mapa. Además, dará la posibilidad a estos usuarios de consultar la información de incidencias ya archivadas. Por último, tanto los usuarios administradores como los equipos asignados a una incidencia podrán intercambiar mensajes tanto de texto como de audio, pudiendo además observar en tiempo real la ubicación del resto de equipos desplegados en dicha incidencia.

Este proyecto hace uso de otro trabajo fin de grado desarrollado previamente por Carlos Martínez, que proporciona diversas funcionalidades tales como la de facilitar un servicio web para la gestión de la base de datos, un servicio de señalización necesario para la comunicación entre clientes y un cliente web cuya funcionalidad es replicada e incrementada por la aplicación Android que es el objetivo de este trabajo fin de grado, haciendo uso de la mejor capacidad de geolocalización que una aplicación nativa para el dispositivo aporta sobre el cliente web, así como una interfaz mejor adaptada a dichos dispositivos.

En esta aplicación se lleva a cabo la implementación de diferentes componentes software entre los cuales se encuentran un cliente de servicio web, un cliente de *websocket*, un cliente *WebRtc* y un servicio Android.

Palabras clave

Gestión de incidencias, Android, WebRtc, websocket, patrón MVVM.

Abstract

This Final Year Dissertation, located within the scope of software development, has been done with the purpose of developing a mobile Android application in order to be used by firefighters, it could be used by any other emergency response teams, to help them manage teams and incidents.

In order to achieve this goal, in our application the administrators will be able to add, modify and delete administrator and team accounts, as well as add, modify and archive incidents, allowing them to assign the teams that will participate in an incident and to precisely set the location where the incident is taking place by filling in the address or clicking on a map. In addition, administrators will be able to check the information of archived incidents. Finally, both administrators and teams assigned to an incident will be able to communicate by text and audio messages, as well as observe in real time the location of all the teams involved in an incident.

This project use another final year dissertation, previously developed by Carlos Martínez, that provides different features such as a web service for the database management, a signaling server that is needed for the communication between the clients and a web client whose functionality is replicated and improved by the Android application which is the purpose of the dissertation, taking advantage of a better geolocation provided by a native application compared to a web client, as well as a user interface that's best adapted to those kind of devices.

In this software, as it will be more deeply explained in this document, different software components are deployed such as a webservice client, a websocket client, a webrtc client, an Android service apart from some recommended patterns for Android software development such as Navigation for the views management.

Keywords

Incidents management, Android, WebRtc, websocket, MVVM-pattern.

Índice

Resumen **7**

Índice **9**

1. Introducción **11**

1.1 Motivación **11**

1.2 Objetivos **11**

1.3 Estructura de la memoria **12**

2. Análisis **15**

2.1 Análisis de los servicios proporcionados 15

2.1.2 Análisis de la base de datos 15

2.1.3 Análisis del servicio RETS para la gestión de la base de datos 17

2.1.4 Análisis del servidor de señalización 18

2.2 Análisis funcional **21**

2.2.1 Requisitos funcionales 21

2.2.2 Casos de Uso 23

3. Implementación **45**

3.2. Librerías y componentes Android utilizados **46**

3.2.1. Retrofit 48

3.2.2. OkHttp 48

3.2.3. WebRTC 50

3.2.4. Navigation 51

3.2.5. Google Maps 58

3.3. Desglose de la implementación **59**

3.3.1. Servicio 60

3.3.2. Modelo 61

3.3.2. Vistas 65

3.3.3. Vista-Modelos 69

4. Conclusiones **73**

5. Bibliografía **75**

6. Anexo: Manual de Usuario **77**

1. Introducción

En este apartado de introducción se expondrá la motivación por la cual se ha realizado este trabajo fin de grado, así como los objetivos que deberán quedar cubiertos tras su realización. Por último, se expondrán cuáles son las partes principales de las que va a constar este documento de memoria, con un pequeño resumen de cuáles van a ser los aspectos cubiertos en cada uno de estos apartados.

1.1 Motivación

Poner a disposición de los servicios de emergencia que se encargan de intervenir y resolver las incidencias que día a día tienen lugar herramientas tecnológicas que permitan, faciliten o agilicen esas labores es uno de los aspectos más importantes en los que el desarrollo de software puede contribuir socialmente.

En este marco se engloba la realización de este trabajo fin de grado, cuyo objetivo principal es el desarrollo de una aplicación móvil destinada al cuerpo de bomberos que permita la gestión de equipos e incidencias, así como la comunicación entre los diferentes equipos que lleven a cabo la intervención en dichas incidencias. La manera en que dichos equipos podrán comunicarse se procede a ser detallada en el apartado siguiente.

1.2 Objetivos

Tal como se ha indicado en el apartado de motivación, el objetivo de este trabajo fin de grado es desarrollar la aplicación en Android que permita que tanto administradores como equipos que forman parte del cuerpo de bomberos gestionen y coordinen las intervenciones en las incidencias.

Entre las acciones que podrán llevar los administradores se encuentran:

- Gestión de usuarios administradores
- Gestión de equipos
- Gestión de incidencias

- Comunicación con los equipos de cada incidencia y con otros administradores por chat y mensajes de voz
- Control de la posición de los equipos en las incidencias activas.

Por su parte los equipos podrán realizar las siguientes acciones:

- Comunicación con los equipos de cada incidencia su incidencia activa y con administradores por chat y mensaje de voz.
- Control de la posición de los equipos en las incidencias activas.

El desglose de los distintos aspectos que componen las acciones que acabamos de citar se llevará a cabo en el apartado de análisis funcional en el que se expondrán los diferentes casos de uso y sus respectivos escenarios que estarán cubiertos por la funcionalidad de la aplicación desarrollada.

Para proporcionar estas características, se va a hacer uso del proyecto citado en el resumen de este trabajo. Por tanto, uno de los objetivos iniciales a cubrir será el análisis de las funciones que tanto el servicio web como el servidor de señalización presentes en dicho proyecto ofrecen para la implementación de nuestros clientes móviles.

Cabe mencionar que este trabajo parte de un sistema realizado anteriormente en otro trabajo de fin de grado realizado por el estudiante Carlos Martínez. En ese trabajo se desarrolló un componente de backend que proporciona un conjunto de servicios que son necesarios para la aplicación móvil que se ha desarrollado en el presente trabajo.

1.3 Estructura de la memoria

Esta memoria se va a estructurar en tres partes principales.

La primera de ellas será el apartado de análisis dentro del cual, se encontrará tanto el análisis de los servicios proporcionados para la base de datos y el servidor de señalización, como los casos de uso y diseños de interfaz que se han considerado adecuados para la implementación posterior de toda la funcionalidad requerida.

En el siguiente apartado se procederá a la exposición de la implementación llevada a cabo que dará lugar a la aplicación resultado de este trabajo. En dicha exposición, se desglosarán los distintos componentes que se han desarrollado, haciendo hincapié en los patrones, librerías y elementos de Android empleados, así como el uso de modelos para facilitar la comprensión del funcionamiento de la aplicación.

Por último, en la parte final se expondrán las conclusiones a las que se ha llegado tras la realización de este trabajo fin de grado, tanto desde es el punto de vista de lo que ofrece Android como ciertos problemas que se han ido encontrando a lo largo de la elaboración del mismo. También se analizarán posibles mejoras que se han considerado que podrían ser implementadas en futuras versiones de esta aplicación y que permitirían ofrecer mayores funcionalidades que las abarcadas en el resultado de este trabajo.

Además, como anexo se proporciona un pequeño manual de usuario para facilitar el uso de la aplicación por parte de los interesados.

2. Análisis

En este apartado procedemos a realizar la explicación de las fases de trabajo consistentes en el análisis de los servicios proporcionados por el trabajo fin de grado de Carlos Martínez y el análisis funcional llevado a cabo previo a la fase de implementación de la aplicación.

2.1 Análisis de los servicios proporcionados

Este análisis se va a descomponer en tres partes para facilitar su comprensión: en primer lugar se detallará cuál es la base de datos sobre la que trabaja el servicio web que proporciona el acceso al mismo con el fin de comprender mejor su funcionamiento. A continuación, se hará un pequeño análisis del servicio web que se nos proporciona y sobre el cual nuestra aplicación deberá hacer las llamadas. Finalmente, se hará un análisis de las funciones disponibles en el servidor de señalización que es usado por los clientes WebRTC para la comunicación.

2.1.2 Análisis de la base de datos

La base de datos sobre la que trabajará la aplicación es una base de datos no relacional MongoDB que consta de tres colecciones: Admin, Equipo e Incidencia, tal como se observa en la Figura 2.1.2.1

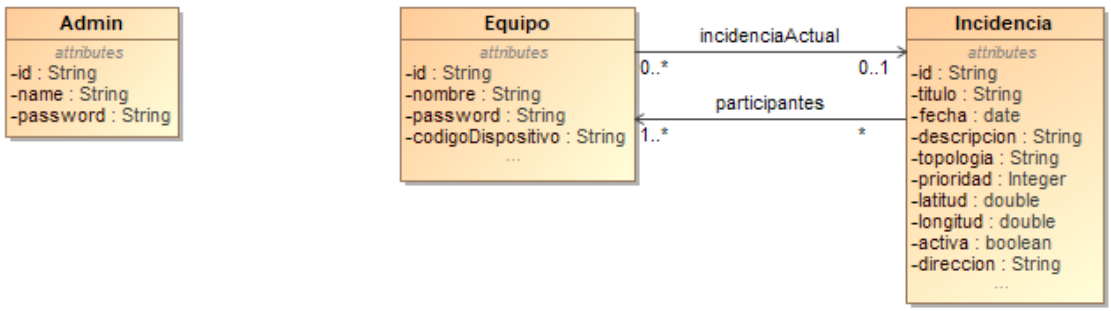


Figura 2.1.2.1 Modelo de datos

Para cada elemento de la colección Admin se almacenan tres datos:

- _id
- name
- pass

Para la colección Equipo se observa que se almacenan los siguientes datos:

- _id
- nombre
- password
- codigoDispositivo
- incidenciaActual: Incidencia asignada actualmente al equipo, la cadena vacía en caso de no tener ninguna.

Finalmente, para la colección Incidencia los datos almacenados para cada elemento son:

- _id
- titulo
- fecha
- descripción
- participantes: Array con el listado de participantes de esta incidencia.
- topología
- prioridad
- latitud
- longitud
- activa: Valor booleano que permitirá distinguir las incidencias activas de las archivadas
- dirección

Una vez que se realizó el estudio de sobre que base de datos se necesita trabajar es momento de analizar el servicio REST que permitirá obtener información y realizar modificaciones sobre la misma.

2.1.3 Análisis del servicio REST para la gestión de la base de datos

El servicio web que se nos proporcionó para interactuar con la base de datos está desarrollado haciendo uso de Loopback, un framework de Node.js que puede ser usado para proporcionar un servicio web REST.
Para cada una de las entidades que se analizaron en el apartado anterior (Admin, Equipo e Incidencia), este servicio web proporciona un *Endpoint:*

- {URL_SERVICIO}/api/Admins
- {URL_SERVICIO}/api/Equipos
- {URL_SERVICIO}/api/Incidencias

Entre las operaciones que vemos nos proporciona entre servicio web nos quedamos con aquellas que nos van a ser necesarias para la funcionalidad requerida. Fundamentalmente, desglosándolas según el método de la petición http, serían las siguientes:

- **GET:**
    - {EndPoint}: Devuelve todas las instancias de esa colección o bien aquellas que cumplan las condiciones indicadas con un parámetro filter enviado en la llamada.
    - {EndPoint}/findOne: Devuelve la primera instancia de aquella o bien la primera que cumpla las condiciones indicadas con un parámetro filter enviado en la llamada.
    - {EndPoint}/{id}: Devuelve la instancia que tiene ese id.

- **POST:**
    - {EndPoint}: Crea una nueva instancia con los datos que se le pasan en formato JSON en el cuerpo de la petición.
- **PUT**:
    - {EndPoint}/{id}: Actualiza los datos de las instancias cuyo id coincida con el pasado en la URL con los datos que se le pasan en los datos de la petición.

- **DELETE:**
    - {EndPoint}/{id}: Borra la instancia que tiene ese id.

Esas serían las peticiones que necesitamos usar en nuestra aplicación para la gestión de los datos del sistema. Posteriormente, en el apartado de implementación se detallará cuál ha sido la solución por la que se ha optado para la gestión del modelo desde Android.

2.1.4 Análisis del servidor de señalización

En primer lugar, es necesario explicar por qué necesitamos un servidor de señalización en nuestro sistema, lo que nos lleva a profundizar en qué es exactamente WebRTC, qué funcionalidad proporciona y cuál es su relación con el servidor de señalización.

Tal como se indica en la web del WebRTC project, se trata de un proyecto de código abierto que permite la comunicación en tiempo real (Real Time Communications, RTC) ofreciendo además capacidades multimedia como la transmisión de audio y video. Ahora bien para que esta comunicación se pueda llevar a cabo es necesario un intercambio previo de información, labor en la que interviene el servidor de señalización del que hemos hablado. Entra las funciones que este servidor de señalización puede realizar se encuentran:

- Intercambio de mensajes de control usados para la apertura y cierre de la comunicación.
- Metadatos sobre aspectos como ancho de banda, códecs de audio y video, etc.
- Información de red como direcciones IPs y puertos.

El servidor de señalización que se nos ha proporcionado consiste en un servidor websocket que almacenará datos de las conexiones y de las incidencias creadas y permitirán el intercambio de información previo a la comunicación WebRTC entre los distintos clientes por medio de mensajes en formato JSON. A continuación se detallan los tipos de mensajes disponibles en este servidor y de los cuales hará uso el cliente websocket que será implementado en nuestra aplicación móvil:

- **loginEquipo**: Inicia la sesión de un usuario en el servidor de señalización. A pesar de su nombre se usará también para los administradores ya que éstos también tendrán la capacidad de participar en los chats. El formato del mensaje JSON a enviar es el siguiente:
    - { **"type"**: "loginEquipo",
      "**origen**" : nombreUsuario,
      "**lat**" : latitudUsuario,
      "**lng**" : longitudUsuario
      }

  El servidor de señalización se encargará de enviar la respuesta con las coordenadas de todos los usuarios que estén con sesión iniciada en el servidor, así como los nombres de los usuarios en la misma incidencia en caso de que existiera.

- **update**: Recibe la actualización de coordenadas de la ubicación del usuario y la retransmite al resto de usuarios conectados. El formato del mensaje es:

    - {"**type**": "update",
      "**origen**" : nombreUsuario,
      "**lat**" : latitudUsuario,
      "**lng**" : longitudUsuario
      }

- **createIncidencia:** Recibe los datos para crear la incidencia en el servidor de señalización y envía un mensaje a cada uno de los participantes de la incidencia con los datos de la misma. El formato del mensaje para esta operación sería:
    - {"**type**": "createIncidencia",
      "**origen**" : nombreUsuario,
      "**titulo**" : tituloIncidencia,
      "**participantes**" : listadoParticipantes,
      "**coordenadas**": coordenadasIncidencia
      }

- **deleteIncidencia:** Borra la incidencia del servidor de señalización y envía un mensaje a todos los participantes de la misma indicando que se borró. El formato para este tipo de operación es:
    - {“**type**”: “deleteIncidencia”,
      “**titulo**” : tituloIncidencia
      }

- **leave:** Elimina la conexión del usuario con el servidor de señalización y envía un mensaje al resto de usuarios conectados indicando esta desconexión. El mensaje a usar es:
    - {“**type**”: “leave”,
      “**origen**” : origen
      }

Los siguientes tres tipos de mensajes son los involucrados en el establecimiento de la conexión WebRTC:

- **offer:** Recibe la descripción de la sesión para la conexión WebRTC y la reenvía al usuario destinatario de la misma. El formato a usar es:
    - {“**type**”: “offer”,
      “**name**” : name
      “**offer**” : descriptorDeSesion
      }

- **answer:** De forma análoga a la operación anterior recibe la respuesta con la descripción de sesión del otro extremo y la reenvía al usuario destinatario de la misma. El formato a usar es:
    - {“**type**”: “answer”,
      “**name**” : name
      “**offer**” : descriptorDeSesion
      }

- **candidate**: Esta operación permite el intercambio de los IceCandidates. Estos objetos serán indicadores de las opciones de conexión existentes de cada

extremo y su intercambio es necesario antes de llevar a cabo la conexión. El formato del mensaje es:

- {“**type**”: “candidate”,
  “**name**” : name
  “**candidate**” : iceCandidate
  }

Con esos tres tipos de mensajes finalizamos el análisis de los servicios ofrecidos en el proyecto con el que tendrá que comunicarse nuestra aplicación. A continuación, se presenta el análisis funcional llevado a cabo y los casos de usos resultantes del mismo.

2.2 Análisis funcional

A partir del cliente web que se nos proporcionó inicialmente y de las distintas funcionalidades a incorporar al mismo que se nos han transmitido, se lleva a cabo el análisis funcional de las capacidades que debe tener nuestra aplicación.
Esta aplicación será utilizada por dos tipos de usuario, por lo tanto, se ha optado por dividir los distintos requisitos funcionales según quien lleva a cabo la operación. Como se podrá observar, la mayor parte de las acciones serán realizadas por los administradores, mientras que el uso de los equipos se basa a su participación en la incidencia asignada en caso de existir ésta.

2.2.1 Requisitos funcionales

- **Administradores**:
  - Iniciar sesión
  - Cerrar sesión
  - Listar las incidencias activas
  - Ver una incidencia activa
    - Enviar un mensaje (de texto, predefinido o de audio) a todos los equipos de una incidencia

        - Enviar un mensaje individual (de texto, predefinido o de audio) a alguno de los equipos de la incidencia
        - Enviar un mensaje individual (de texto, predefinido o de audio) a algún administrador
        - Ver la posición actual de todos los equipos de una incidencia
    - Añadir una nueva incidencia
        - Ver los equipos disponibles para una incidencia
        - Añadir la ubicación de la incidencia indicando la dirección
        - Añadir la ubicación de la incidencia marcando sobre un mapa
    - Editar una incidencia activa
    - Archivar una incidencia
    - Listar incidencias ya archivadas
    - Ver detalles de la incidencia archivada
    - Listar administradores
    - Dar de alta un administrador nuevo
    - Editar un administrador
    - Dar de baja un administrador
    - Listar equipos
    - Dar de alta un equipo
    - Editar un equipo
    - Dar de baja un equipo

- **<u>Equipos</u>**
    - Iniciar sesión
    - Cerrar sesión
    - Si tiene una incidencia activa:
        - Ver los datos de la incidencia
        - Ver la posición de los equipos sobre el mapa
        - Enviar un mensaje (de texto, predefinido o de audio) a todos los equipos de una incidencia
        - Enviar un mensaje individual (de texto, predefinido o de audio) a alguno de los equipos de la incidencia
        - Enviar un mensaje (de texto, predefinido o de audio) a algún administrador

2.2.2 Casos de Uso

Una vez que ya disponemos de los requisitos funcionales es momento de desarrollar los casos de uso. El diagrama con estos casos de uso se puede observar en la figura 2.2.2.1

Algunos de los casos de uso son comunes para los administradores y para los equipos, por tanto, solo se detallarán una única vez. Es el caso del inicio y cierre de sesión y la vista de incidencia activa que incluye el ver las posiciones y el chat con el resto de los usuarios. Por ello, el tipo de usuario Equipo aparece en el diagrama pero no tiene ningún caso de uso más de los que hereda.

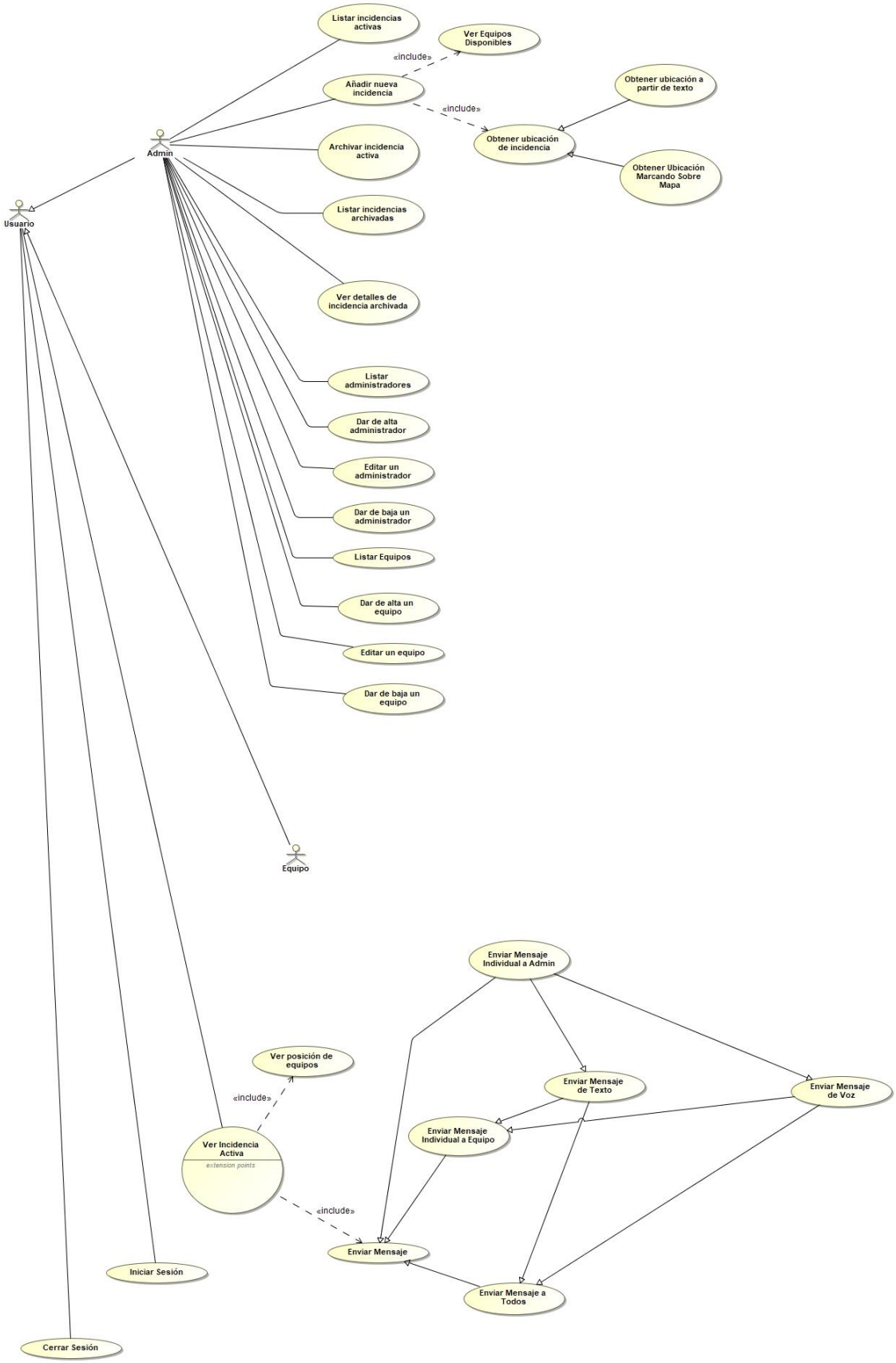
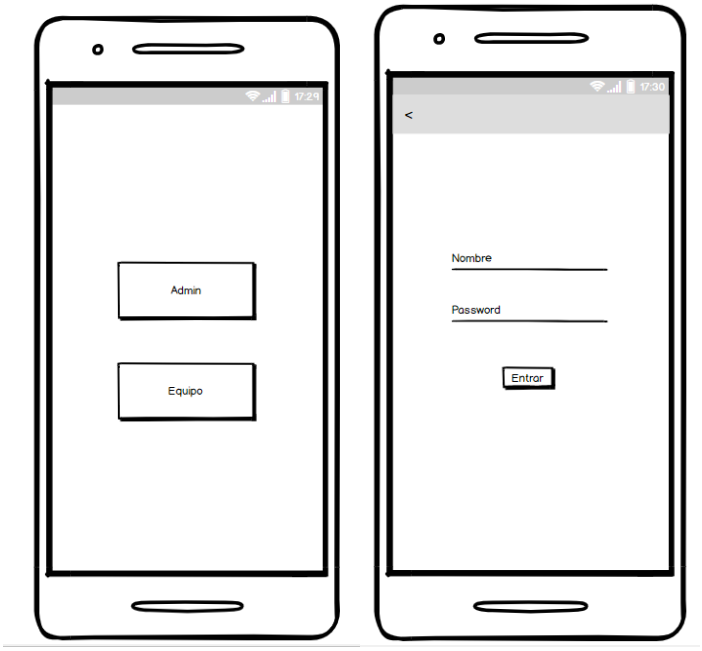


Figura 2.2.2.1 Diagrama de casos de uso

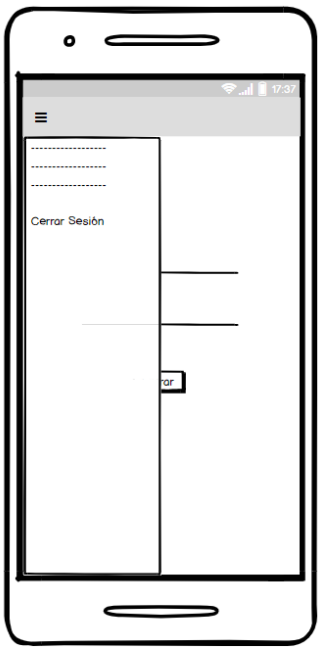
Caso de Uso: Iniciar Sesión

Título	Iniciar Sesión
Descripción	Permite iniciar sesión al usuario en la aplicación
Pre-condición	El usuario tiene la aplicación abierta sin haber iniciado sesión.
Post-condición	El usuario se encuentra con la sesión iniciada en la aplicación. (Caso de Éxito)
Escenario principal	
1. El usuario selecciona su rol (Admin o Equipo) 2. El sistema muestra el formulario para introducir el nombre de usuario y la contraseña 3. El usuario introduce los datos y pulsa en Entrar 4. El sistema valida las credenciales, realiza las conexiones necesarias a los servidores y muestra la pantalla inicial de la aplicación.	
Escenarios alternativos	
3.a El usuario pulsa en el botón atrás. 3.a.1 Se vuelve a la pantalla inicial.	
4.a El sistema no valida las credenciales. 4.a .1 El sistema muestra un dialogo de error.	

Maquetas de interfaz


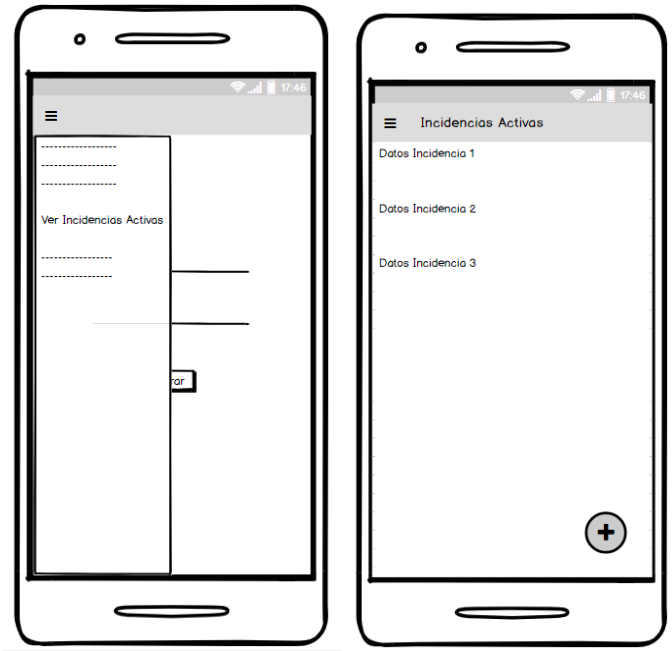
Caso de Uso: Cerrar Sesión

Título	Cerrar Sesión
Descripción	Permite cerrar sesión al usuario en la aplicación
Pre-condición	El usuario tiene la aplicación abierta y ha iniciado sesión.
Post-condición	El usuario se encuentra en la pantalla para el inicio de sesión con la sesión cerrada (Caso de Éxito)
Escenario principal	
1. El usuario abre el menú haciendo clic sobre el botón de menú desde cualquier pantalla de la aplicación. 2. El usuario selecciona la opción cerrar sesión. 3. El sistema borra los datos de sesión del usuario y realiza la desconexión de los distintos servicios a los que se conecta.	

Maquetas de interfaz


Caso de Uso: Listar Incidencias Activas

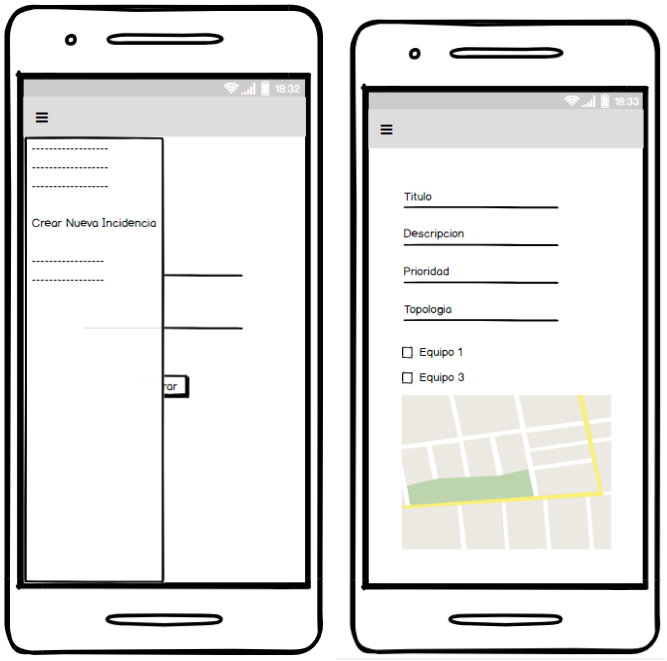
Título	Listar incidencias activas
Descripción	Permite ver a los administradores el listado de incidencias activas.
Pre-condición	El administrador se encuentra con la sesión iniciada.
Post-condición	El administrador visualiza el listado de incidencias activas (Caso de Éxito)
Escenario principal	
1. El usuario abre el menú haciendo clic sobre el botón de menú desde cualquier pantalla de la aplicación 2. El usuario selecciona la opción “Ver Incidencias Activas” 3. El sistema recupera el listado de incidencias activas y la muestra en pantalla	

Escenario alternativo
3.a El sistema no puede recuperar el listado de incidencias activas 3.a.1 El sistema muestra un mensaje de error
Maquetas de interfaz


Caso de Uso: Crear Incidencia Nueva

Título	Crear incidencia nueva
Descripción	Permite a los administradores crear una nueva incidencia.
Pre-condición	El administrador se encuentra con la sesión iniciada.
Post-condición	El sistema almacena correctamente la nueva incidencia y muestra un mensaje de éxito (Caso de Éxito)

Escenario principal
1. El administrador abre el menú haciendo clic sobre el botón de menú desde cualquier pantalla de la aplicación. 2. El usuario selecciona la opción “Crear Nueva Incidencia”. 3. El sistema muestra un formulario para introducir los datos de la nueva incidencia. 4. El usuario introduce los datos de la incidencia. 5. El usuario selecciona los equipos que participan en la incidencia. 6. a.1) El usuario hace una pulsación larga sobre el mapa mostrado. a.2) El sistema obtiene la dirección a partir del punto seleccionado. b.1) El usuario introduce la dirección en el campo destinado a ello. b.2) El usuario pulsa sobre el botón Obtener Coordenadas b.3) El sistema obtiene los datos de las coordenadas a partir de la dirección. 6. El usuario pulsa en el botón guardar. 7. El sistema almacena los datos de la incidencia en el sistema y avisa al servidor de señalización de la nueva incidencia.
Escenario alternativo
5-7.a El sistema no puede llevar a cabo la acción con éxito. 5-7.a.1 El sistema muestra un mensaje de error

Maquetas de interfaz


Caso de Uso: Ver Incidencia Activa

Título	Ver Incidencia Activa / Ver Incidencia Actual
Descripción	Permite a los administradores ver una incidencia activa o a los equipos ver su incidencia actual en caso de tenerla.
Pre-condición	El administrador se encuentra con la sesión iniciada o el equipo se encuentra con la sesión iniciada.
Post-condición	El sistema muestra la pantalla con las pestañas de Chat, Mapa e Info para la incidencia seleccionada.

Escenario principal

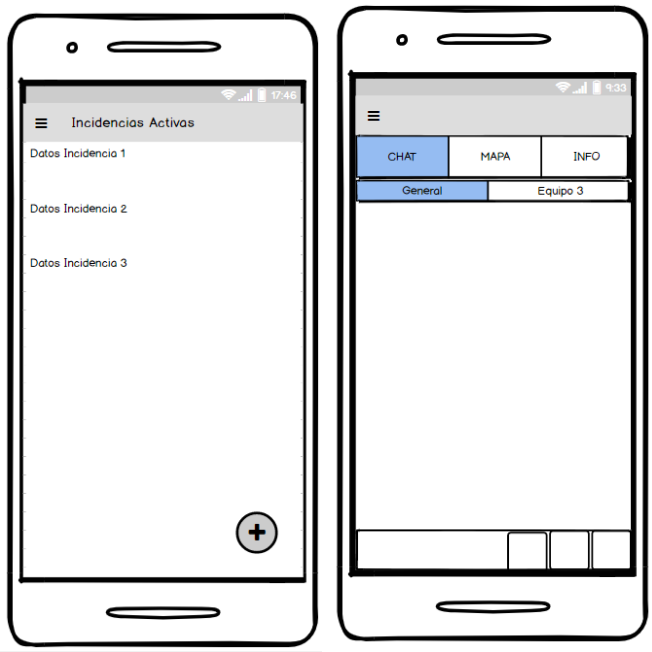
1. El administrador selecciona la incidencia en el listado de incidencias activas.
2. El sistema recupera la información de la incidencia y los usuarios conectados para la misma.
3. El sistema muestra la pantalla con una pestaña de Chat general y una por cada uno de los otros usuarios de la incidencia conectados

Escenario alternativo

1.b) El equipo carga la aplicación disponiendo de una incidencia asignada
Los dos pasos son iguales al caso de ser un administrador.

2. El sistema no puede cargar los datos de la incidencia
2.a.1 El sistema muestra un mensaje de error

Maquetas de Interfaz



Caso de Uso: Enviar Mensaje

En este único caso de uso, se van a detallar las acciones de envío de mensajes ya sea de texto, predefinido o de voz y a todos los usuarios o a uno exclusivamente, desglosándolo dentro de los escenarios.

Título	Enviar Mensaje
Descripción	Permite a los administradores y a los equipos enviar mensajes dentro del chat de la incidencia.
Pre-condición	El usuario se encuentra dentro de la ventana de incidencia activa.
Post-condición	El mensaje se ha enviado a los destinatarios y se ha añadido a la ventana de chat.
Escenario principal (Mensaje de Texto)	
1. El usuario selecciona la pestaña del chat al que quiere enviar el mensaje. 2. El usuario escribe el mensaje en la caja de texto. 3. El usuario pulsa el botón enviar. 4. El sistema envía el mensaje a los destinatarios. 5. El sistema muestra el mensaje enviado en la ventana de chat.	
Escenario alternativo 2.b) Mensaje de Audio 2.b.2 El usuario pulsa el botón del micrófono para enviar un audio. 2.b.3 El sistema muestra una ventana emergente y comienza a grabar. 2.b.4 El usuario habla mientras se está grabando el audio. 2.b.5 El usuario pulsa el botón enviar. 2.b.6 El sistema envía el mensaje a los destinatarios. 2.b.7 El sistema muestra el mensaje enviado en la ventana de chat.	

2.c) Mensaje de Texto predefinido

2.b.2 El usuario pulsa el botón del micrófono para enviar un audio.

2.b.3 El sistema muestra una ventana emergente con los mensajes predefinidos disponibles.

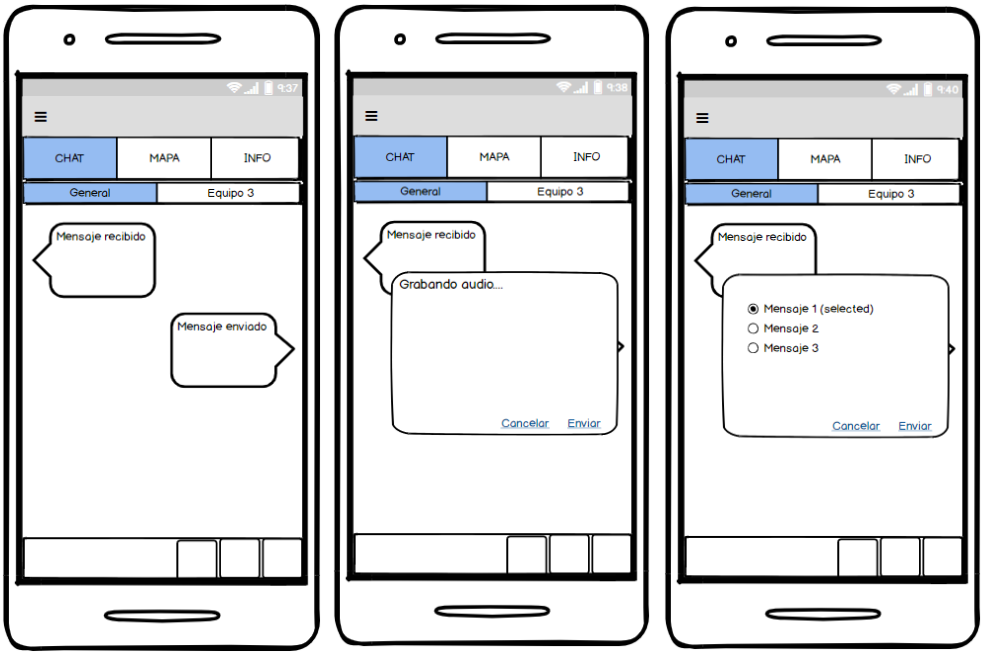
2.b.4 El usuario pulsa sobre el mensaje que desea enviar.

2.b.5 El usuario pulsa el botón enviar.

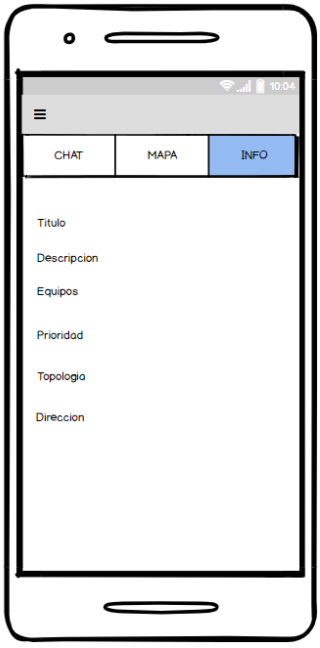
2.b.6 El sistema envía el mensaje a los destinatarios.

2.b.7 El sistema muestra el mensaje enviado en la ventana de chat.

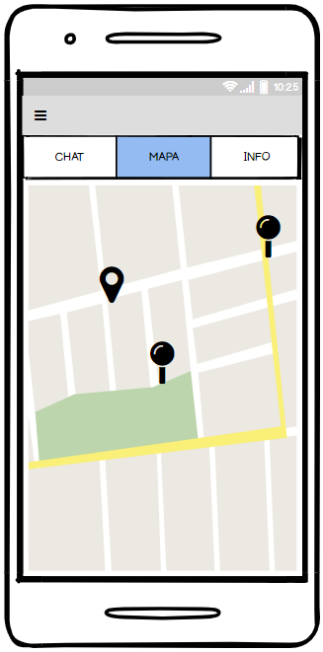
Maquetas de Interfaz



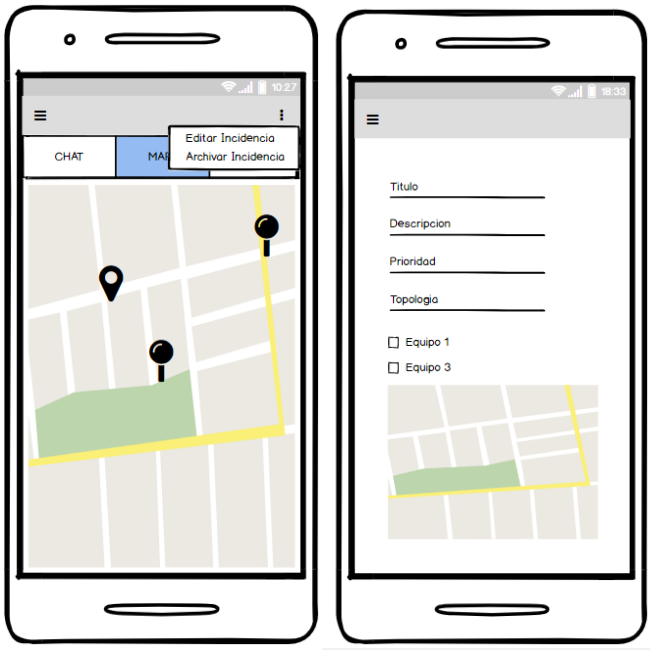
Caso de Uso: Ver Información de Incidencia Activa

Título	Ver Información de Incidencia Activa
Descripción	Permite a los administradores y a los equipos ver la información acerca de la incidencia activa.
Pre-condición	El usuario se encuentra dentro de la ventana de incidencia activa.
Post-condición	El sistema muestra la información de la incidencia.
Escenario principal	
1. El usuario selecciona la pestaña de INFO. 2. El sistema muestra el mensaje la información de la incidencia actual.	
Maquetas de Interfaz	
	

Caso de Uso: Ver posiciones de equipos en Incidencia

Título	Ver posiciones de equipos en Incidencia
Descripción	Permite a los administradores y a los equipos ver las posiciones de los equipos participantes en la incidencia.
Pre-condición	El usuario se encuentra dentro de la ventana de incidencia activa.
Post-condición	El sistema muestra una pantalla con las posiciones sobre un mapa.
Escenario principal	
1. El usuario selecciona la pestaña de MAPA. 2. El sistema muestra un mapa con la posición de la incidencia y las posiciones de los equipos en tiempo real.	
Maquetas de Interfaz	
	

Caso de Uso: Editar Incidencia Activa

Título	Editar Incidencia Activa
Descripción	Permite a un administrador editar una incidencia activa.
Pre-condición	El administrador se encuentra con la sesión iniciada en la pantalla de Ver Incidencia Activa.
Post-condición	El sistema muestra un mensaje de éxito tras archivar la incidencia (Caso de Éxito).
Escenario principal	
1. El usuario pulsa en el menú desplegable de la barra. 2. El usuario selecciona la opción Editar Incidencia. 3. El sistema muestra la pantalla con el formulario para editar la incidencia. 4. Se procede como a partir del paso 4 del caso de uso Crear Nueva Incidencia, modificando la instancia de Incidencia en el sistema en lugar de crear una nueva.	
Maquetas de interfaz	
	

Caso de Uso: Archivar Incidencia

Título	Archivar Incidencia
Descripción	Permite a un administrador archivar una incidencia.
Pre-condición	El administrador se encuentra con la sesión iniciada en la pantalla de Ver Incidencia Activa.
Post-condición	El sistema muestra un mensaje de éxito tras archivar la incidencia (Caso de Éxito).
Escenario principal	
1. El usuario pulsa en el menú desplegable de la barra y selecciona Archivar Incidencia. 2. El sistema muestra un dialogo de confirmación. 3. El usuario confirma su acción en el dialogo. 4. El sistema archiva la incidencia y muestra un mensaje de éxito.	
Escenario alternativo	
5.a El sistema no puede archivar la incidencia y muestra un mensaje de error	
Maquetas de interfaz	
	

Caso de Uso: Listar Incidencias Archivadas

Este caso de uso es prácticamente idéntico al de listar incidencias, únicamente cambia que el listado devuelto pero el sistema es el de incidencias archivadas.

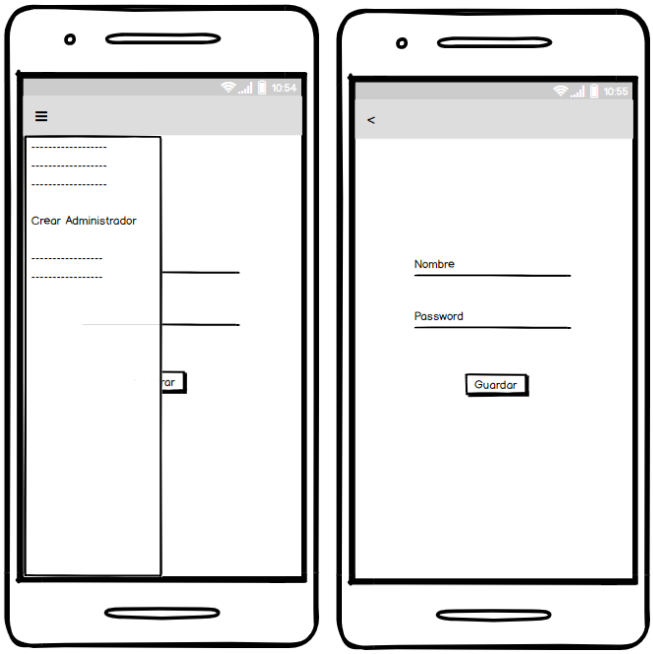
Caso de Uso: Ver Incidencia Archivada

Título	Ver Incidencia Archivada
Descripción	Permite a un administrador ver los datos de una incidencia archivada.
Pre-condición	El administrador se encuentra con la sesión iniciada en la pantalla de Incidencias Archivadas.
Post-condición	El sistema muestra la información de la incidencia archivada (Caso de Éxito).
Escenario principal	
1. El usuario pulsa sobre la incidencia deseada en el listado de incidencias. 2. El sistema muestra la pantalla con los datos de la incidencia.	
Maquetas de interfaz	
	

Los casos de uso referente al listado, creación, edición y borrado de administradores y equipos se van a agrupar ya que la funcionalidad es muy similar en ambos, por lo que se detallara una única vez cada acción.

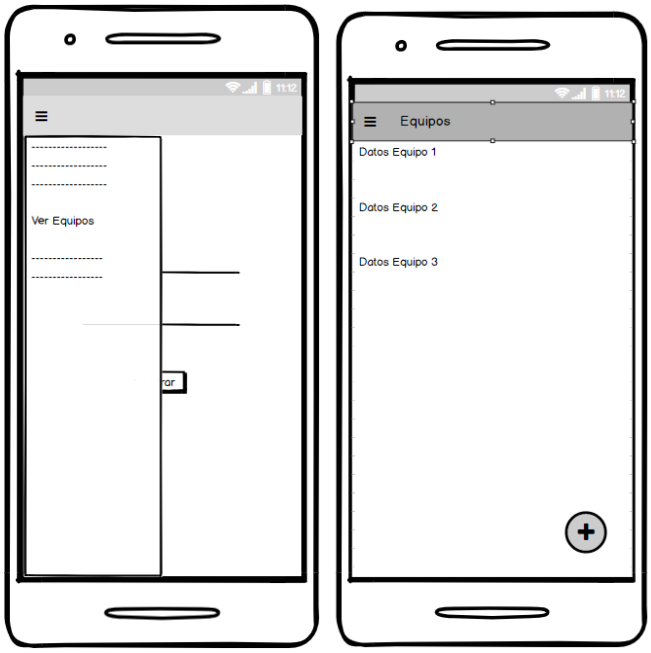
Caso de Uso: Crear Admin / Equipo

Título	Crear Admin / Equipo
Descripción	Permite a un administrador crear un administrador o un equipo.
Pre-condición	El administrador se encuentra con la sesión iniciada.
Post-condición	El sistema muestra un mensaje de éxito tras crear el usuario (Caso de Éxito).
Escenario principal	
1. El usuario pulsa en el menú y selecciona Crear Administrador / Crear Equipo. 2. El sistema muestra una pantalla con el formulario para introducir los datos. 3. El usuario introduce los datos del usuario. 4. El usuario pulsa el botón guardar. 5. El sistema guarda el usuario muestra un mensaje de éxito.	
Escenario alternativo	
5.a El sistema no puede guardar el usuario y muestra un mensaje de error	

Maquetas de interfaz


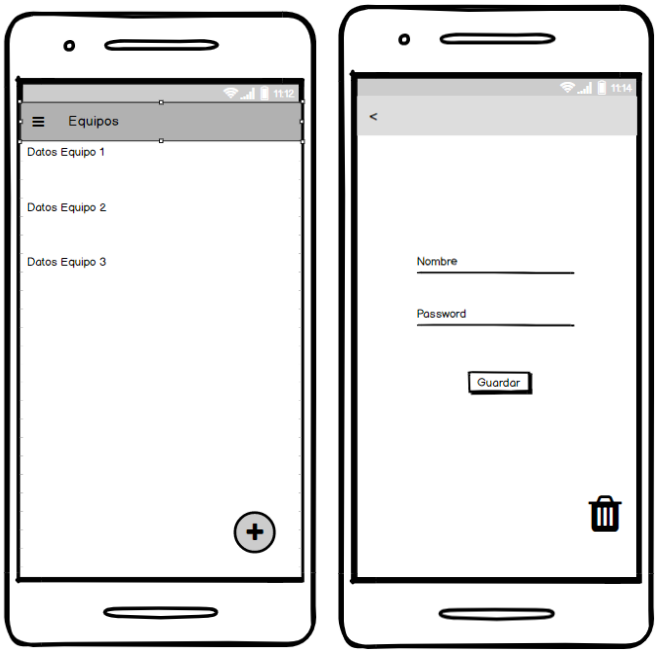
Caso de Uso: Listar Admins / Equipos

Título	Listar Admins / Equipos
Descripción	Permite a un administrador listar los administradores o equipos registrados en el sistema.
Pre-condición	El administrador se encuentra con la sesión iniciada.
Post-condición	El sistema muestra un listado con los administradores / equipos registrados (Caso de Éxito).
Escenario principal	
1. El usuario pulsa en el menú y selecciona Ver Administradores / Ver Incidencia 2. El sistema muestra un listado con los administradores / equipos registrados.	

Maquetas de interfaz


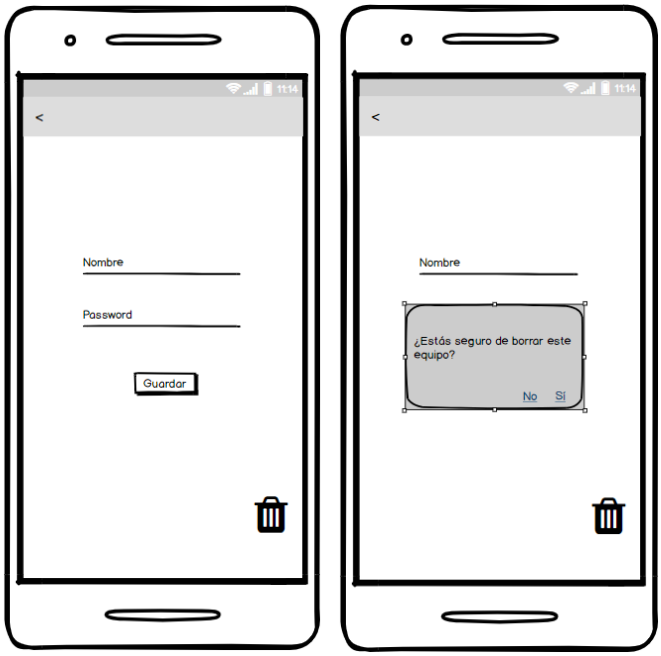
Caso de Uso: Editar Admin / Equipo

Título	Editar Admin / Equipo
Descripción	Permite a un administrador editar los datos de un usuario.
Pre-condición	El administrador se encuentra con la sesión iniciada en la pantalla de Ver Administradores / Equipos
Post-condición	El sistema ha guardado los cambios en el usuario (Caso de Éxito).
Escenario principal	
1. El usuario pulsa sobre uno de los elementos de la lista de administradores / equipos. 2. El sistema muestra el formulario con los datos del usuario. 3. A partir de aquí es el mismo comportamiento del Crear Admin / Equipo, simplemente modificando la instancia ya existente.	

Maquetas de interfaz


Caso de Uso: Borrar Admin / Equipo

Título	Borrar Admin / Equipo
Descripción	Permite a un administrador borrar un administrador o un equipo.
Pre-condición	El administrador se encuentra con la sesión iniciada en la pantalla de Editar Admin / Equipo.
Post-condición	El sistema muestra un mensaje de éxito tras borrar el usuario (Caso de Éxito).
Escenario principal	
1. El usuario pulsa sobre el botón de Borrar. 2. El sistema muestra un dialogo de confirmación. 3. El usuario confirma su acción en el dialogo. 4. El sistema borra el usuario y muestra un mensaje de éxito.	
Escenario alternativo	

5.a El sistema no borrar el usuario y muestra un mensaje de error
Maquetas de interfaz


3. Diseño e Implementación

En este apartado se van a proceder a detallar los procesos de diseño e implementación de la aplicación. Para ello se comenzará explicando el patrón de diseño utilizado, siendo en este caso MVVM (Model-View-ViewModel o Modelo-Vista-VistaModelo). A continuación, se detallan algunas de las librerías y componentes de Android que se usan en el software desarrollado, tales como RetroFit, OkHttp o Navigation entre otros, de manera que cuando más adelante se indique que se emplean, quede claro a qué nos estamos refiriendo. Finalmente, se detallarán las clases y métodos empleados en las distintas capas de la aplicación.

3.1. Patrón de diseño MVVM

El patrón de diseño por el que se ha optado a la hora de implementar esta aplicación ha sido el patrón MVVM, siglas que se refieren a Model-View-ViewModel, o en castellano, Modelo-Vista-VistaModelo.

En este patrón de diseño existen tres partes diferenciadas:

- Model (Modelo): Es la abstracción de los datos de la aplicación. En nuestro caso, estos datos se van a obtener a partir del servicio web que se nos proporcionó, haciendo uso de Retrofit como ya veremos más adelante. Con esta capa es con la que nuestras clases pertenecientes al *ViewModel* o Vista-Modelo, se comunicarán cuando se tengan que llevar a cabo operaciones de lectura o escritura de datos sobre el modelo.

- View (Vista): Este nivel se encarga de registrar las interacciones que realizan los usuarios con la interfaz gráfica, así como de mostrar la información recibida desde el *ViewModel.* En nuestra aplicación la vista estará compuesta por clases de Actividad (*Activity*) y Fragmentos (*Fragments)*, que como se detallará más adelante presentan una forma de organizar las vistas en las aplicaciones de Android.

- ViewModel (Vista-Modelo): Esta capa contendrá las clases que se encargarán de recibir las solicitudes que se producen como consecuencia de las acciones

que son llevadas a cabo por los usuarios sobre la vista y de comunicar a esta los datos recibidos desde el modelo.

En la figura 3.1.1 se muestra el diagrama creado por Hazem Saleh done se pueden ver las interacciones que se producen en este patrón de diseño.

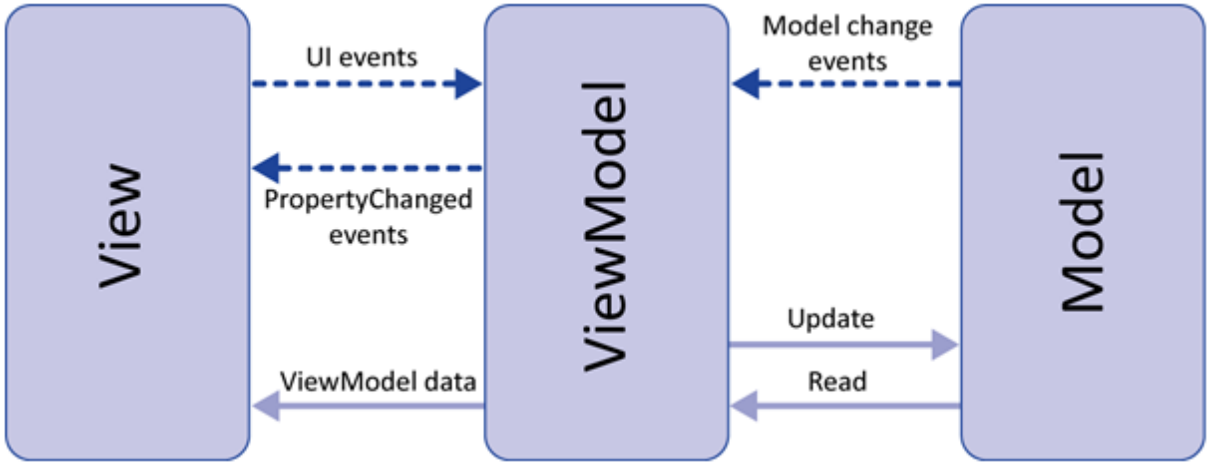


Figura 3.1.1. Saleh,H. (2017) Componentes e interacciones del patrón MVVM Recuperado de: https://proandroiddev.com/mvvm-architecture-viewmodel-and-livedata-part-1-604f50cda1

A la hora de usar este patrón en Android, intervienen dos clases fundamentales para el intercambio de información. La primera de ellas es *LiveData*, que es un componente de la arquitectura de Android que actúa como un contenedor de información observable. Esto es fundamental para el uso del servicio REST que nos devuelve la información de la base de datos de forma asíncrona, ya que junto al *Observer*, que veremos a continuación, nos permite reaccionar asíncronamente a la recepción de esta información. La interfaz *Observer*, que forma parte de la arquitectura de Android, nos permite definir los eventos a realizar cuando se produzca un cambio en el objeto *LiveData* que se está observando.

Este uso de *LiveData* y *Observer* se lleva a cabo en todas las interacciones de lectura y actualización de datos de nuestra aplicación por lo que en las figuras 3.2.2 y 3.2.3 se muestra la estructura del fragmento de código a emplear tanto en la clase perteneciente a la vista como a la que forma parte del *viewmodel* para llevar a cabo estos intercambios de datos. En el aparece una clase repositorio que se detallará cuando se explique el modelo de la aplicación.

```
viewModel.getAdmins().observe(this, new Observer<List<Admin>>() {
        @Override
        public void onChanged(@Nullable List<Admin> admins) {
            // Acciones a realizar sobre la vista tras recibir los datos
        }
    });
```

Figura 3.2.2. Fragmento de ejemplo de solicitud de datos desde la vista al vista-modelo

```
public LiveData<List<Admin>> getAdmins(){
    AdminRepository adminRep = new AdminRepository();
    return adminRep.getAdmins();
}
```

Figura 3.3.3. Fragmento de ejemplo de solicitud al modelo desde el vista-modelo.

En ocasiones será necesario realizar diferentes acciones en la vista-modelo antes de devolver los datos a la vista. Para ello se hace uso de la clase Transformations. En la figura 3.3.4 se muestra un ejemplo de cómo una vez recibida la información en el viewmodel se procede a realizar otra llamada asíncrona, de forma que de esta manera es como si tuviéramos un observador de los datos dentro del propio view-model

```
LiveData<Equipo> equipoCn = equipoRep.getEquipoByNombre(user);
        ok = Transformations.switchMap(equipoCn, equipo -> {
        // Para esperar a al respuesta del obtener por nombre
            if (equipoCn.getValue() == null){
           // Se hace una nueva petición al modelo
                Equipo newEquipo = new Equipo();
                LiveData<Boolean> equipoCr =
equipoRep.createEquipo(newEquipo);
                return Transformations.switchMap(equipoCr, ok_create -> {
                    // Acciones tras la segunda respuesta asincrona
                });
            }
            else{
                //
            }
        });
```

Figura 3.3.3. Fragmento de ejemplo del uso de Transformations para reaccionar desde el vista-modelo a un cambio en los datos

3.2. Librerías y componentes Android utilizados

En este apartado vamos a detallar algunas de los componentes y librerías que se han usado en nuestra aplicación y que son necesarios para elementos como el cliente del servicio REST, el cliente de websocket, los clientes WebRTC o el uso de mapas.

3.2.1. Retrofit

Retrofit es un cliente REST para Java y Android que nos va a facilitar la realización de peticiones al cliente web con el que vamos a obtener información y modificar la misma de la base de datos.

Para usarlo es necesario añadir las siguientes dependencias al archivo build.gradle:

- implementation ‘com.google.code.gson:gson:2.8.5’
- implementation ‘com.squareup.retrofit2:retrofit:2.4.0’
- implementation ‘com.squareup.retrofit2:converter-gson:2.3.0’
- implementation ‘com.squareup.okhttp3:okhttp:4.0.0-alpha02’

Para comunicarse con el servicio REST es necesario proporcionarle una interfaz con las llamadas existentes para cada uno de los *endpoints.* Estas interfaces, contendrán un método en cada una de ellas que tendrán la siguiente estructura:

- Anotación indicando el protocolo a usar, la ruta y los campos adicionales en casos de existir en la URL.
- Cabecera del método con el tipo de dato que es devuelto encapsulado en un objeto de la clase Call y con parámetros en el caso de existir.
- Los parámetros podrán ser:
    - @Path: Si hace referencia a algún campo de la URL.
    - @Body: Si se le debe pasar contenido en el cuerpo de la petición.
    - @Query: Si se le debe pasar algún parámetro en la URL.

Por ejemplo, el método para actualizar un administrador en la interfaz sería:

```
@PUT(“/api/Admins/{id}”)
   Call<Admin> updateAdmin(@Path(“id”) String id, @Body Admin admin);
```

Para realizar una llamada usando Retrofit, primero es necesario obtener una instancia del servicio a utilizar, tal y como se muestra en el código mostrado en la figura 3.2.1.1.

Todas estas operaciones se añadirán dentro de una clase ServiceGenerator que se explicará dentro del apartado modelo de la implementación.

```
Retrofit.Builder builder =
        new Retrofit.Builder()
            .baseUrl(BASE_URL)
.addConverterFactory(GsonConverterFactory.create());

retrofit = builder.build();

return retrofit.create(serviceClass);
```

Figura 3.2.1.1. Creación de servicio para invocar a las llamadas sobre el servidor REST.

3.2.2. OkHttp

OkHttp es un cliente HTTP para Java que nos va a facilitar a la hora de realizar la conexión al websocket del servidor de señalización. La dependencia ya la añadimos anteriormente ya que es usado también por Retrofit por lo que no sería necesario añadir nada más al archivo build.gradle.

La conexión al servidor de señalización se realizará desde la clase WebSocketClient y en la figura 3.2.2.1 se muestra el proceso a realizar para llevar a cabo la conexión.

```
WebSocket ws;
Request request = new
Request.Builder().url(BuildConfig.WSEndpoint).addHeader("Connection",
"close").build();
OkHttpClient client = new OkHttpClient();
ws = client.newWebSocket(request, listener);
client.dispatcher().executorService().shutdown();
```

Figura 3.2.2.1. Fragmento para la conexión al WebSocket

En la creación del WebSocket se le pasa un objeto pertenecerá a una clase que implementará a la clase WebSocketListener que definimos para controlar los eventos que se producen sobre el mismo, en nuestro caso será la clase MyWebSocketListener que se detallará en el Servicio posteriormente.

3.2.3. WebRTC

WebRTC es un proyecto de código abierto que permite la comunicación en tiempo real (*Real-Time Communications*) por medio de APIs sencillas. Para Android existe disponible una librería para implementar clientes WebRTC, para usarla es necesario añadir la dependencia:

- implementation ‘org.webrtc:google-webrtc:1.0.+’

En el diagrama de Vivek Chanddru mostrado en la figura 3.2.3.1 se observa el intercambio de mensajes necesario para establecer la comunicación WebRTC.

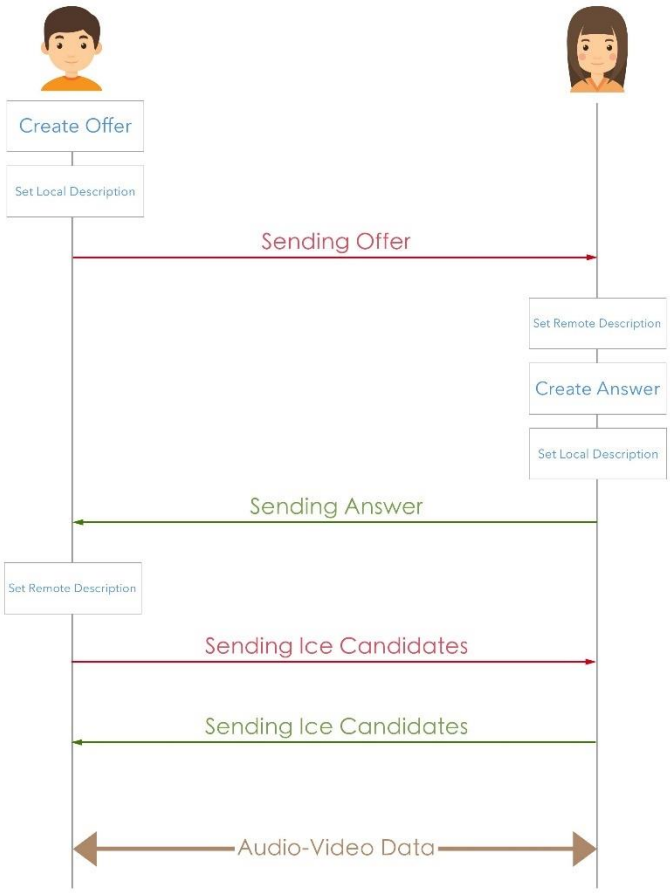


Figura 3.2.3.1. Intercambio de mensajes para el establecimiento de una conexión WebRTC (Chanddru, 2018)

Recuperado de https://vivekc.xyz/getting-started-with-webrtc-part-4-de72b58ab31e

Para la gestión de las conexiones WebRTC en nuestro proyecto se van a utilizar las clases:

- WebRTCClient
- PeerConnectionObserver
- DataChannelObserver

A continuación, se detalla la funcionalidad de cada una de ellas.

WebRTCClient

Esta clase va a ser la encargada de gestionar las conexiones WebRTC, para lo cual dispone de los siguientes miembros:

- ComClientService service: Una referencia al servicio Android de la aplicación para transmitirle los cambios producidos en la conexión y la llegada de mensajes.
- PeerConnectionFactory factory: Este objeto permitirá la creación de conexiones WebRTC
- Map<String, PeerConnection> conexiones: Un mapa que permitirá almacenar las conexiones establecidas con los distintos usuarios.
- Map<String, DataChannel> canales: Un mapa donde almacenamos los canales de datos con cada usuario para enviar mensajes.
- List<PeerConnection.IceServer> iceServers: Una lista de los servidores ICE que son necesarios para detectar las posibilidades de conexión.

Los principales métodos existentes en dicha clase serían:

- void initializeIceServers(): Su labor es la de inicializar el IceServer que se encargara de obtener los ICECandidates. Este IceServer precisa de un servidor STUN o TURN que se encargue de detectar las posibles rutas de conexión. Para el desarrollo de esta aplicación se ha utilizado el servidor STUN gratuito de Google stun:stun1.l.google.com:19302 con el cual se ha logrado la conexión exitosa de dos clientes bajo la misma red. En caso de existir problemas de conexión podría ser necesario el uso de un servidor TURN que es capaz de resolver rutas de conexión más complejas.

- PeerConnection createPeerConnection(String usuarioRemoto): Se encarga de crear la conexión; a la hora de crearla se le pasa como parámetro un objeto de la clase PeerConnectionObserver responsabe de gestionar los eventos recibidos a través de la conexión y que se detallará más adelante.

- void addUser(String name, Boolean starter, SessionDescription ses): Se encarga de invocar a la creación de la conexión para el usuario cuyo nombre se recibe como parámetro. Los dos parámetros restantes nos permiten detectar si es este usuario el que inicia el proceso de conexión o si por el contrario recibió el descriptor de sesión por parte de un usuario remoto.

  Según el caso en el que se encuentre tendrá que crear o no el canal de datos, ya que este solo se crea en este extremo si es el que inicia la conexión como se aprecia en el fragmento de código de la figura 3.2.3.2.

```
        if (starter) { // El canal lo crea el que inicia
            DataChannel dc = con.createDataChannel(name, new
DataChannel.Init());
            DataChannelObserver dco = new DataChannelObserver();
            dco.setCliente(this);
            dco.setRemoteUser(name);
            dco.setDataChannel(dc);
            dc.registerObserver(dco);
            canales.put(name, dc);
        }
```

Figura 3.2.3.2. Creación del canal de datos

En el caso de ser el extremo que inicia la conexión deberá enviar el mensaje *offer* al otro extremo para lo cual se crea el descriptor de sesión y se envía al servicio para que lo transmita. Este proceso queda reflejado en el fragmento de la figura 3.2.3.3.

En caso contrario, se ha recibido el “offer” del usuario remoto, por lo que se almacenará el descriptor de sesión y se genera el propio para mandárselo

como mensaje “answer” a través del servicio. Se refleja este proceso en la figura 3.2.3.4

```
        con.createOffer(new SdpObserver() {
                @Override
                public void onCreateSuccess(SessionDescription
sessionDescription) {
                    con.setLocalDescription(new SdpObserver() {
                        ...
                    }, sessionDescription);
                    WebRTCClient.this.service.enviarWsOffer(name,
sessionDescription);
                }
```

Figura 3.2.3.3. Creación del descriptor de sesión y envío a través del servicio

```
        con.setRemoteDescription(new SdpObserver() {
                ...
            }, ses);
            con.createAnswer(new SdpObserver() {
                @Override
                public void onCreateSuccess(SessionDescription
sessionDescription) {
                    con.setLocalDescription(new SdpObserver() {
                       ...
                    }, sessionDescription);
                    WebRTCClient.this.service.enviarWsAnswer(name,
sessionDescription);
                }
                ...
            }, sdpMediaConstraints);
```

Figura 3.2.3.4. Recepción del descriptor de sesión remoto y creación del propio para la respuesta en el mensaje “answer”.

- void removeUser(String name): Se encarga de cerrar la conexión y el canal de datos con este usuario para cuando el usuario remoto se desconecte.
- void enviarMensajeTexto(List<String> destinatarios, String mensaje, String chatId, String incidencia)
  void enviarMensajeAudio(List<String> destinatarios, byte[] audio, String chatId, String incidencia): Estos dos métodos se van a encargar de enviar el mensaje a una lista de destinatarios a través del canal de datos. En la figura 3.2.3.5 se muestra cómo se compondrá el mensaje JSON para el envío de un mensaje de texto. Para el caso de un mensaje de audio se envían dos mensajes, el primero con los datos de origen del mensaje y el segundo con el contenido binario del audio.

```
JSONObject json = new JSONObject();
json.put("blob", false);
json.put("chat", chatId);
json.put("origen", this.service.getUser());
json.put("mensaje", mensaje);
json.put("incidencia", incidencia);
Log.i("INFO","Destinatarios: "+destinatarios);
for(String destinatario : destinatarios){
    DataChannel channel = canales.get(destinatario);
    ByteBuffer data =
ByteBuffer.wrap((json.toString().getBytes(Charset.defaultCharset())));
    channel.send(new DataChannel.Buffer(data, false));
}
```

Figura 3.2.3.5. Envío de mensaje a través del canal de datos

- void mensajeTextoRecibido(String remitente, String mensaje, String chatId, String incidenciaId)

void mensajeAudioRecibido(String remitente, byte[] audio, String chatId, String incidenciaId): Estos dos métodos se usan para transmitir los mensajes recibidos al servicio, serán invocados desde el DataChannelObserver que se explica a continuación.

PeerConnectionObserver

Esta clase implementa la interfaz PeerConnection.Observer y de ella reimplementados dos métodos:

- void onIceCandidate(IceCandidate iceCandidate): Cuando se ha obtenido un IceCandidate se reenvía al usuario remoto para que conozca que es una opción de conexión.
- void onDataChannel(DataChannel dataChannel): Este método se ejecuta cuando se recibe un canal de datos por parte del usuario que inició la conexión; se almacena como canal de datos asociado a ese usuario remoto.

DataChannelObserver

Esta clase implementa la interfaz DataChannel.Observer y en ella redefinimos el método encargado de procesar el mensaje recibido a través del canal de comunicación.

- void onMessage(DataChannel.Buffer buffer): Se encarga de procesar el mensaje recibido a través del canal de comunicación. Una vez leído y compr56irec el tipo de mensaje, se le comunica al WebRTCClient su recepción para que este a su vez lo transmita al servicio.

3.2.4. Navigation

Navigation es un componente de la arquitectura de Android que facilita la construcción y la transición entre las diferentes pantallas que componen una aplicación Android. De acuerdo a Google, entre los beneficios más destacados que obtenemos al usar este componente para construir las vistas se encuentran:

- Gestiona automáticamente la transición entre fragmentos.
- Permite el enlace directo a un fragmento concreto de la aplicación, aspecto que resulta especialmente útil para los menús.
- Permite la transmisión de información en las transiciones.

- Es posible visualizar y editar el flujo de navegación de nuestra aplicación desde Android Studio.

Para usar este componente es necesario añadir las siguiente dependencias:

- implementation ‘android.arch.navigation:navigation-fragment:1.0.0-alpha07’
- implementation ‘android.arch.navigation:navigation-ui:1.0.0-alpha07’

A la hora de diseñar las vistas para usar la navegación se debe definir un contenedor de tipo NavHostFragment como el que se observa en la figura 3.2.4.1.

```
<fragment
        android:id="@+id/nav_host_admin"

android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:navGraph="@navigation/admin_client_navigation"
/>="@navigation/admin_client_navigation" />
```

Figura 3.2.4.1. Elemento a añadir en el layout de la actividad en la que usaremos Navigation.

En este elemento se le indica en el atributo app:navGraph cuál va a ser el mapa de navegación referente a este contenedor. Ese mapa de navegación se crea desde el mismo Android Studio y a él se le van añadiendo las vistas y las transiciones entre fragmentos que reflejen el funcionamiento de nuestra aplicación. En la figura 3.2.4.2 se observa el mapa de navegación para la Actividad de Login que posteriormente se detallará.

A la hora de realizar transiciones se hará uso de las acciones que se hayan definido en el mapa de navegación y del elemento NavController. En la figura 3.2.4.3 se observa por ejemplo el código a emplear para transitar de un fragmento a una actividad.

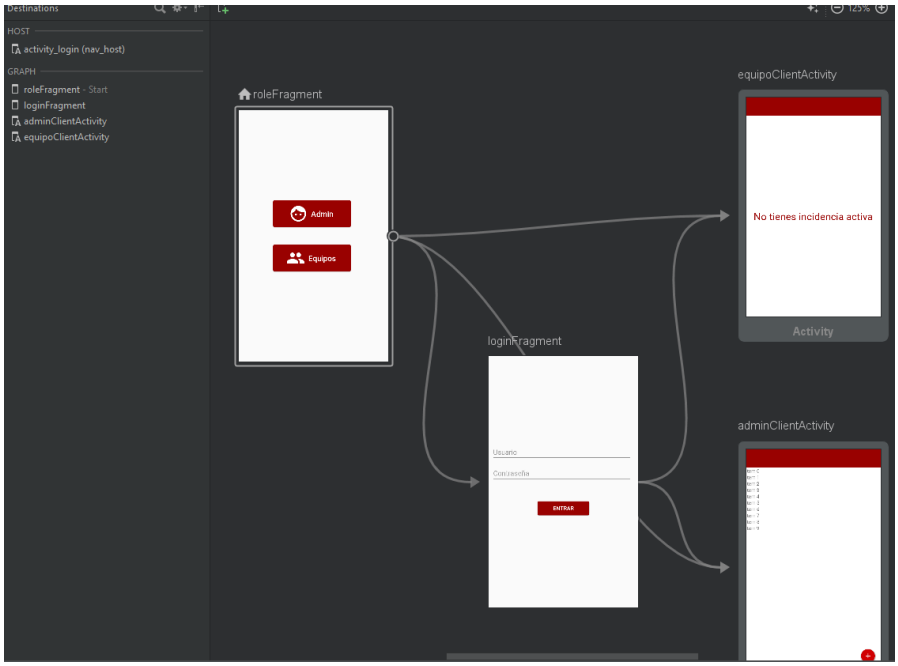


Figura 3.2.4.2. Mapa de navegación para la actividad de login.

```
LoginFragmentDirections.ActionLoginFragmentToAdminClientActivity
action =
LoginFragmentDirections.actionLoginFragmentToAdminClientActivity();
                    Navigation.findNavController(view).navigate(action);
```

Figura 3.2.4.3. Transición entre fragmento y actividad haciendo uso de NavController

También es posible enviar información en las transiciones; para ello se definen los argumentos de entrada en el mapa de navegación antes citado y se le pasa como parámetro a la acción.

3.2.5. Google Maps

En nuestra aplicación también usamos la librería de Google Maps, tanto para mostrar mapas como para obtener la localización de una dirección u obtener la dirección a partir de una posición en el mapa.

Para poder usar esta librería añadimos la siguiente dependencia:

- implementation ‘com.google.android.gms:play-services-location:16.0.0’
- implementation ‘com.google.android.gms:play-services-maps:16.0.0’

Con el objetivo de simplificar el uso de los mapas en diferentes pantallas de nuestra aplicación se ha definido un fragmento MapEmbeddedFragment que contiene los métodos necesarios para integrar un mapa en las vistas y realizar las acciones antes citadas. Los métodos más destacados de esta clase son:

- void inicializarMapa(): Se encarga de inicializar el mapa en la vista; en la figura 3.2.5.1 se refleja el fragmento de código necesario para que no existan problemas cuando se recarga una vista. Cuando se inicialice se llamará asíncronamente a onMapReady que será una función que se implementará en cada fragmento que herede de éste.

```
FragmentManager fm = getChildFragmentManager();
        markersPosiciones = new HashMap<>();
        mapFragment = (SupportMapFragment)
fm.findFragmentByTag("mapFragment");
        if (mapFragment == null) {
            mapFragment = new SupportMapFragment();
            FragmentTransaction ft = fm.beginTransaction();
            ft.add(R.id.layout_mapa, mapFragment, "mapFragment");
            ft.commit();
            fm.executePendingTransactions();
        }
        mapFragment.getMapAsync(this);
```

Figura 3.2.5.1. Fragmento para la inicialización de un mapa incrustado en una vista

- void onMapLongClick(LatLng latLng): Este método añadirá un Marker en la posición del mapa donde se haya realizado una pulsación larga y obtendrá la dirección de dicha posición como se necesita en el añadir y editar incidencia.
- void establecerPosicionMapa(Double lat, Double lng): Centra el mapa en la posición indicada añadiendo un marker en dicha ubicación, este método se usará para marcar la posición de la incidencia cuando se vea la incidencia activa.
- void obtenerCoordenadasDireccion(String 59irección)
  void obtenerDireccionCoordenadas(LatLng latLng): Estos dos métodos hacen uso de la clase GeoCoder de la librería de GoogleMaps para obtener las

coordenadas y la dirección respectivamente a partir del dato del que se disponga.

- void updateMarker(String usuario, LatLng latLng): En este método se implementa la funcionalidad necesaria para actualizar el marker de un equipo de acuerdo a la posición recibida.

3.3. Desglose de la implementación

En este apartado se van a detallar los elementos más destacados de los diferentes niveles de nuestra aplicación.

3.3.1. Servicio

En la aplicación se ha implementado la clase ComClientService que es el punto central de nuestro software. Se trata de un servidor que, con el objetivo de que no sea cerrado por el sistema y pueda seguir recibiendo eventos, se ha declarado como servicio de primer plano o *foreground service*. Esto se consigue mediante el código mostrado en la figura 3.3.1.1, en la cual se observa que es necesario asignarle una notificación que quedará fija mientras el servicio esté en ejecución.

```
Notification notification = new NotificationCompat.Builder(this,
"Notifications")
        .setContentTitle("TFGAppBomberos conectado al servidor")
        .setSmallIcon(R.mipmap.ic_launcher)
        .build();
startForeground(1, notification);
```

Figura 3.3.1.1. Inicio del servicio como servicio de primer plano

Este servicio contendrá una instancia de WebRTCClient y otra de WebSocket con las que gestionará las comunicaciones entre los clientes WebRTC y con el servidor de señalización, habiendo explicado ya ambos componentes previamente en esta memoria.

Además contendrá una instancia de la clase LocationHelper. Esta clase es una clase auxiliar que se ha implementado para obtener la localización de los equipos en tiempo real. Para ello, en el constructor de la clase LocationHelper se invocará a su método

getDeviceLocation, en el que se configurará la recepción periódica de la ubicación mediante el código mostrado en la figura 3.3.1.2.

```
locationRequest = new LocationRequest();
locationRequest.setInterval(5000);
locationRequest.setFastestInterval(5000);
locationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_P
OWER_ACCURACY);
locationCallback = new LocationCallback() {
public void onLocationResult(LocationResult locationResult) {
        …
};
};
mFusedLocationProviderClient.requestLocationUpdates(locationRequest,
locationCallback , null);
```

Figura 3.3.1.2. Esqueleto del código para configurar la recepción periódica de la ubicación

Este servicio va a actuar como *listener* de los eventos que se produzcan tanto en el cliente del servidor de señalización como en el de WebRTC y va a actuar de intermediario entre ellos y el resto de componentes de nuestra aplicación. Para ello, se definen métodos para cada uno de los eventos que vimos que pueden existir en el servidor de señalización, tanto para enviar como para gestionar su recepción: login, update, offer, answer, createIncidencia, deleteIncidencia y leave. En este servicio dispondremos de un mapa con instancias de la clase auxiliar SalaIncidencia. Esta clase almacena, tanto las coordenadas como los mensajes intercambiados por cada incidencia activa.

Consideramos especialmente destacar en esta memoria el método onWsLogin, que será invocado después de recibir la respuesta al mensaje de inicio de sesión enviado al servidor.

A partir de la respuesta obtenida en onWsLogin, obtenemos los usuarios que hay conectados en ese momento en el servidor de señalización lo que permitirá deducir, cuáles son las incidencias creadas en ese servidor, ya que no tenemos garantía de

que existan en él al no recuperarlos el mismo del sistema en caso de reiniciarse. Para ello definimos dos métodos, crearSalasParaAdmin y crearSalasParaEquipo, que se encargarán tanto de crear las instancias de SalaIncidencia a usar por nuestra app como de crearlas en el servidor de señalización si no hay garantía de su existencia.
Otro aspecto importante que se ha implementado en ese método es establecer las conexiones necesarias con cada uno de los usuarios, para lo cual usamos el método addUser de la clase WebRTCClient explicado anteriormente.
Como respuesta a mensajes de offer, answer y candidate del servidor de señalización en esta clase se invocará a los métodos de la clase WebRTCClient correspondiente, para lograr el intercambio de mensajes necesario para establecer la conexión. También en esta clase servicio disponemos de los métodos necesarios tanto para invocar el envío de mensajes sobre las instancias de WebRTCClient como para trasladar el mensaje recibido a través de la misma a nuestras actividades, mostrando una notificación en caso de que la actividad no se encuentre en primer plano. Este comportamiento entendemos adecuado mostrarlo en la figura 3.3.1.3
En ese fragmento podemos observar el uso de sendBroadcast que se utilizará para emitir los mensajes recibidos en la actividad por parte del broadcastReceiver que se explicará más adelante.
Por último, cabe destacar que en este servicio también se han implementado diversos métodos encargados de codificar en JSON objetos referentes al cliente webrtc, como es el caso de los descriptores de sesión y de los icecandidates, además de definir otros métodos auxiliares que permiten obtener información acerca de los usuarios conectados.

```
if("admin".equals(role)){
        activityFg =
PreferenceManager.getDefaultSharedPreferences(ComClientService.this.getAp
plicationContext()).getBoolean("isAdminActive",false);
}
else{
        activityFg =
PreferenceManager.getDefaultSharedPreferences(ComClientService.this.getAp
plicationContext()).getBoolean("isEquipoActive",false);
}

if (activityFg){
        // Si la actividad esta en primer plano
        Intent intent = new Intent("messageReceived");
        intent.putExtra("message", mes);
        intent.putExtra("incidencia", incidenciaId);
        intent.putExtra("chatId", chatId);
        sendBroadcast(intent);
}
```

Figura 3.3.1.3. Procesamiento de mensaje recibido

3.3.2. Modelo

Como ya se ha comentado con anterioridad, los datos con los que va a trabajar nuestra aplicación Android se van a obtener a través de un servicio REST.

Por un lado, tendremos una clase por cada una de las colecciones, es decir, existe una clase Admin, una Equipo y una Incidencia, de acuerdo al modelo de datos que explicamos en el apartado referente al patrón de diseño. Para componer estas clases a partir de la respuesta JSON que proporciona el servicio REST, se ha empleado la plataforma jsonschema2pojo que nos permite obtener las clases Java anotadas para cada una de nuestras colecciones.

Se ha optado por usar una clase ServiceGenerator como la recomendada por Norman Peitek en su artículo “Retrofit 2 — Creating a Sustainable Android Client”. En esta clase se añade la creación del servicio Retrofit como se explicó en el apartado

referente a dicha librería. Para Retrofit, como dijimos anteriormente, es necesario definir cuáles son las llamadas a realizar para cada *endpoint*, como referencia, se detallan a continuación las de la clase AdminService, siendo similares para EquipoService e IncidenciaService

- Admin:
    - @GET(“/api/Admins”)
      Call<List<Admin>> getAdmins();
    - @GET(“/api/Admins/{id}”)
      Call<Admin> getAdmin(@Path(“id”) String id);
    - @POST(“/api/Admins”)
      Call<Admin> createAdmin(@Body Admin admin);
    - @PUT(“/api/Admins/{id}”)
      Call<Admin> updateAdmin(@Path(“id”) String id, @Body Admin admin);
    - @DELETE(“/api/Admins/{id}”)
      Call<Admin> deleteAdmin(@Path(“id”) String id);
    - @GET(“/api/Admins/findOne”)
      Call<Admin> findAdmin(@Query(“filter”) String filter);

Finalmente, por cada elemento de nuestro modelo tendremos una clase Repository (AdminRepository, EquipoRepository e IncidenciaRepository), que será la que desde los view-models se llamará para realizar las operaciones sobre el modelo de datos, como por ejemplo la llamada a getAdmins que se citó al explicar el patrón MVVM y que se expone en la figura 3.3.1.1. De esta manera se consigue un nivel de abstracción bastante elevado respecto nuestra capa de modelo.

```
public LiveData<List<Admin>> getAdmins(){
      final MutableLiveData<List<Admin>> data = new
MutableLiveData<>();
      service.getAdmins().enqueue(new Callback<List<Admin>>() {
         @Override
         public void onResponse(Call<List<Admin>> call,
Response<List<Admin>> response) {
                  …
         }
         @Override
         public void onFailure(Call<List<Admin>> call, Throwable t) {
               data.setValue(null);
         }
      });
      return data;
   }
```

Figura 3.3.1.1. Ejemplo de uso de método en clase Repository

3.3.2. Vistas

Se ha descompuesto nuestra aplicación en tres actividades fundamentales: LoginActivity, AdminClientActivity y EquipoClientActivity. Estas dos últimas actividades debido a que comparten parte de su comportamiento como es el caso de la conexión con el servicio de nuestra aplicación, heredan de una clase ClientActivity por lo que se explicará que comportamiento contiene también esta.

LoginActivity

En esta actividad intervienen dos fragmentos, como se aprecia en el mapa de navegación de la figura 3.2.4.2:

- **RoleFragment**: En esta vista se muestran los dos botones para elegir el tipo de usuario y se gestiona la transición hacia el siguiente fragmento.

- **LoginFragment**: Se encarga de mostrar el formulario de login y se comunicará con el LoginViewModel para realizar el inicio de sesión y transicionar a la actividad adecuada en caso de éxito.

ClientActivity

Esta actividad se encargará de inicializar el servicio de nuestra aplicación. Ello se realiza de la forma que se observa en la figura 3.3.2.1.

```
Intent intent = new Intent(this, ComClientService.class);
intent.putExtra("user",this.getLoggedUser());
intent.putExtra("role", this.role);
ontextCompat.startForegroundService(this, intent);
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

Figura 3.3.2.1. Inicialización de servicio desde actividad

La comunicación entre la actividad y el servicio se realiza mediante dos mecanismos. El primero para obtener una referencia del servicio y poder invocar a métodos contenidos en él es el que se lleva a cabo mediante el método bindService, que recibe como parámetro el elemento mConnection que es una instancia de ServiceConnection, una clase que nos permite detectar cuando el servicio se ha conectado. Cuando en ella se recibe el evento de conexión podemos obtener la referencia a la instancia del servicio mediante nuestro método linkService como se refleja en la figura 3.3.2.2.

```
ComClientService.ComClientBinder binder =
(ComClientService.ComClientBinder) service;
comClientService = binder.getService();
serviceConnected = true;
registarReceptorBroadcast();
```

Figura 3.3.2.2. Obtención de la referencia al servicio

El otro mecanismo para comunicarse con el servicio es necesario para recibir los eventos de este y se lleva a cabo mediante un BroadcastReceiver que recibirá los eventos lanzados que se explicaron para el servicio y que se registra como se ve en la figura 3.3.2.3.

```
broadcastReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if (action != null) {
                procesarAccion(action, intent);
            }
        }
    };
    IntentFilter filter = new IntentFilter();
    filter.addAction("salasCreadas");
    filter.addAction("locationUpdate");
    filter.addAction("messageReceived");
    registerReceiver(broadcastReceiver, filter);
```

Figura 3.3.2.3. Creación del broadcast receiver para obtener los mensajes del servicio

En esta actividad también existen métodos para comunicarse con el servicio y que serán invocados en función de la acción a realizar, ejemplo de estos métodos son: enviarMensaje, enviarAudio, enviarBorrarIncidencia, getMensajesIncidenciao getUsuariosConectadosIncidencia.

AdminClientActivity y EquipoClientActivity

En estas dos actividades se define el método procesarAccion que vimos anteriormente en el BroadcastReceiver y que actuarán respondiendo a los eventos llevando a cabo la acción deseada.

En el mapa de navegación de la figura 3.2.3.4 se observan los fragmentos involucrados en la actividad del administrador, existiendo básicamente uno de ellos por cada uno de los casos de uso detallados anteriormente.

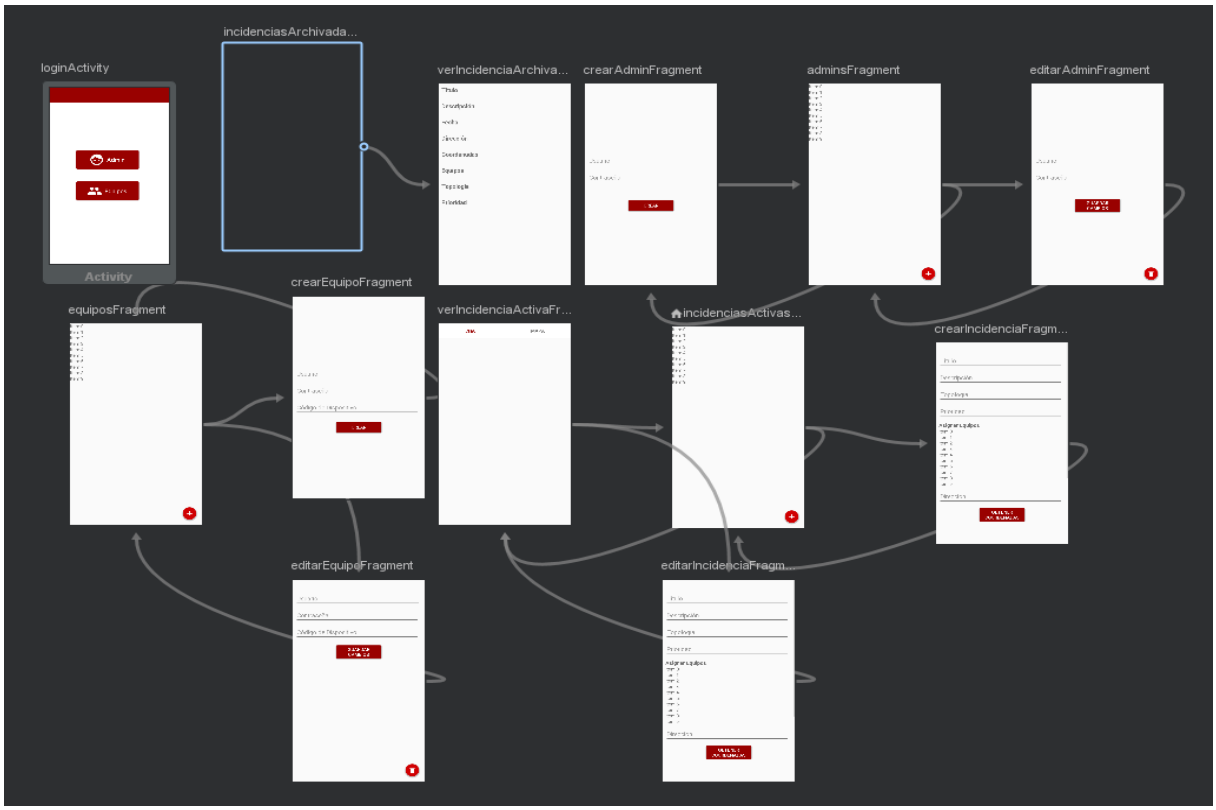


Figura 3.2.3.4. Mapa de navegación para la actividad del administrador

Por su parte en la figura 3.2.3.5 se observan los fragmentos de la actividad de Equipos siendo ésta como se observa mucho más simple.

Estos fragmentos tendrán una referencia al vista-modelo correspondiente, realizando las llamadas a los métodos que modificarán nuestro modelo. En el caso de los fragmentos que se encargan de ver una incidencia activa en caso de los administradores y de ver la incidencia asignada en caso de equipo con alguna asignada, ambos fragmentos heredan de IncidenciaFragment, siendo en el dónde se implementa la lógica necesaria para el correcto funcionamiento de estas pantallas.

A nivel de implementación de las vistas consideramos importante destacar el uso de los PagerAdapter en el caso de la pantalla de Incidencia Activa. Para ello se han declarado las clases IncidenciaPagerAdapter y ChatsPagerAdapter, que permitirán controlar el contenido que se muestra en las vistas cuando se selecciona una u otra de las pestañas existentes en esta vista, que, como se observó en los casos de uso,

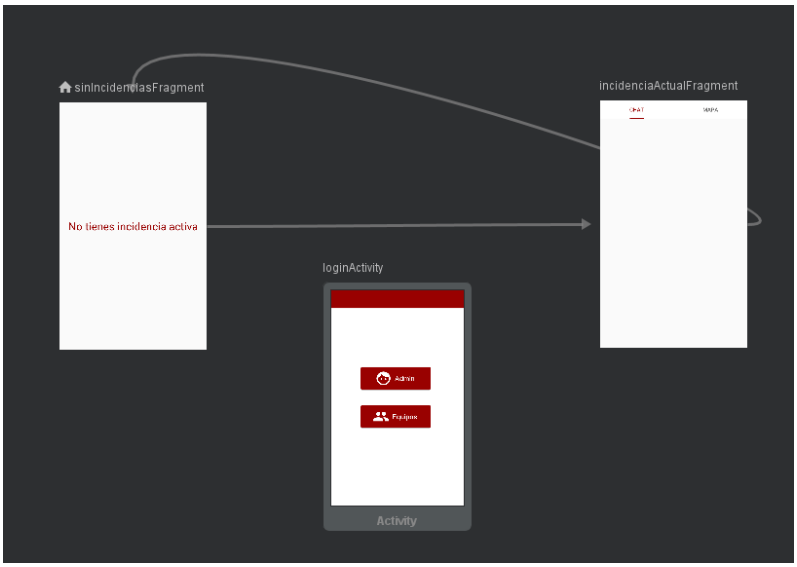


Figura 3.2.3.5 Mapa de navegación para la actividad del equipo

consta de dos niveles, el primero de ellos permite elegir si mostrar el chat, el mapa con los equipos o la información de la incidencia, y el segundo de ellos, en caso de encontrarnos en la pestaña chat, permite movernos por las conversaciones. Para controlar este comportamiento estas clases heredan de PagerAdapter y redefinen los siguientes métodos:

- int getCount(): Devuelve el número de pestañas existentes
- Object instantiateItem(ViewGroup container, final int position): El cual contendrá la lógica para inicializar la vista del contenido de cada una de las pestañas.

Por último, dentro de apartado de las vistas se considera reseñable destacar la grabación y reproducción de audio para los mensajes de voz, cuyos códigos de ejemplo se pueden observar en las figuras 3.2.3.6 y 3.2.3.7

```
File tempAudio = File.createTempFile("tmpAppBomb",".tmp");
tempAudio.deleteOnExit();
recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
recorder.setOutputFile(tempAudio.getAbsolutePath());
recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
recorder.prepare();
recorder.start();
recorder.stop();
recorder.reset();
```

Figura 3.2.3.6 Grabación de audio desde la aplicación

```
File tempAudio = File.createTempFile("tmpAppBomb",".tmp");
tempAudio.deleteOnExit();
FileOutputStream fos = new FileOutputStream(tempAudio);
fos.write(mes.getAudio());
fos.close();

FileInputStream is = new FileInputStream(tempAudio);
FileDescriptor fd = is.getFD();
MediaPlayer mp = adapter.getMediaPlayer();
mp.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
        @Override
        public void onCompletion(MediaPlayer mediaPlayer) {
                mp.reset();
                botonPlay.setVisibility(View.VISIBLE);
                botonStop.setVisibility(View.INVISIBLE);
        }
});
mp.setDataSource(fd);
mp.prepare();
mp.start();
```

Figura 3.2.3.7. Reproducción de Audio desde la aplicación

3.3.3. Vista-Modelos

Por cada uno de los fragmentos anteriores se ha definido una clase vista-modelo, que se encargará de realizar las llamadas a los repositorios como se ha explicado en el apartado de modelo, por lo que se considera redundante reproducir aquí de nuevo el código para alguno de ellos.

Sí consideramos importante destacar la clase SharedViewModel, cuya instancia será compartida entre la actividad y el fragmento de VerIncidenciaActiva, tanto en la actividad de cliente de administrador como en el cliente de los equipos. Estos es así ya que en esta clase se almacenarán las coordenadas y los mensajes recibidos desde el servicio, de forma que desde el propio fragmento se puedan obtener. Para ello se disponen de dos propiedades y de diferentes métodos para obtener y modificar estos datos, siendo las propiedades las siguientes:

- Map<String,Map<String,List<MensajeChat>>> chatMessages;
- Map<String, Map<String, LatLng>> coordinates;

Con esto entendemos cubierto el apartado de explicar en detalle los componentes de la implementación que consideramos más importantes destacar y es momento de finalizar con un apartado referente a conclusiones.

4. Conclusiones y Líneas Futuras

Con la aplicación desarrollada entendemos que quedan satisfechas las necesidades y los objetivos con los cuales se ha llevado a cabo este trabajo fin de grado. A través de su uso, se permite a los equipos de bomberos gestionar las incidencias y la intervención en las mismas desde dos aspectos: por un lado, la administración de usuarios y equipos a través de formularios para su creación, modificación y archivado y por el otro la comunicación en tiempo real en el momento de la intervención en las incidencias entre administradores y equipos participantes, ya sea a través de mensajes de texto, con la posibilidad de disponer de mensajes predefinidos, como a través de mensajes de audio, todo ello acompañado de la capacidad de observar la ubicación de los equipos haciendo uso de su posición que es transmitida a través de la propia aplicación.

Por medio de su realización, se ha alcanzado un conocimiento bastante profundo del desarrollo de aplicaciones Android, considerando la variedad de componentes empleados y la necesidad de lograr una integración entre todos ellos.

Entendemos que hay algunos apartados en los que sería posible incorporar mejoras al sistema, como sería el caso de almacenar el contenido de las conversaciones en archivos locales o permitir la transmisión de flujos de audio o video así como el envío de archivos y que podría ser interesante en futuros trabajos sobre el mismo. Además, también se considera que resultaría apropiado el desarrollo de pruebas unitarias y de integración sobre el software desarrollado, que debido a la complejidad de la aplicación queda fuera del ámbito de este trabajo fin de grado.

Bibliografía

- Android Open Source Project (19 de abril de 2018) Introducción a Android. Recuperado de https://developer.android.com/guide/
- Android Open Source Project (3 de julio de 2018) Android Jetpack. Recuperado de https://developer.android.com/jetpack/
- Android | WebRTC (n.d) Recuperado de https://webrtc.org/native-code/android/
- Chanddru, V. (4 de febrero de 2018) Getting Started with WebRTC for Android— Develop video call app easily! Recuperado de https://vivekc.xyz/getting-started-with-webrtc-part-4-de72b58ab31e
- Surkis, T. (18 de octubre de 2018) Architecture Components & MVP \ MVVM. Recuperado de https://android.jlelse.eu/architecture-components-mvp-mvvm-237eaa831096
- Tsakiridis, T. (30 de abril de 2018) RESTful API consuming on Android using Retrofit and Architecture Components (LiveData, Room and ViewModel). Recuperado de https://medium.com/@thanasakis/restful-api-consuming-on-android-using-retrofit-and-architecture-components-livedata-room-and-59e3b064f94
- Pattanaik, R. (5 de enero de 2018) Learning Android Development in 2018— Part 1. Recuperado de https://android.jlelse.eu/learning-android-development-in-2018-part-1-83a514f6a205
- Tomás Gironés, J. (2018) El gran libro de Android avanzado (4ª Ed). Barcelona: Marcombo
- Duttom, S. (4 de noviembre de 2013) WebRTC in the real world: STUN, TURN and signaling. Recuperado de https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/
- Muntenescu, S. (4 de noviembre de 2016) Android Architecture Patterns Part 3: Model-View-ViewModel. Recuperado de https://medium.com/upday-devs/android-architecture-patterns-part-3-model-view-viewmodel-e7eeee76b73b
- Navigation Codelab. Recuperado de https://codelabs.developers.google.com/codelabs/android-navigation/#0

- Saleh, H. (31 de mayo de 2017) MVVM architecture, ViewModel and LiveData (Part 1). Recuperado de https://proandroiddev.com/mvvm-architecture-viewmodel-and-livedata-part-1-604f50cda1
- Castronounis, A. What Is WebRTC and How Does It Work?. Recuperado de https://www.innoarchitech.com/blog/what-is-webrtc-and-how-does-it-work
- Garibay, V. (6 de julio de 2016) Consumiendo una API con Retrofit 2 en Android. Recuperado de https://stories.devacademy.la/mi-primer-app-con-retrofit-y-android-ac61a8954a2c
- Peitek, M. (19 de enero de 2017) Retrofit 2 — Creating a Sustainable Android Client. Recuperado de https://futurestud.io/tutorials/retrofit-2-creating-a-sustainable-android-client
- Shrestha, V. (9 de noviembre de 2017) How to create Nested TabLayout with ViewPager in Android. Recuperado de https://www.thecodecity.com/2016/05/create-nested-tablayout-with-viewpager-android.html

Anexo I: Manual de Usuario

Administradores y Equipos:

- **Inicio de sesión**

Al abrir la aplicación deberá escoger su rol, Admin o Equipo. En la siguiente pantalla deberá introducir sus credenciales para entrar al sistema.

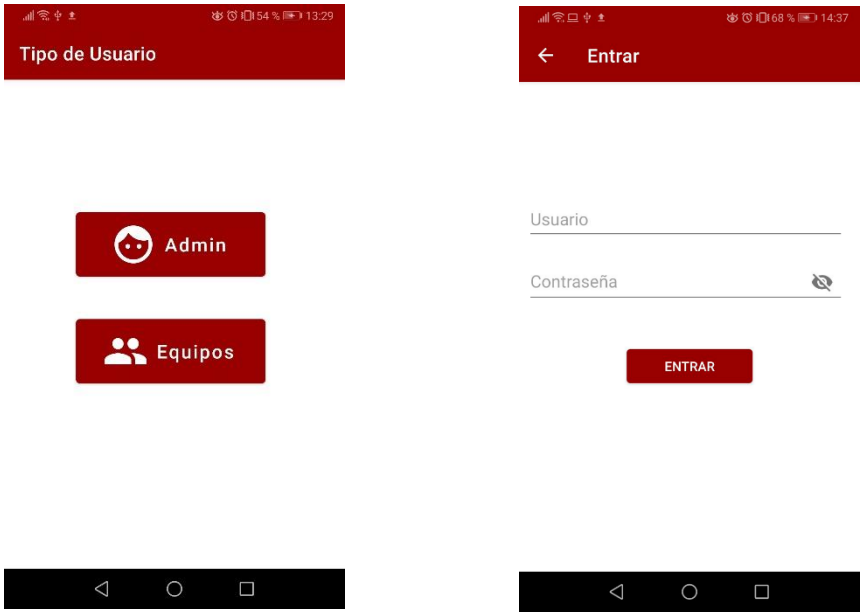


Figura A.1.1 Pantallas para el inicio de sesión

- **Envío de mensajes**

  Una vez situado en la pantalla de una incidencia activa o de la incidencia actual en caso de ser un equipo, seleccionar la pestaña CHAT y debajo el chat, al que se desea enviar el mensaje, en caso de existir, algún usuario conectado, aparecerá una sala privado para enviárselo a él.

  Para enviar un mensaje de texto escribir en el campo destinado a ello y pulsar sobre el botón con la flecha para enviar.

  Para enviar un mensaje predefinido, pulsar sobre el icono con el bocadillo y seleccionar de la ventana emergente el mensaje a enviar.

  Si se desea enviar un mensaje de audio pulsar sobre el icono del micrófono y comenzar a hablar, cuando este el mensaje completo pulsar sobre enviar en dicha ventana.

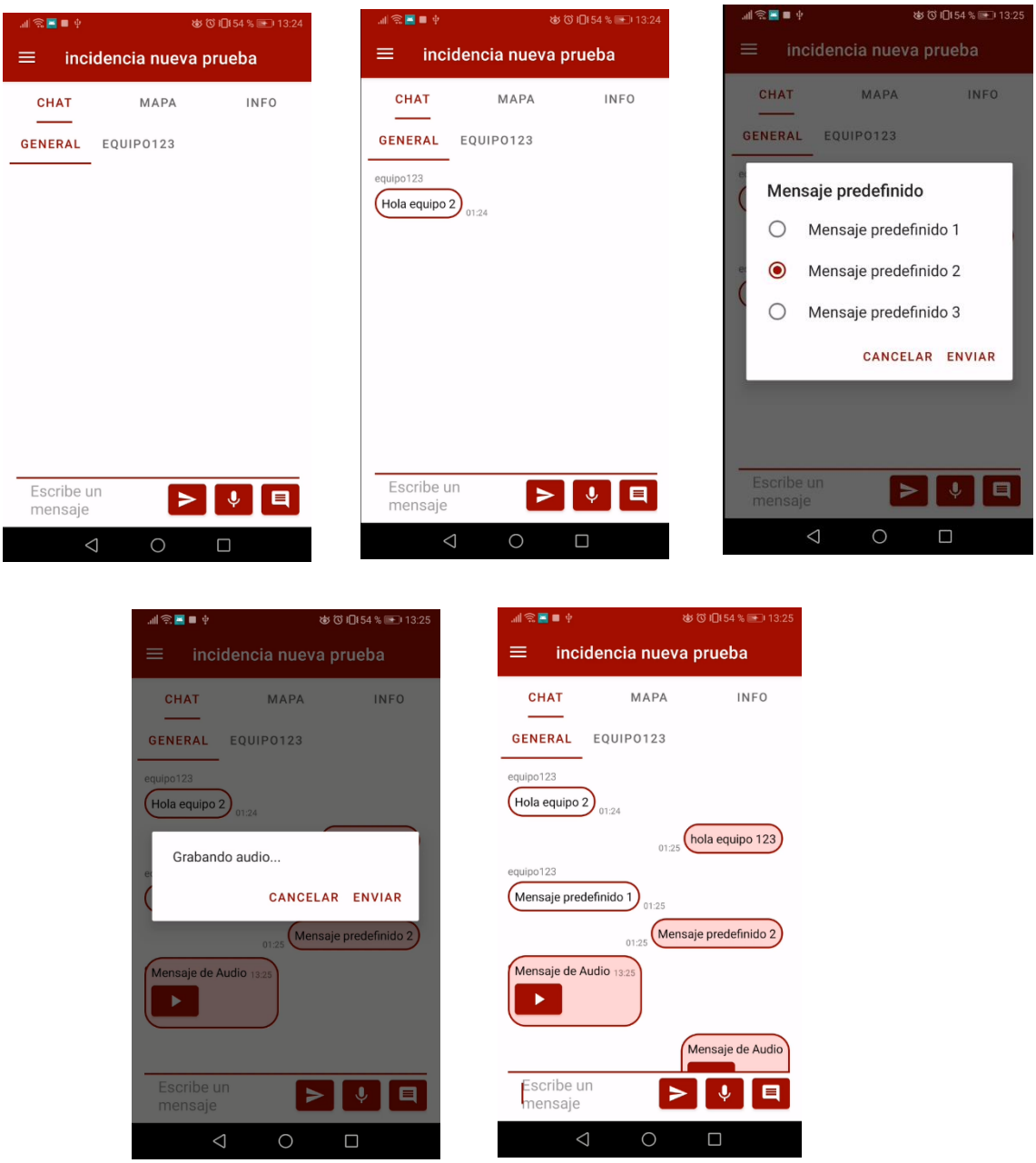


Figura A.1.2. Pantallas para el envío de mensajes de texto y de audio

- **Visualización de posiciones de equipos**

  Dentro de la pantalla de visualización de incidencia pulsar sobre la pestaña MAPA, ahí aparecen marcadores tanto para la incidencia como para los equipos conectados con su ubicación actual.

Figura A.1.3. Pantalla para observar la posición de los equipos en la incidencia

- **Visualización de información de incidencia activa**

  Dentro de la pantalla de visualización de incidencia pulsar sobre la pestaña INFO, ahí aparecen los datos de la incidencia.

Figura A.1.4. Pantalla para observar la información de la incidencia

Administradores:

Las pantallas que se indican aquí son accesibles a través del menú de la aplicación que se observa en la siguiente imagen.

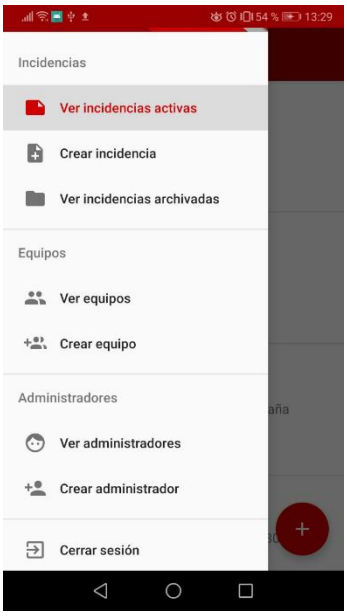


Figura A.1.5. Menú para acceder a las distintas funciones de administrador

- **Creación de un nuevo administrador**

  Elegir la opción crear administrador, en el formulario que aparece rellenar los datos deseados y pulsar sobre el botón Crear.

Figura A.1.6. Pantalla para crear un nuevo administrador

- **Listado de administradores**

  Elegir la opción Ver Administradores en el menú, aparecerá en pantalla la lista de administradores del sistema.

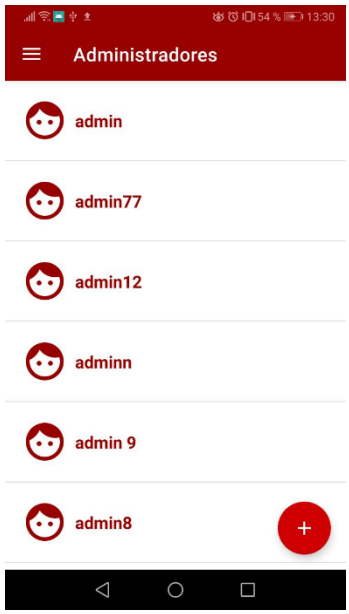


Figura A.1.7. Pantalla para visualizar los administradores dados de alta

- **Edición / borrado de un administrador**

  Pulsar sobre un administrador del listado de administradores. En la pantalla que aparece se podrán editar los datos o bien borrar el administrador pulsando sobre el botón flotante.

Figura A.1.8. Pantalla para editar o borrar un administrador

- **Listado de equipos**

  Elegir la opción Ver Equipos en el menú, aparecerá en pantalla la lista de administradores del sistema.

Figura A.1.9. Pantalla para visualizar los equipos dados de alta

- **Creación de un nuevo equipo**

  Elegir la opción Crear Equipo del menú, en el formulario que aparece rellenar los datos deseados y pulsar sobre el botón Crear.

Figura A.1.10. Pantalla para crear un equipo

- **Edición / borrado de un equipo**

  Pulsar sobre un equipo del listado de equipos. En la pantalla que aparece se podrán editar los datos o bien borrar el equipo pulsando sobre el botón flotante.

Figura A.1.11. Pantalla para editar / borrar un equipo

- **Listado de incidencias activas**

  Elegir la opción Ver incidencias activas en el menú, aparecerá en pantalla la lista de incidencias activas del sistema.

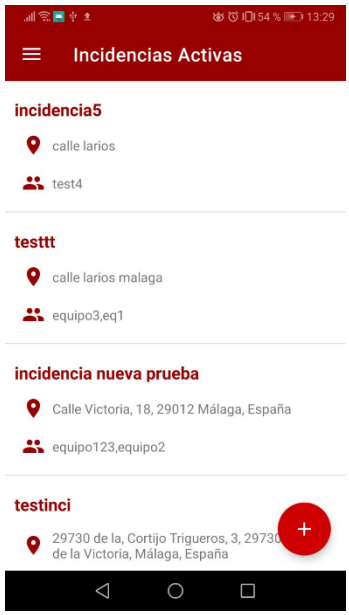


Figura A.1.12. Pantalla para ver el listado de incidencias activas

- **Listado de incidencias archivadas**

  Elegir la opción Ver incidencias archivadas en el menú, aparecerá en pantalla la lista de incidencias archivadas del sistema.

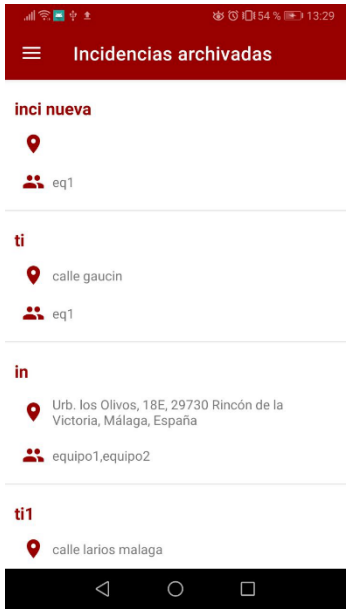


Figura A.1.13. Pantalla para ver el listado de incidencias archivadas

- **Visualización de incidencia archivada**

  Pulsar sobre la incidencia deseada en el listado de incidencias archivadas, en la pantalla se mostrará los datos de esa incidencia.

Figura A.1.14. Pantalla para ver los datos de una incidencia archivada

- **Creación de nueva incidencia**

  Seleccionar la opción Crear Incidencia del menú, en la pantalla siguiente introducir los datos de la incidencia. Para la ubicación es posible bien escribir la dirección y pulsar sobre el botón “Obtener Coordenadas” o realizar una pulsación larga sobre el mapa. Del listado de equipos disponibles marcar aquellos que se desean asignar a la incidencia.

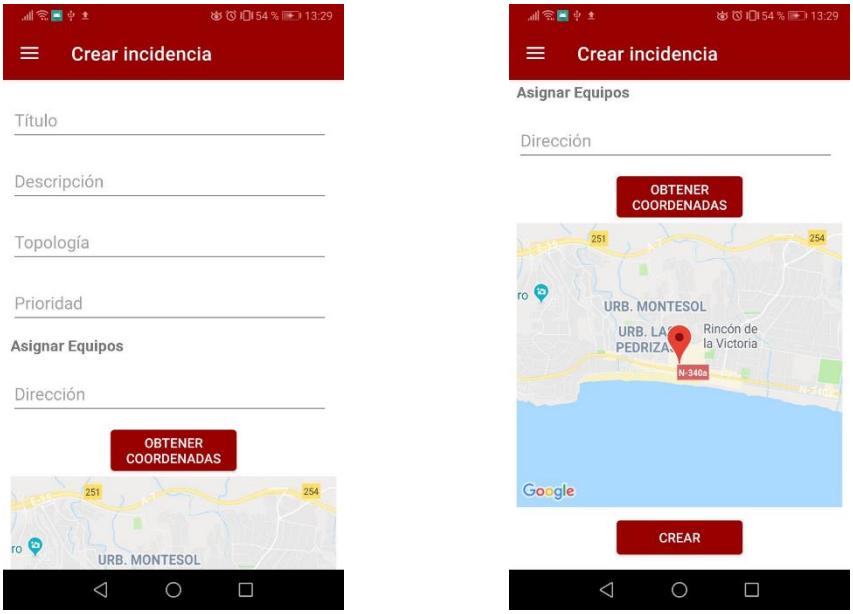


Figura A.1.15. Pantallas para la creación de una nueva incidencia

- **Edición y archivado de incidencia**

  Desde la visualización de una incidencia activa, seleccionar la opción deseada desde el menú que aparece en la barra. En el caso de elegir Editar incidencia aparece un formulario igual al de la acción de crear incidencia sobre el que realizar los cambios.

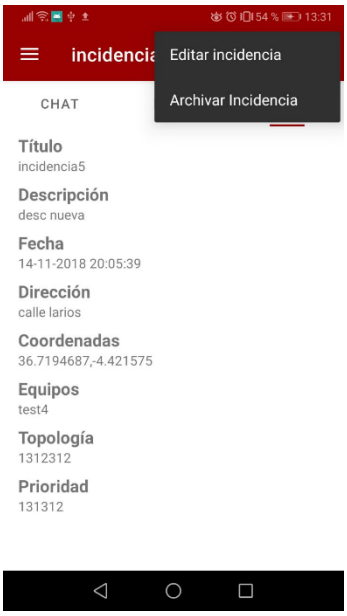


Figura A.1.16. Menú para acceder a la edición o archivar una incidencia actual