Sistema de seguimiento de espacios colectivos
Parte 1

Autor:
Oriol Fort

Director:
Xavi Burguès

Fecha defensa:
30/10/2020

Grado Ingeniería Informática
Especialidad en Ingeniería del Software

Índice

Resumen

En África, las ciudades no tienen las facilidades que se encuentran en la mayoría de ciudades de Europa. En Maputo, la capital de Mozambique, el ayuntamiento no tiene la infraestructura necesaria para poder monitorizar el estado del entorno público en toda la ciudad. Hay barrios que pueden pasar años hasta que el ayuntamiento llegue a arreglar una farola que está averiada. El ayuntamiento no tiene forma de saber dónde se le necesita, y nuestro proyecto pretende ayudar a las instituciones de Maputo a gestionar las incidencias en el entorno público para poder agilizar la recepción de incidencias y su correcta gestión.

Este trabajo desarrollará un software que permita a los ciudadanos de Maputo reportar incidencias para que el ayuntamiento tenga conocimiento de donde es necesario actuar. Nuestro sistema también pretende ayudar a la gestión de las incidencias, con perfiles de administrador para la gestión de las incidencias reportadas. Las diferentes instituciones que puedan hacerse cargo de solucionar las incidencias también tendrán acceso al sistema, pudiendo consultar y actualizar las incidencias.

Ahora mismo, el ayuntamiento de Maputo tiene una linea de teléfono para recibir los reportes de los ciudadanos. El software diseñado pretende proporcionar una plataforma para que el ayuntamiento de Maputo pueda digitalizar las incidencias reportadas por los ciudadanos, y agilizar su gestión.

Resum

A l'Àfrica, les ciutats no tenen les facilitats que es troben en la majoria de ciutats d'Europa. En Maputo, la capital de Moçambic, l'ajuntament no té la infraestructura necessària per poder monitoritzar l'estat de l'entorn públic a tota la ciutat. Hi ha barris que poden passar anys fins que l'ajuntament arribi a arreglar un fanal que estigui avariat. L'ajuntament no té forma de saber on és necessària la seva intervenció, i el nostre projecte pretén ajudar les institucions de Maputo a gestionar les incidències en l'entorn públic per poder agilitzar la recepció d'incidències i la seva correcta gestió.

Aquest treball desenvoluparà un programa que permeti als ciutadans de Maputo reportar incidències perquè l'ajuntament tingui coneixement d'on cal actuar. El nostre sistema també pretén ajudar a la gestió de les incidències, amb perfils d'administrador per a tractar les d'incidències reportades. Les diferents institucions que puguin fer-se càrrec de solucionar les incidències també tindran accés a sistema, podent consultar i actualitzar les incidències.

Ara mateix, l'ajuntament de Maputo té una línia de telèfon per rebre els informes dels ciutadans. El software dissenyat pretén proporcionar una plataforma perquè l'ajuntament de Maputo pugui digitalitzar les incidències reportades pels ciutadans, i agilitzar la seva gestió.

Abstract

In Africa, cities do not have the facilities found in most cities in Europe. In Maputo, the capital of Mozambique, the city council does not have the necessary infrastructure to be able to monitor the state of the public environment throughout the city. There are neighborhoods that can take years for the city council to fix a faulty lamppost. The institutions has no way of knowing where it is needed. Our project aims to help Maputo institutions to manage incidents in the public environment in order to speed up the reception of incidents and their correct management.

This work will develop a software that allows the citizens of Maputo to report incidents so that the city council has knowledge of where it is necessary to act. Our system also aims to help the management of incidences, with administrator profiles to manage the reported incidences. The different institutions that can take charge of solving the incidences will also have access to the system, being able to consult and update the reports.

Right now, the Maputo city council has a telephone line to receive reports from citizens. The designed software aims to provide a platform for the Maputo city council to digitize incidences reported by citizens, and streamline their management.

1- Introducción

1.1- Contexto

Este trabajo de final de grado trata de desarrollar un sistema que permita un seguimiento de los espacios colectivos/públicos. Este software será desarrollado para el ayuntamiento de Maputo, Mozambique. El proyecto está subvencionado por el Centro de Cooperación para el Desarollo de la UPC (CCD). El sistema se utilizará para facilitar y agilizar los procesos de mejora y mantenimiento de los barrios. Nosotros nos centraremos en el distrito George Dimitrov, una zona pobre con calles sin asfaltar y con problemas de monitorización desde el ayuntamiento del estado de los espacios públicos.

Debido a la condición humilde de algunas zonas de Maputo, los residentes no tienen ordenadores, ni smartphones, ni internet en ocasiones, pero sí que tienen un teléfono móvil básico antiguo.

Así pues, nuestro proyecto utilizará USSD, a parte de la interficie web, así los ciudadanos podrán reportar las incidencias que haga falta a través del móvil, con la tecnología USSD, que se basa en un sistema de menu/submenús. Tendremos un servidor que almacene estos mensajes en una base de datos. Los funcionarios del ayuntamiento tendrán un portal a través del cual podrán visualizar las incidencias enviadas por los ciudadanos, actuando en respuesta.

Actualmente el ayuntamiento tiene un servicio telefónico, *“Linha verde”*, en donde los ciudadanos pueden llamar a un teléfono público para realizar quejas o sugerencias. Los trabajadores se hallan desbordados y las reclamaciones no son digitalizadas. Nosotros proponemos utilizar sistemas informáticos para llevar a cabo estas tareas, de forma que se agilizaría y facilitaría el trabajo de muchas personas, lo que se vería reflejado en un mejor mantenimiento de los espacios públicos de Maputo. Nuestro sistema servirá tanto para ayudar a linha verde como para que los usuarios puedan reportar incidencias directamente.

Contenedores llenos o quemados, inundaciones, farolas sin luz, socavones …. son algunos de los ejemplos de incidencias que los ciudadanos podrán reportar a través del sistema.

Este proyecto lo realizaré conjuntamente con Eynar Arnez. Debido a la gran carga de trabajo que supone la creación de un sistema completo (aunque se trate de un prototipo) nos será posible dividir el proyecto en dos partes y seguir teniendo suficiente trabajo como para desarrollar dos proyectos de final de grado.

El cliente, supuestamente el ayuntamiento de Maputo, por culpa de la pandemia no ha estado atento a nuestro proyecto, y es posible que al final el proyecto acabe en manos de otra entidad, como la eléctrica de Maputo, que si mostró interés.

1.2- Actores Implicados

A continuación describimos un listado con los diferentes actores.

Ayuntamiento de Maputo

El Ayuntamiento de Maputo es la institución que recibirá el software, se encargará de gestionarlo y de resolver las incidencias resultantes.

Operarios Linha Verde

Utilizarán la plataforma para introducir en el sistema las incidencias recibidas por teléfono usando un perfil de administrador.

Residentes de Maputo

Son los principales beneficiados. Al poder reportar las incidencias del espacio público, ayudarán al ayuntamiento a saber los problemas existentes y poder solucionarlos.

CCD

Es la institución responsable de financiar el proyecto.

Director y ponente

Xavi Burgués será el director. Ha viajado a Maputo y es quien nos ha dado los primeros datos e instrucciones sobre lo que hay que hacer. Él nos hace de intermediario con Maputo.

Equipo de desarrollo

Eynar Arnez y yo, con la ayuda y consejo de nuestro director, desarrollaremos el prototipo de este sistema.

1.3-Justificación

El desarrollo de este sistema será financiado por el CCD y el Ayuntamiento de Maputo, que muestra un especial interés y necesidad.

Los barrios periféricos de Maputo están aislados de las instituciones y administraciones, que tienen dificultades para llegar y mantener una monitorización del estado de los espacios públicos efectiva. Para mejorar las condiciones de estos barrios es necesaria una forma de acceder a las administraciones a fin de informar de las necesidades de trabajos de mejora y mantenimiento. De esta manera el ayuntamiento podrá actuar en consecuencia.

Actualmente la única vía de información que tiene el ayuntamiento para informarse de las necesidades de mantenimiento en estos barrios es a través del servicio telefónico. Los trabajadores se quejan de la cantidad de llamadas que reciben y la dificultad de organizar y analizar toda la información que les llega. Digitalizar estos datos ayudará al ayuntamiento para llevar a cabo más rápida y eficazmente este trabajo de gestión de incidencias. Todas las incidencias reportadas por teléfono (linha verde) podrán ser introducidas al sistema por los administradores, lo que ayudará también a la estandarización de las incidencias, que resultará en una gestión más fácil. Como ya se ha mencionado antes, no solo los administradores darán de alta incidencias, los ciudadanos podrán también reportar incidencias tanto por USSD como vía web.

1.3.1- Estado del arte

Listado de algunas aplicaciones o webApps.

- Cityzn: *“Esta app permite que los habitantes de las ciudades aporten ideas para fomentar elbienestar común y facilitar el consenso entre vecinos y comunidades”*

- YoVeoVeo (Ecuador): *“Con el objetivo de reducir el exceso de burocracia e incentivar al ciudadano a denunciar incidencias a la administración nació esta app en Ecuador, que sirve de canal directo de comunicación entre el gobierno y los ciudadanos comprometidos con la gestión pública.”*

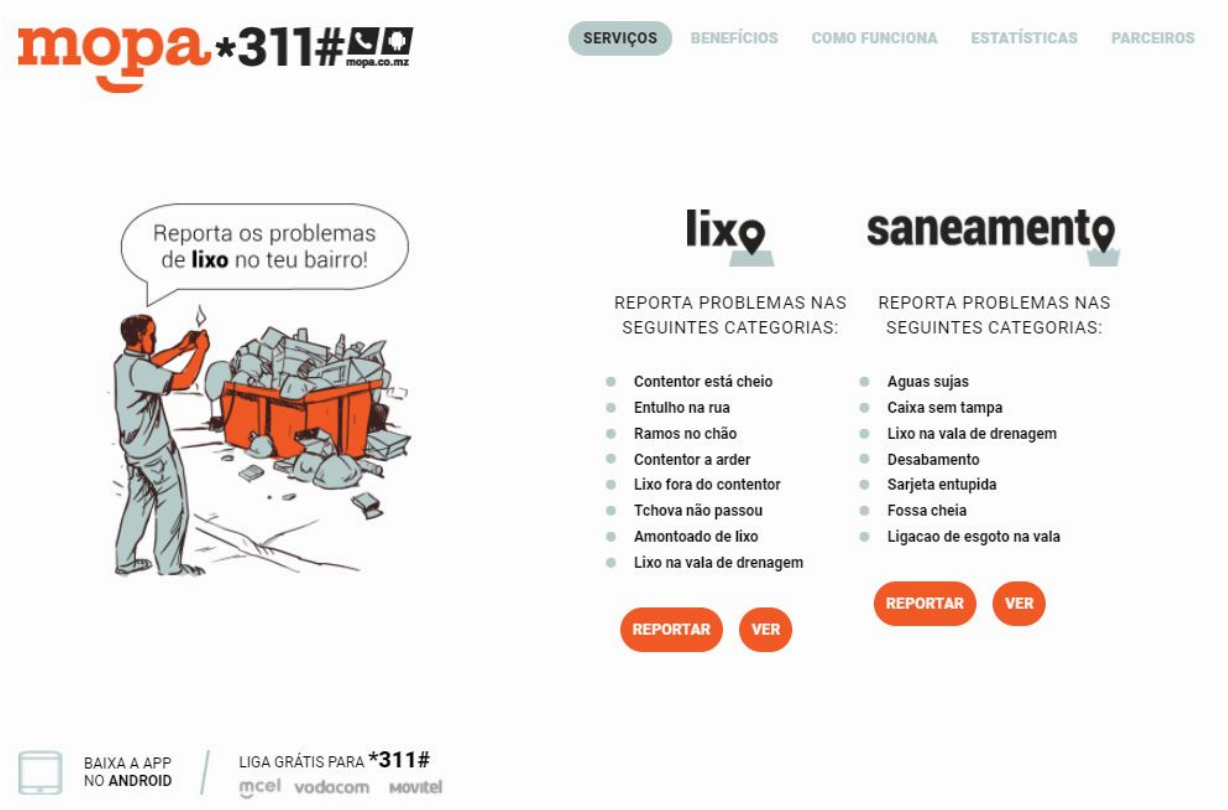
- Arrels Localizador (España): *“Desde la Fundación Arrels han desarrollado esta aplicación cuyo objetivo es facilitar el trabajo a los equipos de asistencia a personas sin hogar, gracias a la participación ciudadana. Desde la app, cualquier usuario puede informar de la localización de personas que pasan la noche a la intemperie para que los equipos de la Fundación puedan asistirlas.”*

Como se puede observar, la tendencia que se está siguiendo es que el ciudadano tenga participación en su ciudad.

Este tipo de sistema puede ser desarrollado con fondos públicos, ya que está enfocado a la gestión de los espacios públicos. El caso más destacable relacionado con nuestro proyecto sería el **MOPA**, y será el único ejemplo a mencionar y estudiar.

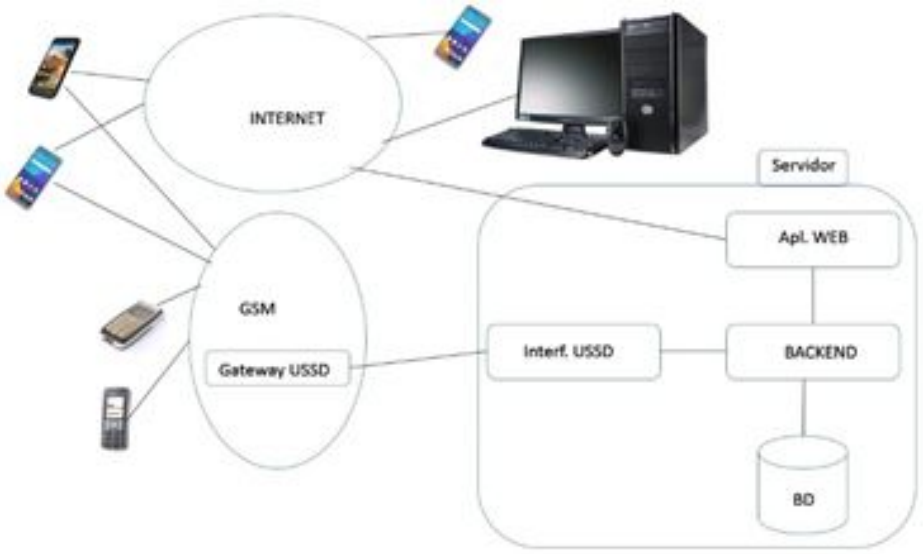
El MOPA fue desarrollado con fondos internacionales por la ONU, destinado a Mozambique. El plan era hacer una aplicación como la que estamos intentando desarrollar con nuestro trabajo. La aplicación parecía bastante funcional, pero para el buen uso de ésta desde el ayuntamiento se necesitaba un mantenimiento del aplicativo. El problema principal es que los fondos internacionales sirvieron para que se desarrollara MOPA, pero una vez finalizado el presupuesto, el equipo de desarrollo desapareció y el software quedó desamparado, de manera que cualquier actualización necesaria para su uso fue totalmente desatendida. El problema es que se desarrolló un software bastante funcional, pero que al final no les pertenecía al Ayuntamiento de Maputo. Es por eso que ahora el Ayuntamiento de Maputo está interesado en un nuevo software que sea suyo, de manera que ellos puedan gestionar y actualizar de acuerdo a las necesidades.

En cualquier caso, este software será una gran inspiración para el diseño y desarrollo de nuestro portal.



1.4- Alcance

El desarrollo de este sistema lo haremos de acuerdo a este primer esbozo de arquitectura a seguir:



El objetivo final del proyecto es proporcionar al Ayuntamiento de Maputo una herramienta para poder gestionar mejor las incidencias en el entorno público.

1.4.1- Objetivos

Para acotar mejor el alcance del proyecto, definiremos unos objetivos que nos ayudaran a detallar mejor todo lo que nos permitirá hacer la aplicación:

- Reportar incidencias en el entorno público como ciudadano desde una web o USSD (móvil). Estas incidencias serán almacenadas en el sistema.

- Consultar las incidencias reportadas en el sistema desde una web.

- Gestionar las incidencias como ayuntamiento, pudiendo indicar en la web el estado de las mismas, e información útil para la correcta resolución del problema.

- En este sistema se guardarán las principales instituciones responsables de resolver las incidencias (como los bomberos, por ejemplo). El sistema también permitirá gestionar estas instituciones.

- El sistema también será usado por empleados de estas instituciones, de manera que ellos mismos podrán reportar cambios en las incidencias que han estado solucionando, e consultar incidencias pendientes de resolver.

Una vez desarrollado el sistema se planea viajar a Maputo y hacer pruebas de usuario real para poder mejorar el prototipo en un futuro con el feedback del cliente.
Los posibles obstáculos identificados en el proyecto son:

1.4.2- Obstáculos

Algunos obstáculos previstos podrían ser:

- Desconocimiento de USSD. Nunca hemos oído hablar de esta tecnología. Parece ser una buena solución para los barrios mas marginales de Maputo, permitiendo a esta parte de la población mas fácil acceso al sistema. Creemos que la implementación de esta tecnología puede llevar problemas, debido a el uso de redes gestionadas a nivel de operadoras telefónicas.

- Falta de comunicación con Maputo. El ayuntamiento es el principal interesado en este prototipo, así que las necesidades de los usuarios pueden no coincidir con la implementación real. Si no llegamos a entender las necesidades del ayuntamiento, el prototipo podría carecer de muchas funcionalidades.

- Encontrar un servidor para el desarrollo. Necesitaremos un servidor para poder probar la aplicación. Sabemos que en Maputo tendrán un servidor que podremos usar, nos hará falta encontrar uno de características similares donde poder desarrollar nuestro sistema..

- Tener una fecha fija de entrega es un riesgo que puede provocar que el equipo deba decidir renunciar a funcionalidades. Al ser un prototipo nos interesa implementar las funcionalidades principales, que den una idea del uso que se le puede dar al sistema. Tendremos que priorizar tareas, de tal forma que si al final del desarrollo no hemos tenido tiempo de acabar todas las funcionalidades, nos aseguremos que lo que falte para implementar sea lo menos trascendental para nuestra aplicación.

2- Metodología

Los primeros pasos para el desarrollo del proyecto se han hecho conjuntamente, tomando las decisiones en reuniones, y repartiéndose el trabajo del diseño y requisitos a medias

Gracias a las especificaciones en los casos de uso, hemos tenido una guía de qué funcionalidades desarrollar. La comunicación con Eynar ha sido crucial, y hemos estado haciendo una reunión semanal los dos con el director del proyecto. De esta forma hemos estado informados en todo momento de qué tareas estaba desarrollando cada uno de nosotros, y hemos podido responder rápido a los problemas y necesidades que se iban dando durante el proceso.

Tal y como se explicará en el apartado de planificación, Eynar será el responsable de la parte frontend de la web, y yo del backend del sistema. El backend será usado por la aplicación USSD también. El desarrollo de la aplicación USSD también estará asignada a mi.

Generalmente la dinámica ha sido: yo informaba de nuevas funcionalidades del backend para que Eynar pudiera integrarlas en el frontend, y Eynar me iba pidiendo cambios y mejoras en el backend para poder alinearse con el frontend.

He usado git con Bitbucket para ir subiendo el código de las últimas versiones del backend que iba desplegando en el servidor. También he configurado el entorno local como el entorno del servidor. Convirtiéndose el entorno local, como entorno de desarrollo, y el entorno servidor como el entorno de producción. Antes de subir nada a producción, testeaba los cambios respecto a la última versión con JUnit y comprobando desde Postman que los endpoints actuaban de forma deseada.

3- Planificación

Este proyecto está siendo desarrollado por dos alumnos. Así que hemos tenido que dividir parte de las tareas y otras las haremos conjuntamente. La partición principal del desarrollo se puede resumir en frontend/backend. Yo me ocuparé del backend mientras que Eynar desarrollará el frontend.

El proceso de desarrollo de este proyecto lo hemos ideado en dos iteraciones principales, siguiendo aproximadamente una planificación en cascada, que nos lleve hasta la implementación funcional del sistema prototipo. La parte de implementación será incremental.

3.1- Definición y estimación de tareas

Primera iteración

Tareas **conjuntas** primera iteración: (120h conjuntas / 60h por persona)

- Decidir tecnología open source para instalar en un servidor. (10h)
- Instalación/configuración servidor + Documentación, paso a paso de la instalación (40h)
- Análisis requisitos (25h)
- Historias de usuario (30h)
- Contratos API Rest Frontend-backend (15h)

El objetivo de esta iteración es poner en marcha el servidor y preparar el entorno para su desarrollo en la siguiente iteración.

Una tarea a tener en cuenta será la documentación paso a paso de la instalación del servidor. Para hacer funcionar nuestro sistema, los operadores del Ayuntamiento de Maputo deben poder instalar el entorno. Al no saber cuál es el nivel de conocimiento informático de los trabajadores públicos, es imprescindible una documentación a modo de tutorial para asegurarse que puedan configurar el servidor de la forma correcta para el buen funcionamiento del sistema.

Se acabaría la iteración con el análisis de los requisitos y la definición de las funcionalidades indispensables para poder empezar el desarrollo con una idea suficientemente clara.

Tareas **individuales** primera iteración: (320h)

- Diseño y desarrollo backend + pruebas unitarias

En paralelo a las tareas conjuntas, se deberá diseñar el backend web y la base de datos del sistema, así como las pruebas unitarias para el correcto control de calidad.

Segunda iteración

Tareas **conjuntas** segunda iteración (60h conjuntas - 30h por persona)

- Pruebas integración del sistema (20h)
- Preparación pruebas usabilidad (40h)

Una vez desarrollado el frontend y backend en paralelo, podremos juntar las dos partes y hacer las pruebas de integración finales para asegurarnos que el conjunto del sistema funciona correctamente. Durante el desarrollo ya se habrán probado la mayoría de funcionalidades, y la comunicación front-back habrá sido importante. Así que las pruebas de integración no deberían suponer un gran contratiempo.

Con el software ya desarrollado, podremos diseñar pruebas de usabilidad. Deberemos planear varias sesiones con usuarios que prueben el sistema para recibir feedback. Es importante comprobar que los usuarios para quienes está diseñado nuestro sistema podrán utilizarlo fácilmente, y apuntar todas las mejoras necesarias para ser corregidas. También deberemos ver qué echan de menos los usuarios.

Tareas **individuales** segunda iteración: (50h)

- Memoria
- Presentación/ defensa TFG

Individualmente, cada uno de los dos alumnos que hemos participado en este proyecto redactará la memoria del trabajo efectuado y preparará su presentación. Esta memoria servirá de referencia para futuros desarrollos de mejora del producto entregado al final de este trabajo.

Viaje a Maputo, Mozambique

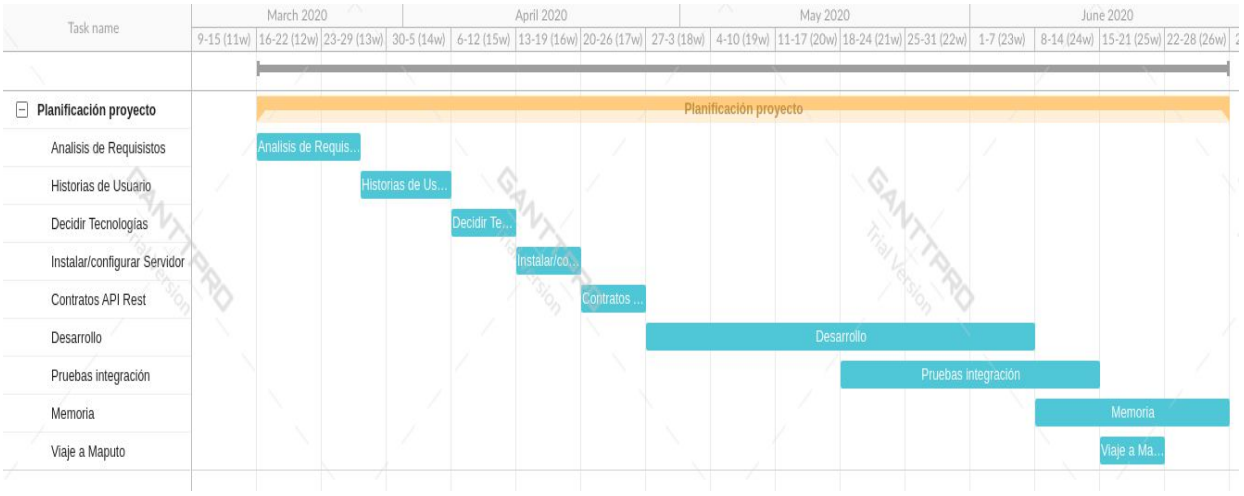
Tareas en Maputo: (20h individuales)

- Ejecución pruebas de usabilidad con ciudadanos
- Presentación técnicos SIGEM (Sistema Integrado de Gestión Municipal)

Se plantea ir a Maputo con el prototipo del sistema desarrollado y las pruebas de usabilidad preparadas para ser efectuadas. El objetivo del viaje es probar el sistema con los usuarios potenciales del mismo, para recibir feedback que nos pueda ayudar a mejorar. A parte de las sesiones de prueba de usabilidad, también se presentará el software a los técnicos locales que se ocuparán de su funcionamiento y mantenimiento.

El total de horas teniendo en cuenta las tareas descritas anteriormente, es de **480h** por estudiente.

3.2- Diagrama de Gantt



4- Presupuesto

Este proyecto está subvencionado por el CCD.

Identificación y estimación de los costes

Costes recursos humanos

- Coste: 12.790€

Somos dos desarrolladores, nos repartirem el trabajo equitativamente, cumpliendo con los siguientes roles. Según la planificación son 480h por persona.

Rol	Precio por hora (€)
Jefe de proyecto	17
Analista	12
Desarrollador	12
Responsable de pruebas	11,5

Se tendrá en cuenta que la definicíon incial del contexto, alcance, requisistos, etc, estará presupuestada para jefe de proyecto. Las 320h de desarrollo se dividirán en 100h por persona de Analista, donde se harán las tareas más de diseño y arquitectura del sistema y 220h de programación por persona con el rol de Desarrollador. La redacción de la memoria y la presentación también se van a presupuestar con el rol de Jefe de proyecto.

Rol	Horas	Precio total
Jefe de proyecto	260h	4420€
Analista	200h	2400€
Desarrollador	440h	5280€
Responsable de pruebas	60h	690€

El presupuesto total en recursos humanos suma **12.790€**.

Costes Hardware

- Coste: 250€

En este caso necesitamos dos ordenadores y un servidor. Calculamos unos 1000€ por ordenador. El servidor nos sale gratis, ya que la UPC nos ha dejado un ordenador reacondicionado para utilizar de servidor.

Necesitaremos dos portátiles para el desarrollo de nuestro sistema, ya que desarrollaremos simultáneamente frontend y backend, Eynar y yo respectivamente. El costo que se calcula, es a partir de la amortización. Donde después de cuatro años el software, quedará obsoleto. teniendo en cuenta que el desarrollo se hará en unos 4 meses, tenemos el siguiente cálculo:

Hardware	Coste(€)	Vida útil (años)	Amortización 4 meses (€)
MacBook Pro	1500	4	125
Total			**250**

Costes Software

- Coste: 0€

En este caso utilizaremos únicamente software libre. No habrá costes de software.

Costes indirectos

- Coste: 1410€

Calculamos **internet** a unos 30€ al mes y **la electricidad** 40€. Necesitaremos unos tres meses para desarrollar el proyecto. También hay que calcular el viaje a Maputo. Viaje + estancia = 1200€.

El proyecto requiere del uso de la luz eléctrica y de internet para el desarrollo. Por tanto, se ha de tener en cuenta.

Recursos	Coste/mes	Coste total (€)
Internet	30	90
Consumo eléctrico	40	120
Total		**210**

A estos 210€ se les suma los 1200€ del viaje a Maputo, así que el presupuesto de los costes indirectos es **1410€**.

Costes imprevistos

Los imprevistos que pueden pasar a lo largo del proyecto pueden venir tanto de los recursos directos como indirectos, por ello se decidió a principio del proyecto tener un margen de maniobra ante posibles problemas que puedan aparecer en el desarrollo del sistema. Para que la viabilidad del proyecto no se vea comprometido, se ha estimado un 15% respecto al presupuesto calculado. Este porcentaje nos dará un margen aceptable para posibles contratiempos.

Imprevisto	Probabilidad (%)	Precio (€)	Coste estimado (€)
Recursos directos	15	12.790 + 250	1.956
Recursos indirectos	15	1.410	211,5
Total			**2.167,5**

Presupuesto Final

- **Presupuesto**: **16.617,5*€***

Recurso	Coste (€)
Recursos humanos	12.790
Recursos hardware	250
Recursos software	0
Costes indirectos	1.410
Costes imprevistos	2.167,5
Total	**16.617,5**

5- Informe de Sostenibilidad

Este proyecto está dedicado al mantenimiento de la ciudad de Maputo. Gracias al sistema, la ciudad solucionará problemas del estado de la misma mucho más rápido. No se prevé ningún impacto negativo. Si la gente utiliza la plataforma, el uso de este sistema hará que Maputo sea una ciudad más sostenible social, ambiental y económicamente.

5.1- Impacto económico

Podríamos decir que se prevee que el uso del sistema mejorará el rendimiento del ayuntamiento, y por tanto ayudará a el óptimo uso de los recursos de la ciudad.

El ayuntamiento (o cualquier institución interesada en Maputo) adquirirá este software en cuando esté funcional, de forma totalmente gratuita. Con nuestro viaje a Maputo enseñaremos a instalar y usar el software a los operarios locales, de esta forma ellos mismos podrán mantenerlo sin necesidad de contratar a una empresa externa que les haga ese servicio.

5.2- Impacto ambiental

Para este proyecto, el único aspecto negativo ambientalmente es la energía usada para hacer correr el programa en los servidores y los mismos dispositivos que quieran acceder a nuestro portal. Realmente no es un impacto significativo.

5.3- Impacto social

Si este sistema acaba siendo utilizado por los ciudadanos de Maputo, la mejora de la ciudad será notable. Los ciudadanos podrán reportar todos los desperfectos que puedan notar en el entorno público, y el ayuntamiento podrá actuar para mejorar sus vidas en las calles. Este aspecto sería el más positivo de nuestra aplicación. Gracias al uso de nuestra plataforma el ayuntamiento tendrá la facilidad de gestión de incidencias en el espacio público suficiente para ejecutar las mejoras de forma mucho más ágil.

6- Especificación

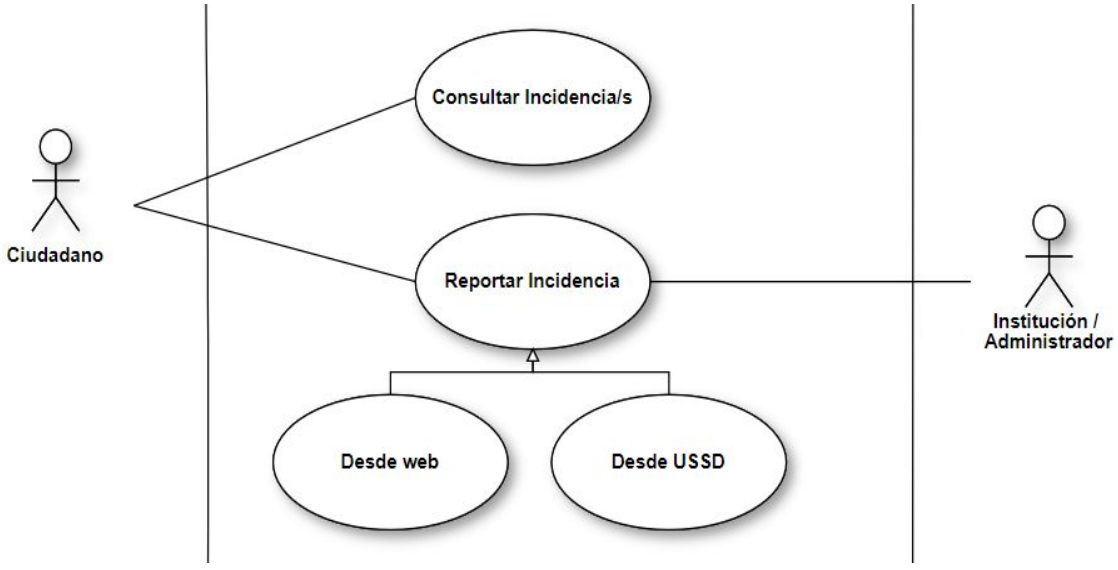
6.1- Diagramas casos de uso

Estos diagramas nos ayudarán a entender las acciones que podrán hacer los usuarios con el sistema. En el apartado de **Requisitos funcionales** se explicarán cada uno de ellos extensamente.

Para los siguientes diagramas se han concretado tres tipo de usuarios del sistema.

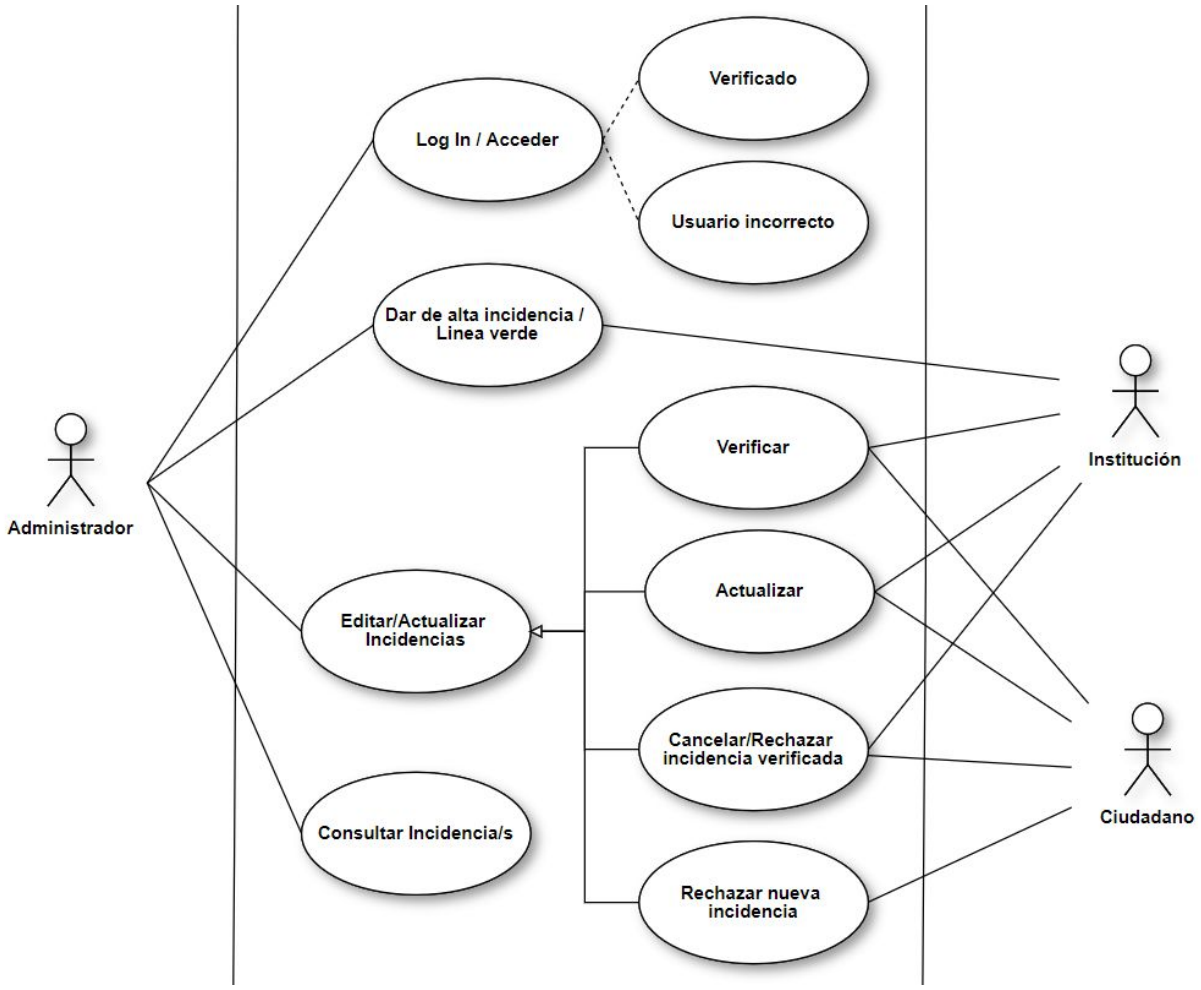
Ciudadanos

Los ciudadanos usarán el sistema sin tener que registrarse. Su interacción con el sistema se basará en reportar incidencias y consultarlas. Las incidencias podrán ser reportadas desde un navegador web, utilizando nuestro portal, o usando su teléfono móvil a través de la tecnología USSD.



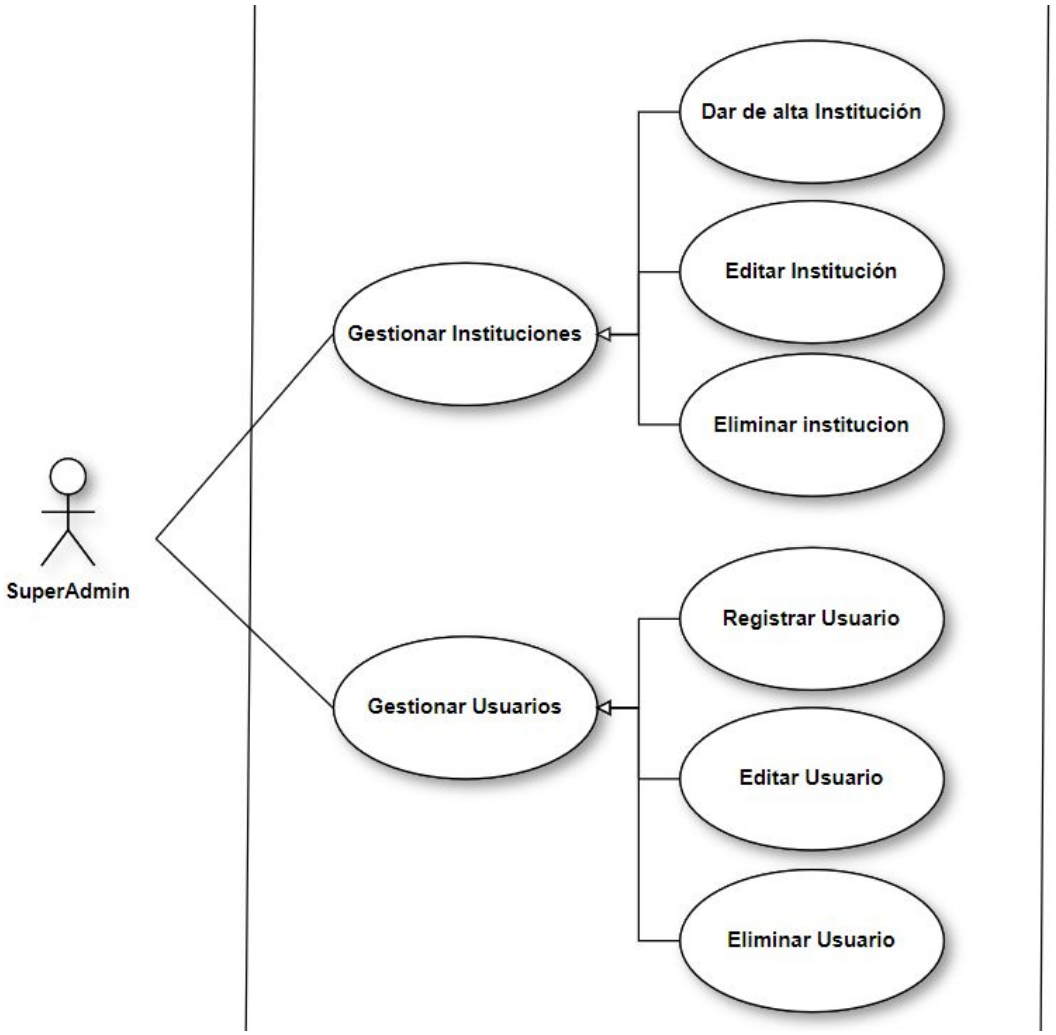
Administradores

Los administradores deberán registrarse con un usuario para acceder al sistema. Ellos serán los encargados de gestionar las incidencias reportadas por los ciudadanos. También podrán introducir incidencias que les puedan llegar por otros canales (linha verde), de manera que queden registradas en el sistema. Tendrán permisos según la institución a la que pertenezcan. De manera que un administrador que pertenezca a la institución de “aguas de Maputo”, por ejemplo, podrá gestionar incidencias del tipo “tubería averiada”, pero si se reporta una incidencia del tipo “contenedor en llamas”, los administradores de la institución “aguas de Maputo” no tendrán permisos para administrar esas incidencias.



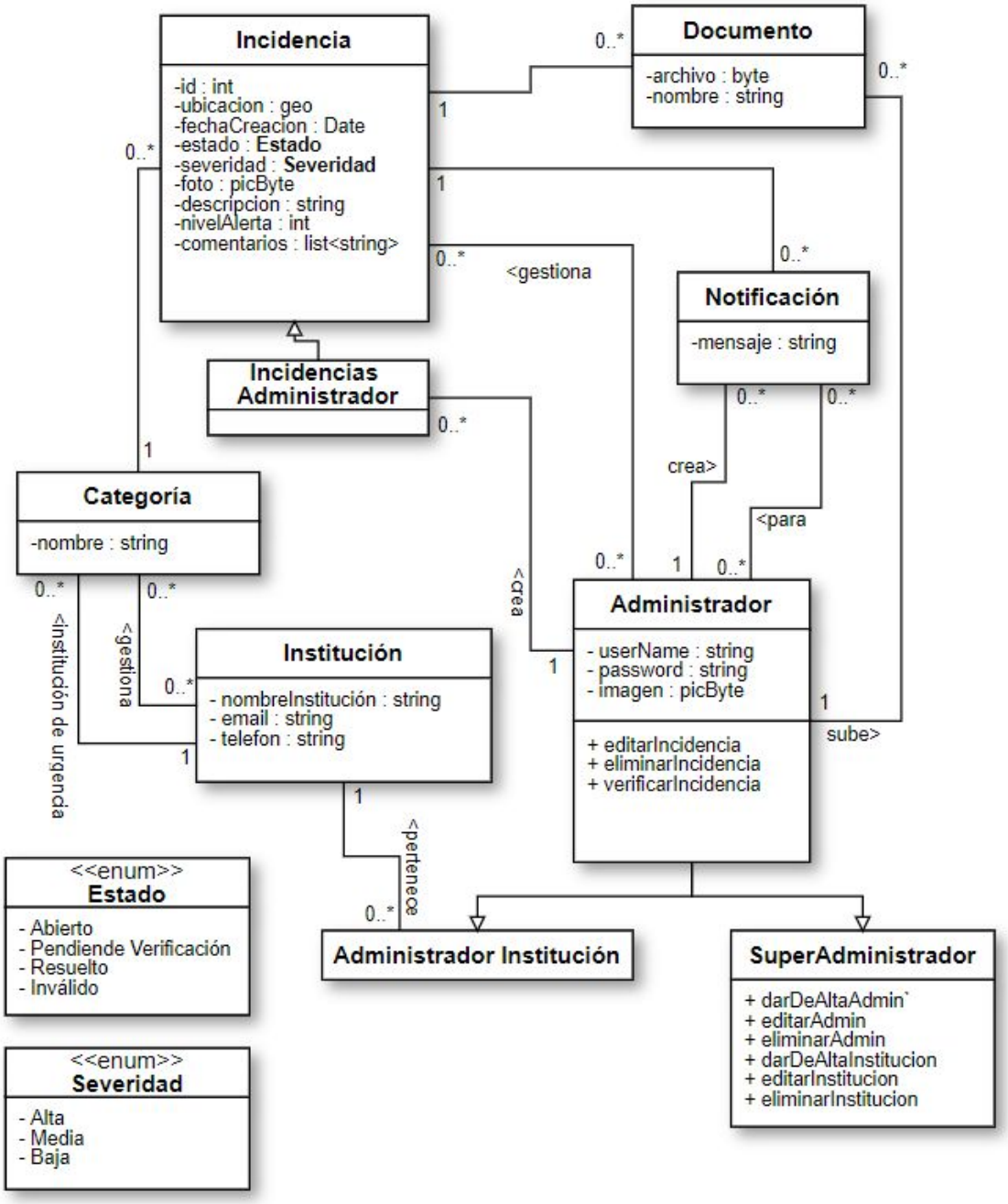
SuperAdministrador

Los superadministradores podrán actuar como administradores, de manera que el diagrama de uso anterior también es aplicable a este tipo de usuario. Podrán usar el sistema con las mismas funcionalidades que los administrados, pero tendrán permisos extra para poder administrar partes clave del sistema, pudiendo gestionar instituciones y usuarios. Estos usuarios tendrán permisos para supervisar todos los tipos de incidencias, sin importar la institución a la que puedan pertenecer.



6.2- Diagrama de clases

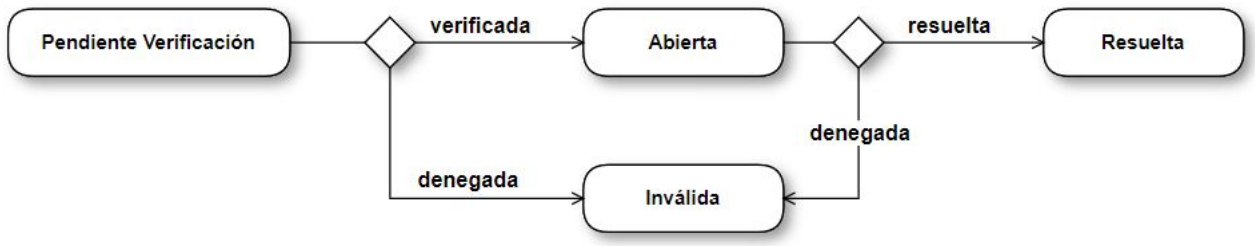
Con las funcionalidades del sistema más claras y definidas, hemos procedido a diseñar el diagrama de clases que defina los principales elementos que podremos encontrar en nuestro sistema y cómo se deben relacionar.



-Un usuario administrador de institución tiene permisos para editar incidencias según las categorías de incidencia gestionadas por su institución
-Al gestionar una incidencia (actualizar, crear, eliminar) se le pregunta al Administrador a que usuarios quiere notificar
-Cuando se crea una incidencia con nivel de alerta Alto, se notifica a la institución de urgencia asignada a la categoria de la incidencia.

6.3- Diagramas de estados

A continuación definiremos el diagrama de estados de las **incidencias**. Con este diagrama nos podremos hacer una idea del flujo normal de trabajo que deberán ejecutar los administradores en el momento de gestionar las incidencias.



Las incidencias al ser reportadas serán creadas en el sistema con estado **Pendiente de Verificación.** Estas incidencias deberán ser supervisadas por un administrador. Este administrador decidirá si la incidencia es procedente, en tal caso verificará la incidencia y ésta pasará a estado **Abierta**. Si esta incidencia no fuera procedente, tanto por una definición insuficiente, porque es antigua o por cualquier otro motivo que el administrador considere, esta incidencia será denegada y pasará a estado **Inválida**. Una incidencia abierta también puede pasar a estado Inválida, si en el transcurso del tiempo se considera que la incidencia no es válida para ser resuelta. Cuando una incidencia abierta es solucionada en la vida real, en el sistema un administrador ha de marcarla como resulta, pasando el estado de la incidencia a estado **Resuelta**.

Cabe comentar que este diagrama de estados es el propuesto por nuestro equipo, pero el sistema permite editar los estado intermedios entre “Pendiente de Verificación” y “Resuelta”. De esta manera la administración que utilice nuestro sistema puede adaptar este flujo de estados a lo más conveniente para su uso.

6.4- Requisitos funcionales

Se ha decidido definir los requisitos funcionales a través de casos de uso. Y se han categorizado para tener una mejor organización:

11.1.1 Usuario ayuntamiento/institución: Acceso a la cuenta

- Iniciar sesión/Cerrar sesión: El sistema tiene que permitir al usuario entrar (introduciendo correo y contraseña válidos en el sistema) y salir del sistema.
- Registro: El sistema tiene que permitir realizar el registro de nuevos perfiles en la aplicación
- Restablecer contraseña: El sistema tiene que permitir al usuario restablecer su contraseña, ya sea por se ha olvidado de su contraseña o por la comodidad del usuario para actualizar su contraseña.
- Editar datos perfil: El sistema tiene que permitir al usuario editar datos del perfil, siendo el correo uno de los parámetros no modificables

11.1.2 Usuario: Gestión incidencias

- Crear una incidencia: El sistema tiene que permitir dar de alta una incidencia
- Buscar una incidencia: El sistema tiene que permitir realizar la búsqueda de incidencias, ya sean las incidencias que muestra el sistema o realizando una búsqueda específica (filtro de estado, categoría, gravedad, rango de fechas o número de incidencia)
- Editar una incidencia: El sistema tiene que permitir a determinados usuarios (perfil ayuntamiento/institución) realizar cambios en una incidencia
- Notificación de una incidencia: El sistema tiene que permitir a determinados usuarios realizar notificaciones a otros usuarios para informar de las modificaciones producidas en una incidencia

11.1.3 Usuario Ayuntamiento: Gestiones super administrador

- Crear una institución: El sistema tiene que permitir al usuario dar de alta nuevas servicios externos, los cuales se se encargaran de resolver incidencias
- Editar/Eliminar institución: El sistema tiene que permitir al usuario gestionar una institución, actualizar datos de perfil (nuevas competencias, nueva dirección de correo) o eliminar institución
- Alta usuario: El sistema tiene que permitir al usuario dar de alta usuarios perfil Ayuntamiento o institución.
- Ajustes: El sistema tiene que permitir al usuario realizar modificaciones en las funcionalidades en el entorno del sistema
    - Crear/Editar/Eliminar un estado
    - Crear/Editar/Eliminar una categoría
    - Crear/Editar/Eliminar una localización

Implementación de los casos de uso del sistema. Tendrán la siguiente estructura:

- Nombre del caso de uso: Titulo del caso de uso.
- Actor principal. usuario o usuarios que realizarán la funcionalidad.
- Precondiciones: el estado en el que debe estar el sistema para que se ejecute el caso de uso.
- Disparador: Acción que activa el caso de uso.
- Escenario principal: Los pasos que se producen en caso de que todo funcione de forma correcta.
- Extensiones: Los pasos alternativos que se producirían en caso de que no vaya bien.

Caso de uso	#1 Iniciar sesión perfil administrador/servicio externo
Actor principal	Perfil administrador
Precondiciones	-
Disparador	El perfil administrador quiere iniciar sesión en el sistema
Escenario principal	1. El administrador introduce correo/identificador y contraseña 2. El sistema valida las credenciales 3. El sistema redirige al usuario accede a la pantalla principal
Extensiones	1. El sistema informa los datos incorrectos 2. El sistema da alternativas para recuperar la contraseña

Caso de uso	#2 Registro al sistema perfil administrador
Actor principal	Perfil administrador
Precondiciones	-
Disparador	El usuario quiere crear una cuenta en el sistema
Escenario principal	1. El usuario rellena el formulario 2. El sistema valida los datos introducidos 3. El sistema guarda los datos introducidos 4. El sistema informa que se ha creado la cuenta 5. El sistema redirige a la página de iniciar sesión
Extensiones	1. El sistema informa de los campos introducidos son incorrectos 2. El sistema valida los datos introducidos por el usuario y el administrador principal no valida la cuenta

Caso de uso	#3 Recuperar contraseña
Actor principal	Perfil administrador
Precondiciones	-
Disparador	El perfil administrador quiere recuperar su contraseña
Escenario principal	1. El usuario quiere recuperar su contraseña 2. El sistema le informa que tiene que introducir correo asociado a la cuenta 3. El sistema valida que la cuenta exista 4. El sistema envía un correo de recuperación de contraseña al correo asociado 5. El sistema genera una url de recuperación de correo 6. El usuario clica sobre la url y accede a la página de cambiar contraseña 7. El usuario cambia contraseña 8. El sistema valida los datos introducidos 9. El sistema informa que se ha cambiado la contraseña
Extensiones	1. El sistema informa que el correo no tiene cuenta asociada 2. El usuario no introduce una contraseña válida.

Caso de uso	#4 Cerrar sesión perfil administrador
Actor principal	Perfil administrador
Precondiciones	Estar logueado en el sistema
Disparador	El perfil administrador quiere cerrar sesión en el sistema
Escenario principal	1. El usuario hace clic en cerrar sesión 2. El sistema desloguea la sesión del usuario 3. El sistema redirige a la pantalla principal del aplicativo
Extensiones	

Caso de uso	#5 Acceder al sistema en perfil usuario
Actor principal	Perfil cliente/administrador
Precondiciones	-
Disparador	El cliente quiere acceder al aplicativo
Escenario principal	1. El cliente accede a la url de la aplicación 2. El sistema muestra el contenido de la página principal de la aplicación
Extensiones	

Caso de uso	#6 Dar de alta una incidencia a través de la web
Actor principal	Perfil cliente/administrador
Precondiciones	-
Disparador	El usuario quiere reportar una incidencia desde la web
Escenario principal	1. El usuario quiere reportar una incidencia desde el navegador 2. El usuario rellena el formulario de alta de incidencia 3. El sistema valida los datos introducidos por el usuario 4. El sistema genera un nuevo número de referencia (incidencia) 5. El sistema da de alta la incidencia 6. El sistema envía un correo notificando que se ha realizado el alta de la incidencia con su correspondiente número de referencia 7. El sistema notifica a la central que se ha dado de alta una incidencia
Extensiones	3. El sistema informa que los datos introducidos por el usuario son incorrectos 4. El sistema comprueba que una incidencia del mismo tipo en la misma zona está abierta. a. Caso de uso # Incidencia repetida 6. El usuario no recibe correo de alta de referencia a. El usuario clica sobre ‘volver a enviar correo’ b. El sistema vuelve a enviar correo con el número de referencia

`

Caso de uso	#7 Dar de alta una incidencia a través de código USSD
Actor principal	Perfil cliente
Precondiciones	-
Disparador	El usuario quiere reportar una incidencia des del móvil,
Escenario principal	1. El usuario quiere reportar una incidencia des del móvil vía código USSD 2. El usuario envía datos vía código USSD 3. El sistema responde al usuario vía USSD 4. La interacción entre el usuario y sistema terminará hasta realizar el alta de la incidencia 5. El sistema notifica a la central que se ha dado de alta una incidencia
Extensiones	

Caso de uso	#8 Dar de alta una incidencia con prioridad
Actor principal	Perfil cliente/administrador
Precondiciones	-
Disparador	El usuario quiere reportar una incidencia
Escenario principal	1. El usuario quiere reportar una incidencia con prioridad alta 2. El usuario introduce los datos 3. El sistema notifica a la central que se ha dado de alta la incidencia y notifica al servicio externo que se ha reportado una incidencia
Extensiones	3. La incidencia introducida es incorrecta, el administrador actualiza la incidencia a no valida y lo notifica al servicio externo

Caso de uso	#9 Categorizar incidencia
Actor principal	Perfil administrador
Precondiciones	Incidencia creada
Disparador	Al crear una incidencia, categorizar nivel de prioridad
Escenario principal	1. El sistema reporta que se ha creado una incidencia 2. El usuario corrobora que la incidencia creada tiene los parámetros estándares 3. El usuario valida la incidencia 4. El usuario categoriza el nivel de prioridad 5. El sistema estima la el tiempo en el que se podrá resolver la incidencia
Extensiones	1. El usuario no valida la incidencia creada 2. La incidencia pasa ha estado no válido 3. El sistema actualiza en base de datos el estado de la incidencia

Caso de uso	#10 Notificar incidencia a operadores externos
Actor principal	Incidencia
Precondiciones	La incidencia es validado por el perfil administrador o se actualiza incidencia
Disparador	El sistema notifica a los operadores externos para que resuelvan la incidencia
Escenario principal	1. La incidencia cambia de estado 2. El sistema genera una notificación al servidor externo apropiado
Extensiones	

Caso de uso	#11 Gestionar incidencia
Actor principal	Perfil cliente/administrador
Precondiciones	El usuario tiene que estar logueado
Disparador	El usuario quiere gestionar incidencia
Escenario principal	1. El usuario busca incidencia que quiere modificar 2. El sistema muestra todos los datos de la incidencia 3. El usuario actualiza la incidencia 4. El sistema actualiza en base de datos las modificaciones 5. El sistema notifica a servicios externos las actualizaciones producidas
Extensiones	1. El usuario hace un cambio incorrecto 2. El sistema notifica que ese cambio no se puede realizar

Caso de uso	#12 Consultar listado de incidencias
Actor principal	Perfil cliente/administrador
Precondiciones	-
Disparador	El usuario quiere buscar el listado de incidencias
Escenario principal	1. El usuario accede a la pestaña de incidencias 2. El sistema devuelve el listado de incidencias disponibles
Extensiones	1. El sistema no devuelve el listado

Caso de uso	#13 Consultar incidencia por categoría
Actor principal	Perfil cliente/administrador
Precondiciones	-
Disparador	El usuario quiere buscar incidencias por categoría
Escenario principal	1. El usuario accede a la pestaña de incidencias 2. El sistema filtra por categoría 3. El sistema devuelve listado de incidencias por el filtro de categoría
Extensiones	1. El sistema no devuelve el listado

Caso de uso	#14 Consultar incidencia por estado
Actor principal	Perfil cliente/administrador
Precondiciones	-
Disparador	El usuario quiere buscar incidencias por estado
Escenario principal	1. El usuario accede a la pestaña de incidencias 2. El sistema filtra por estado 3. El sistema devuelve listado de incidencias por el filtro de estado
Extensiones	1. El sistema no devuelve el listado

Caso de uso	#15 Consultar incidencia por fecha
Actor principal	Perfil cliente/administrador
Precondiciones	-
Disparador	El usuario quiere buscar incidencias por fecha
Escenario principal	1. El usuario accede a la pestaña de incidencias 2. El sistema filtra por fecha introducida 3. El sistema devuelve listado de incidencias por el filtro de fecha
Extensiones	1. La sistema informa que la fecha introducida es incorrecta 2. El sistema no devuelve el listado

Caso de uso	#16 Consultar incidencia por rango de fechas
Actor principal	Perfil cliente/administrador
Precondiciones	-
Disparador	El usuario quiere buscar incidencias por rango de fecha
Escenario principal	1. El usuario accede a la pestaña de incidencias 2. El sistema filtra por rango de fechas introducida 3. El sistema devuelve listado de incidencias por el filtro de fecha
Extensiones	1. La sistema informa que el rango de fechas introducidas son incorrectas 2. El sistema no devuelve el listado

Caso de uso	#17 Consultar incidencia por referencia de incidencia
Actor principal	Perfil cliente/administrador
Precondiciones	-
Disparador	El usuario quiere buscar incidencias por referencia
Escenario principal	1. El usuario accede a la pestaña de incidencias 2. El sistema muestra información relativa a la incidencia.
Extensiones	1. El sistema informa que los datos introducidos son incorrectos

Caso de uso	#18 Detalle incidencia
Actor principal	Perfil cliente/administrador
Precondiciones	Pestaña búsqueda incidencia
Disparador	El usuario quiere visualizar el detalle de una incidencia
Escenario principal	1. El usuario selecciona la incidencia de la que quiere ver el detalle 2. El sistema busca la información del detalle 3. El sistema muestra los datos de la incidencia
Extensiones	

Caso de uso	#19 Comentario incidencia con aviso
Actor principal	Perfil cliente/administrador
Precondiciones	Pantalla detalle incidencia
Disparador	El usuario quiere comentar una incidencia
Escenario principal	1. El usuario introduce un comentario y añade un medio por el cual le llegarán notificaciones respecto a esa incidencia 2. El sistema valida los datos introducidos 3. El sistema muestra mensaje de comentario añadido correctamente 4. La incidencia notificará al usuario cada vez que cambie de estado
Extensiones	

Caso de uso	#20 Comentario incidencia sin aviso
Actor principal	Perfil cliente/administrador
Precondiciones	Pantalla detalle incidencia
Disparador	El usuario quiere comentar una incidencia
Escenario principal	1. El usuario introduce un comentario 2. El sistema valida los datos introducidos 3. El sistema muestra mensaje de comentario añadido correctamente
Extensiones	

Caso de uso	#21 Eliminar incidencia
Actor principal	Perfil administrador
Precondiciones	- El administrador tiene que estar logueado
Disparador	El usuario quiere eliminar una incidencia
Escenario principal	1. El usuario accede al listado de incidencias 2. El usuario elimina una incidencia 3. La incidencia es eliminada del sistema

Caso de uso	#22 Notifica nueva incidencia
Actor principal	Sistema
Precondiciones	-
Disparador	Se crea una incidencia
Escenario principal	1. El usuario crea una incidencia 2. El sistema verifica el tipo de incidencia 3. Según el tipo de incidencia el sistema notifica a la institución correspondiente 4. La institución recibe una notificación de una nueva incidencia sobre la que debe actuar

Caso de uso	#23 Notificar incidencia cerrada/solucionada
Actor principal	Sistema
Precondiciones	El usuario introdujo su correo electrónico al abrir la incidencia
Disparador	Se cierra una incidencia
Escenario principal	1. Un administrador cierra una incidencia. 2. El sistema manda un correo al usuario que abrió la incidencia

Caso de uso	#24 Adjuntar informe
Actor principal	Perfil administrador/servicio externo
Precondiciones	Existe incidencia
Disparador	Añadir informe en un incidencia
Escenario principal	1. Acceder a una incidencia 2. Pulsar botón, Añadir informe 3. Introducir documento (formato pdf) 4. Pulsar botón, Guardar 5. El sistema añade el documento en la incidencia

Caso de uso	#25 Reportar incidencia resuelta
Actor principal	Perfil servicio externo
Precondiciones	Existe incidencia
Disparador	Incidencia solucionada
Escenario principal	1. El usuario reporta que la incidencia ha sido solucionada 2. El sistema actualiza el estado de la incidencia 3. El sistema notifica al administrador de la institución y en caso de que la incidencia tenga usuario registrado, informar al usuario

6.5- Requisitos no funcionales

Tipo de requisito	Personalización e internalización
Número de requisito	1
Descripción	El sistema tiene que estar disponible en español
Justificación de requisito	Al ser un prototipo, se comenzará realizando en la lengua española
Criterio de satisfacción	La webApp estará en castellano

Tipo de requisito	Aprendizaje
Número de requisito	2
Descripción	El sistema no tiene que requerir de ninguna formación previa.
Justificación de requisito	Los usuarios no tienen que conocer todo el sistema para poder utilizarlo
Criterio de satisfacción	Se realizarán pruebas en Maputo del sistema, el 85% de los que realicen la prueba tienen que validar que no se necesita ninguna formación

Tipo de requisito	Fiabilidad y disponibilidad
Número de requisito	3
Descripción	La aplicación tiene que estar siempre operativa
Justificación de requisito	El sistema tiene que estar a disposición de recibir una incidencia
Criterio de satisfacción	El hosting estará en un servidor que se instalará en Maputo.

Tipo de requisito	Seguridad
Número de requisito	4
Descripción	El usuario administrador necesitará un usuario y contraseña
Justificación de requisito	Para poder gestionar las incidencias, se necesitará tener perfil administrador
Criterio de satisfacción	Solo los perfiles administradores podrán gestionar las incidencias

Tipo de requisito	Seguridad
Número de requisito	5
Descripción	Los permisos de acceso al sistema podrán ser cambiados solamente por el administrador de acceso a datos.
Justificación de requisito	Solo los administradores podrán gestionar las incidencias
Criterio de satisfacción	Perfil administrador podrá modificar los estados de las incidencias

Tipo de requisito	Look and feel
Número de requisito	6
Descripción	El sistema tiene que tener la misma base que la aplicación Mopa
Justificación de requisito	Los usuarios de Maputo ya conocimiento de esta aplicación
Criterio de satisfacción	Se realizarán pruebas con usuarios de Maputo, el 90% de los usuarios tienen que realizar las mismas funcionalidades que la aplicación sin cometer error

Tipo de requisito	Interfaz
Número de requisito	7
Descripción	El sistema tiene que ser multiplataforma
Justificación de requisito	Se tiene que poder acceder al sistema desde cualquier plataforma
Criterio de satisfacción	El sistema tiene que garantizar ser responsive dependiendo de la dimensión del dispositivo des del cual se accede

Tipo de requisito	Implantación
Número de requisito	8
Descripción	El sistema no se tiene que instalar
Justificación de requisito	Al tratarse de una webApp, no requiere descargar (Play Store, IOS) e instalar
Criterio de satisfacción	El sistema tiene que ser accesible desde cualquier navegador (últimas versiones y estables). No requiere ninguna instalación previa

Tipo de requisito	Implantación
Número de requisito	9
Descripción	El servidor y componentes deben ser portables en plataformas GNU/Linux y Windows, con máquinas que presentan arquitecturas de 64 bits
Justificación de requisito	El sistema que se está desarrollando tiene que ser implantado y tener un mantenimiento por personas en Maputo
Criterio de satisfacción	El sistema que se desarrollará tendrá que ser instalada en las oficinas de Maputo

Tipo de requisito	Implantación
Número de requisito	10
Descripción	Las interfaces de comunicación deben contener los estándares Web y fundamentalmente se deben basar en protocolos HTTP
Justificación de requisito	Se tiene que definir un estándar para la comunicación entre la capa presentación y dominio
Criterio de satisfacción	Las peticiones seguirán los protocolos HTTP, las respuestas tienen que ser JSON

Tipo de requisito	Usabilidad
Número de requisito	11
Descripción	El sistema debe proporcionar mensajes de error que sean informativos y orientados al usuario final.
Justificación de requisito	El sistema tiene que informar de los errores que va cometiendo el usuario
Criterio de satisfacción	El sistema validará los inputs que va introduciendo el usuario, en caso de error, mostrar mensaje de error

Tipo de requisito	Usabilidad
Número de requisito	12
Descripción	El sistema debe contar con manuales de usuario
Justificación de requisito	
Criterio de satisfacción	El sistema tiene que tener un apartado ‘guía de usuario’

Tipo de requisito	Rendimento
Número de requisito	13
Descripción	El sistema no debe tardar más de diez segundos en mostrar los resultados de una búsqueda
Justificación de requisito	El sistema no tiene que tardar más de 10 segundos en dar respuesta al usuario
Criterio de satisfacción	Las peticiones que se realizarán al BackEnd esperarán como mucho 10 segundos

7- Implementación

7.1- Tecnologías utilizadas

Estas tecnologías son las principalmente usadas para desarrollar el back end y la base de datos.

- **Java:** Será el lenguaje escogido para el back end. Hay muchas aplicaciones y sitios web que funcionan con Java y cada día se crean más. Java es rápido, seguro y fiable. Como desarrollador ya he trabajado con Java, y para desarrollar la lógica del servidor hay muchas tecnologías con Java que nos van a facilitar el desarrollo.

- **Java Persistence API (JPA)**: La aplicación usará JPA para conectarse con la Base de Datos desde el Backend. Es un framework del lenguaje de programación Java que maneja datos relacionales.

- **QueryDSL**: Junto con JPA, esta tecnología nos permitirá simplificar la abstracción de las querys con Java hacia nuestra Base de Datos MySQL.

- **Spring framework:** Este framework open source facilita la creación de aplicaciones en Java. La inyección de dependencias de Spring es uno de los motivos por los que es más conocido. Nos permite crear controladores web para aplicaciones REST, nos proporciona abstracciones sobre Hibernate que nos facilitará el acceso a los datos. También contiene un framework de testing con soporte para JUnit.

- **Hibernate**: Hibernate es una herramienta de mapeo objeto-relacional para Java que facilita el mapeo de atributos entre una base de datos relacional (MySQL) y el modelo de objetos de una aplicación. Gracias a hibernate podremos manipular objetos java que representan las entidades que después serán persistidas en las tablas de nuestra BD.

- **JSON Web Token (JWT)**: Ésta tecnología ha sido usada en el proyecto para la autentificación de usuarios.

- **JUnit:** Esta tecnología de Testing no permitirá desarrollar los tests en Java.

- **H2**: Es un sistema administrador de bases de datos relacionales programado en Java. H2 nos facilitará la simulación de datos para nuestros tests en local.

- **Maven:** Maven es una herramienta de software para la gestión y construcción de proyectos Java. Tiene un modelo de configuración de construcción simple, basado en un formato XML. Esta tecnología nos ayudará a compilar y empaquetar el programa.

- **MySQL:** Está considerada como la base de datos de código abierto más popular del mundo. Desarrollaremos nuestra base de datos relacional con esta tecnología, que junto con Spring, nos dará muchas facilidades para la conexión con nuestro Backend.

- **AWS Elastic Beanstalk**: es un servicio usado para implementar y escalar servicios y aplicaciones web desarrollados con Java. Con este servicio de AWS hemos desplegado nuestro backend en un archivo ".jar" a un servidor configurado para ser accesible públicamente.

- **Amazon RDS**: con este servicio de AWS hemos creado nuestra BD en un servidor. Este servicio se conecta fácilmente con nuestro Backend.

Para el desarrollo se han utilizado los siguientes programas:

- **IntelliJ**: Uno de los IDEs más populares. Se ha usado la versión de la comunidad (versión gratuita). Desde este IDE se ha programado el backend en Java. También nos ha servido como interfaz para el control de versiones.

- **Postman**: Con esta herramienta se ha documentado las llamas HTTP para comunicarse con nuestro backend. Desde esta plataforma se han enviado la mayor parte de las llamadas al backend a modo de prueba, tanto en localhost como al servidor aws.

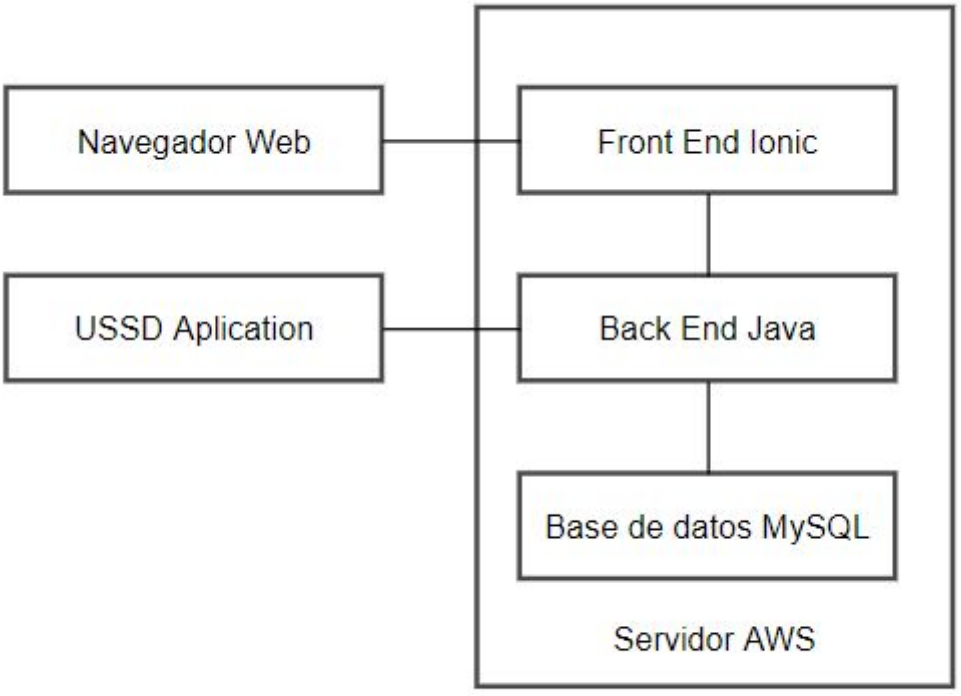
- **MySQL Workbench**: Este programa lo hemos utilizado para conectarnos directamente a nuestra BD. Desde aquí se han ejecutado las querys en MySQL para crear las tablas e ir adaptándolas según la conveniencia durante el proceso de desarrollo.

- **Git**: se ha utilizado este software para el control de versiones de este proyecto.

- **Bitbucket:** es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de versiones GIT. Hemos utilizado este servicio como repositorio Git para nuestro código.

7.2- Arquitectura del sistema

El sistema seguirá un patrón de arquitectura monolítica. Este tipo de arquitectura es frecuentemente usado para aplicaciones en las primeras fases de desarrollo de equipos pequeños, así que calzará perfectamente en nuestro caso. Nuestro sistema podría escalarse a un sistema de microservicios. Esto sólo se propondría en un futuro, cuando haga falta escalar nuestro sistema por un crecimiento del número de usuarios y su



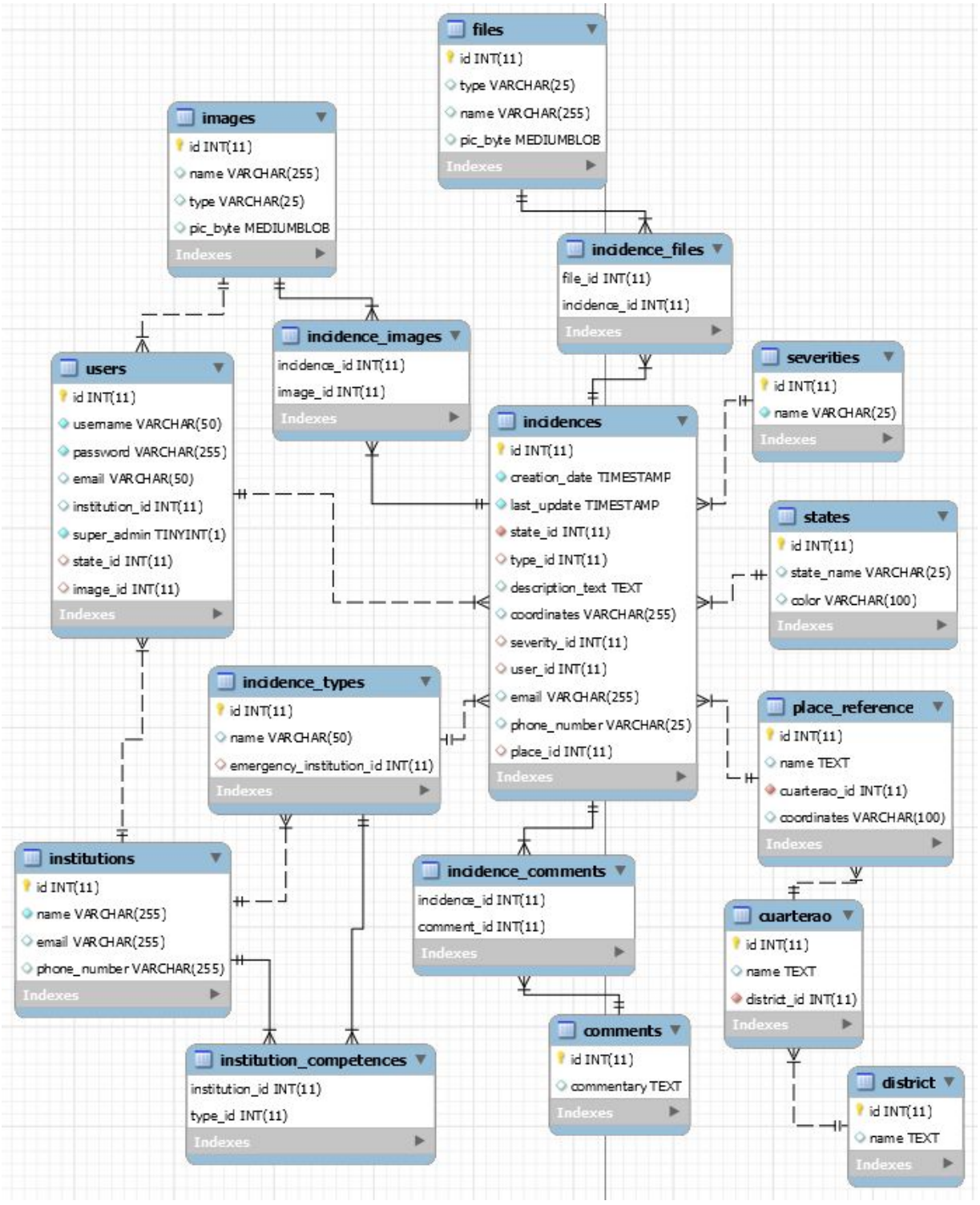
En este esquema podemos encontrar las principales partes que formarán nuestro sistema. Todos los servicios desarrollados del sistema han sido desplegados en servidores AWS, menos la aplicación USSD.

El **Font End**, desarrollado por Eynar, será el acceso desde el ordenador que tendrán los usuarios, para reportar incidencias como ciudadano o administrador o para gestionarlas como administrador. Ha sido desarrollado con Iónic.

El **Back End** recibirá peticiones, tanto del frontend como de la aplicación USSD. El back end se ha desarrollado en Java con Spring Boot. Se ha desarrollado como una API Rest, de manera que tanto el front end como la aplicación USSD puedan ejecutar la lógica del sistema con simples peticiones HTTP. El backend utilizará una base de datos en MySQL, también desplegada en AWS.

7.3- Diagrama ER (Entidad-Relación)

Este diagrama muestra el modelo desarrollado para nuestra base de datos, tal cual está creada en MySQL. Muestra la relación entre las entidades. A parte de las tablas de conexión, como incidence_images, incidence_files, institution_competeneces, incidence_comments, todas las otras tablas corresponden a una entidad del sistema. También comentar que no se ha dedicado tiempo a optimizar èl rendimiento creando ningún índice ni procedimiento de almacenamiento, dado que se trata de un prototipo.

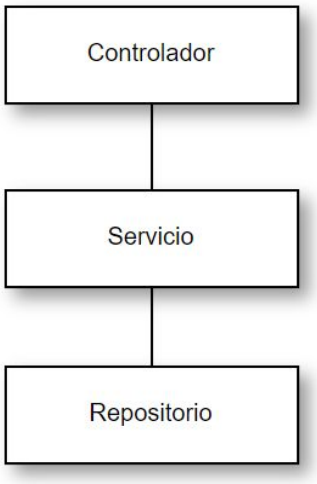


7.4- Backend

El desarrollo de este Backend se ha orientado a construir una REST API que pueda proporcionar tanto al frontend desarrollado por Eynar como a la aplicación USSD. Se ha reducido la comunicación a unas llamadas (o endpoints) HTTP. Con esas llamadas cualquier unidad que pueda hacer llamadas HTTP puede comunicarse directamente a nuestro backend, lo que lo hace un servicio independiente de un frontend específico. Tanto el frontend como la aplicación USSD son capaces de ejecutar todas las funcionalidades del sistema usando nuestra Rest API.

La tecnología Spring Boot ha sido de gran ayuda para desarrollar este servicio. Con Spring y Maven se han inyectado todas las dependencias, como JDBC, JUnit... Usando Java hemos podido implementar la lógica del sistema.

Para la organización de las clases y la lógica, se ha programado siguiendo el patrón Repository-Service. La principal característica de este patrón es usar los Servicios como capa de Negocio. Los Servicios serán los encargados de usar la capa de Repositorios para persistir los datos o consultarlos. En nuestro habrá una capa de Controladores que serán los que reciban las llamadas HTTP y llamarán a la capa de Servicios para ejecutar las acciones deseadas por el cliente. Este sería un esquema básico de la arquitectura del backend, siguiendo el patrón mencionado:



Controller - Sería como la interfaz del Backend. Recibe llamadas HTTP y hace llamadas a la capa de Servicios.

Service - Capa de Negocio. Ejecuta la lógica y hace las llamadas a la capa de Repositorios. Los Servicios retornar el resultado a la capa de Controladores.

Repository - Sería la capa de persistencia. Persiste los datos necesarios y hace las consultas a BD. Retorna a la capa de Servicios los resultados de sus llamadas.

Se ha desarrollado todo el Backend en Inglés, el idioma más internacional y más utilizado para programar. De esta manera nuestras clases tendrán nombres como IncidenceController, InstitutionService o UserRepository.

7.4.1- Dominio

El Dominio del sistema, definido principalmente con el Diagrama de clases, estará representado en Entidades y objetos DTO.

Los elementos sobre los que recae la lógica, se han definido como **Entidades** (Entity). Con JPA hemos mapeado las diferentes tablas de BD de tal manera que cada entidad corresponda a una tabla. Cuando guardamos una Entidad en base de datos, JPA es capaz de mapear los parámetros definidos en la clase con las columnas en BD. De la misma forma, al hacer una consulta en BD, los datos devueltos también serán mapeados en nuestros objetos Entidad.

Mostraremos la clase Institution como ejemplo de la definición de una Entidad.

```
12
13  @Entity
14  @Getter @Setter
15  @Table(name = "institutions")
16  public class Institution {
17      @Id
18      @GeneratedValue(strategy=GenerationType.IDENTITY)
19      private Long id;
20
21      private String name;
22
23      private String email;
24
25      private String phoneNumber;
26
27      @ManyToMany
28      @JoinTable(
29              name = "institution_competences",
30              joinColumns = @JoinColumn(name = "institution_id"),
31              inverseJoinColumns = @JoinColumn(name = "type_id"))
32      private Set<IncidenceType> competences;
33
34  }
35
```

La clave está en las etiquetas. *@Entity* señala que esta clase se trata de una Entidad. *@Table* mapea la entidad con la tabla en base de datos. Usando los nombres de las columnas en los mismos parámetros, JPA mapeará automáticamente los valores correspondientes. Con la etiqueta *@Id* señalamos la primary key, y señalamos también que el valor del Id para los nuevos valores se generará automáticamente.

Con las etiquetas *@ManyToMany* y *@JoinTable* seremos capaces de usar la tabla institution_competences para mapear los competencias de la institución en una lista de tipos de incidencias. Tendremos que definir el nombre de la tabla intermediaria y las columnas que corresponden a cada una de las partes de la Join.

A parte de los objetos Entidad, nuestro sistema también usará los objetos tipo **DTO** (Data Transfer Object). Éstos son simplificaciones de los objetos Entidad, reduciendo todos los atributos de las entidades a tipos básicos. Los DTO son usados para facilitar la comunicación con el frontend. Al hacer el frontend un PUT, por ejemplo, para editar una institución, el backend recibirá los datos tal y como se presentan en el DTO, con tipos simples fáciles de usar en una comunicación HTTP con un JSON como Body.

```
8   @Getter @Setter
9   public class InstitutionDTO {
10
11      private Integer id;
12
13      private String name;
14
15      private String email;
16
17      private String phoneNumber;
18
19      private List<Integer> competences;
20  }
```

Como podemos ver, la definición de nuestro elemento Institución en DTO es mucho más simple que en el caso de la Entidad. Las competencias se han reducido a una lista de Integers. Estos Integers serán los IDs de los tipos de incidencia que forman parte de las competencias de la institución.

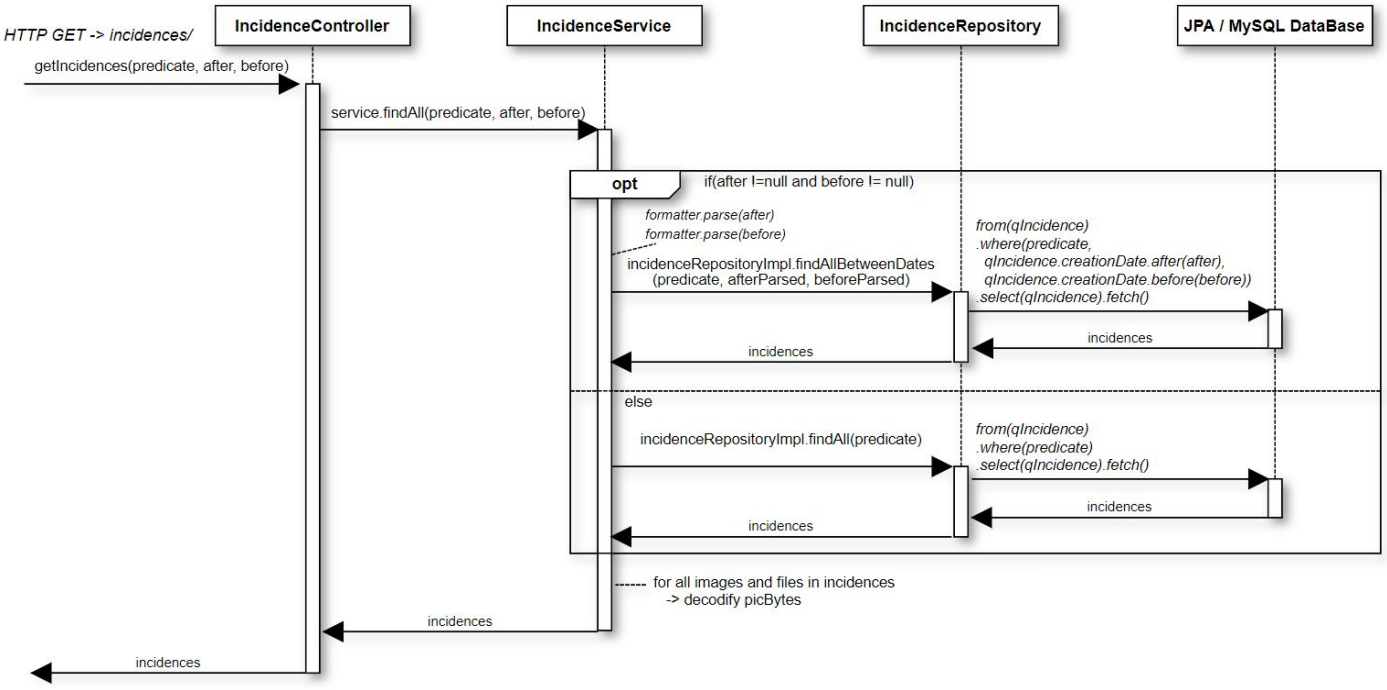
Respecto a las etiquetas *@Getter* y *@Setter*, que se pueden encontrar tanto en las clases DTO como en las Entidades, simplemente son una funcionalidad de Lombock, una tecnología inyectada en las dependencias del proyecto que nos facilitará la vida a la hora de definir este tipo de clases. Gracias a Lombock nos ahorraremos tener que definir todos los gets y sets que corresponden a cada uno de los atributos. Una gran comodidad para cualquier programador.

Así pues, las clases de Servicios se encargarán de transformar los objetos DTO recibidos en los Controladores a Entidades que puedan ser persistidas con JPA desde los Repositorios.

7.4.2- Diagramas de secuencia explicados

Para ejemplificar las relaciones entre clases y el uso que les daremos, mostraremos algunos diagramas de secuencia desde para representar la llamada HTTP hasta la conexión con BD. Aunque cada llamada tiene detalles diferentes, las llamadas que podemos encontrar explicadas en los diagramas nos pueden servir para ilustrar el comportamiento que podremos observar en la gran parte de las entradas HTTP a nuestro backend.

- getIncidences (HTTP GET /incidences)



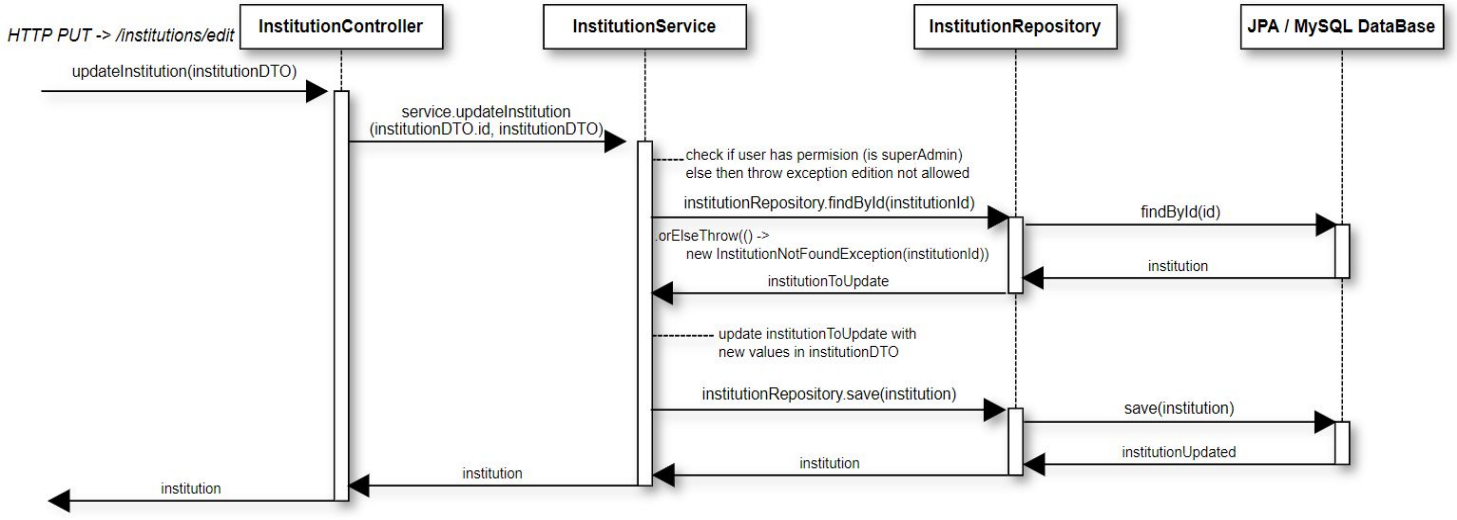
El Controlador recibe la llamada HTTP GET “/incidences”. En esta llamada recibe **predicate, after** y **before**. Los argumentos *after* y *before* son Dates, mientras que *predicate* es un objeto que recibe otros atributos que se puedan haber enviado como parámetros en la llamada get. Los atributos *after* y *before* pueden ser null. Si no se han pasado atributos extra como parámetros de la llamada, *predicate* simplemente estará vacío. El usuario puede hacer una llamada añadiendo un parámetro para filtrar por id, por ejemplo (“/incidences?id=2”). De tal manera pueden filtrarse cualquiera de los atributos del objeto de incidencia, tal y como son definidos en la Entidad Incidence. Éstos atributos añadidos como

parámetros estarán encapsulados en el objeto *predicate*, y junto la librería de QueryDSL, harán que los filtros añadidos en las consultas sean mucho más fáciles de usar.

La función findAll de Service mostrada en el diagrama diferencia entre si *after* y *before* son null o no. Si el usuario ha querido filtrar por fechas, éstas deben ser formateadas a TimeStamp para que el Repositorio pueda usarla, y pasarlas como parámetro en la llamada a la función findAllByDate de IncidenceRepository. Si el usuario no ha querido usar el filtro por fecha, la llamada a repositorio será más simple y no hará falta pasar más parámetros que *predicate* al repositorio.

Cuando el repositorio envía la query (con QueryDSL), JPA transforma el resultado de MySQL a las entidades definidas en Java. Recibiremos directamente una lista de objetos Incidence. El servicio al recibir las incidencias, ha de decodificar las imágenes y documentos, para que el frontend pueda usarlos. Luego retorna las incidencias al controlador, que las devolverá como una lista de objetos Incidence transformada a JSON.

- updateInstitution (HTTP PUT /institutions/edit)



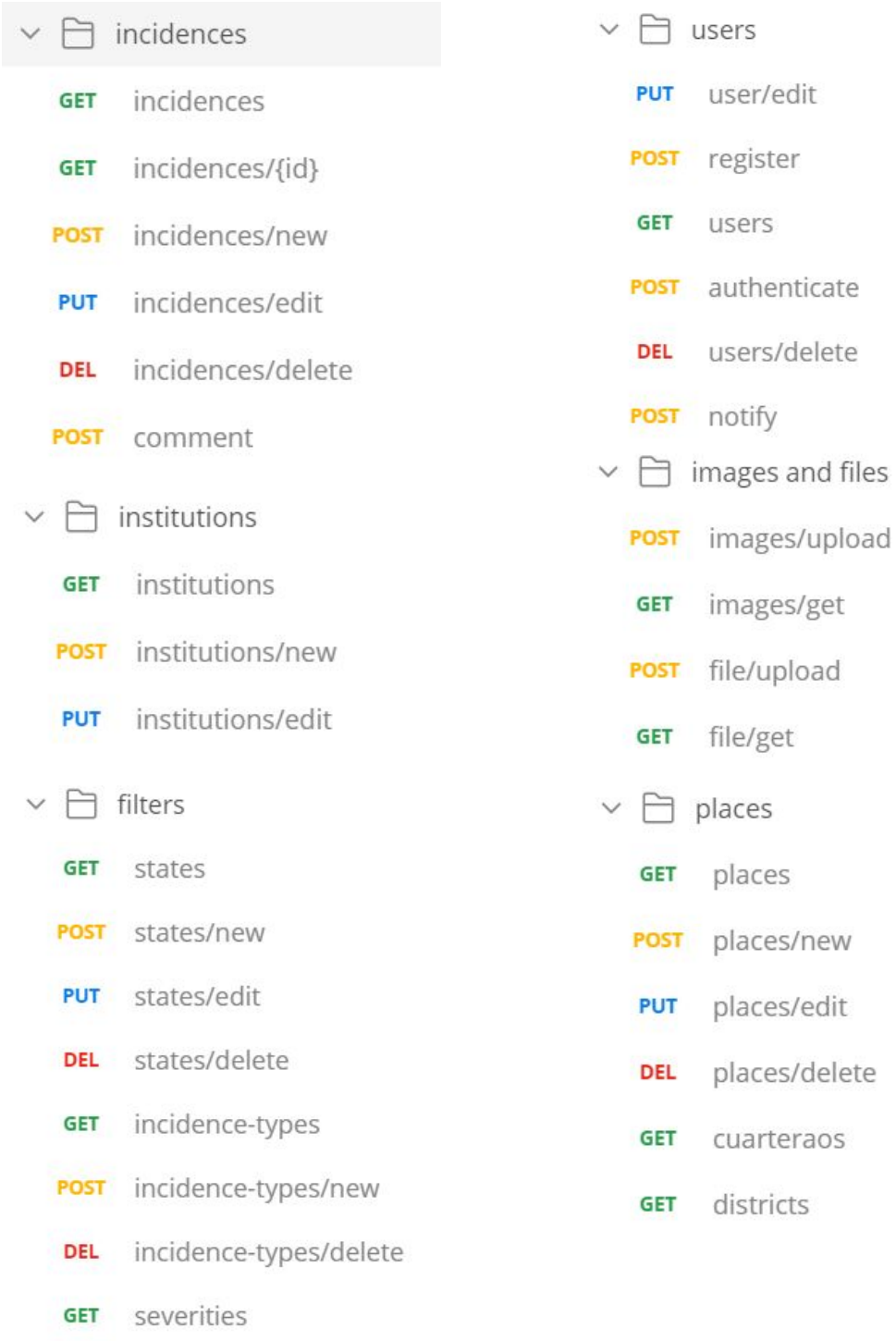
Como hemos ejemplificado anteriormente, en este caso la llamada HTTP también es recibida por el Controlador. El cliente hará una llamada HTTP PUT “/institution/edit” con Body. En el body se especificarán los datos de la incidencia, con los parámetros a editar ya actualizados. El formato de entrega de estos datos será con un DTO con tipos sencillos para facilitar su envío a través del body.

Seguidamente de ejecutarse updateInstitution en controlador, éste llamará al servicio.Lo primero que comprueba el servicio son los permisos de edición de instituciones del usuario. En caso de no tratarse de un superAdmin, se lanzará una excepción. Si los permisos son correctos, se cargará la institución en su estado actual de la base de datos. (en caso de no encontrar la institución se lanzaría una excepción del tipo InstitutionNotFoundException). Una vez obtenida la institución que se desea actualizar, se le setean los datos con los valores proporcionados en el DTO. Una vez tengamos el objeto institución con los valores deseados llamaremos al repositorio para guardar los cambios. Devolveremos la Institución resultante, devuelta por la ejecución de la función save. El cliente recibirá el objeto tal y como se encuentra ahora en BD, supuestamente con los valores editados deseados.

Nótese que en este caso la consulta a BD es más simple. En el caso anterior, queríamos poder usar los filtros en las llamadas a BD, pudiendo filtrar por cualquiera de los parámetros de “incidencia”. En este caso solo queremos filtrar por id, y la clase Repository tiene una función definida con esta utilidad. De la misma forma, la función “save” del repositorio también estará predefinida.

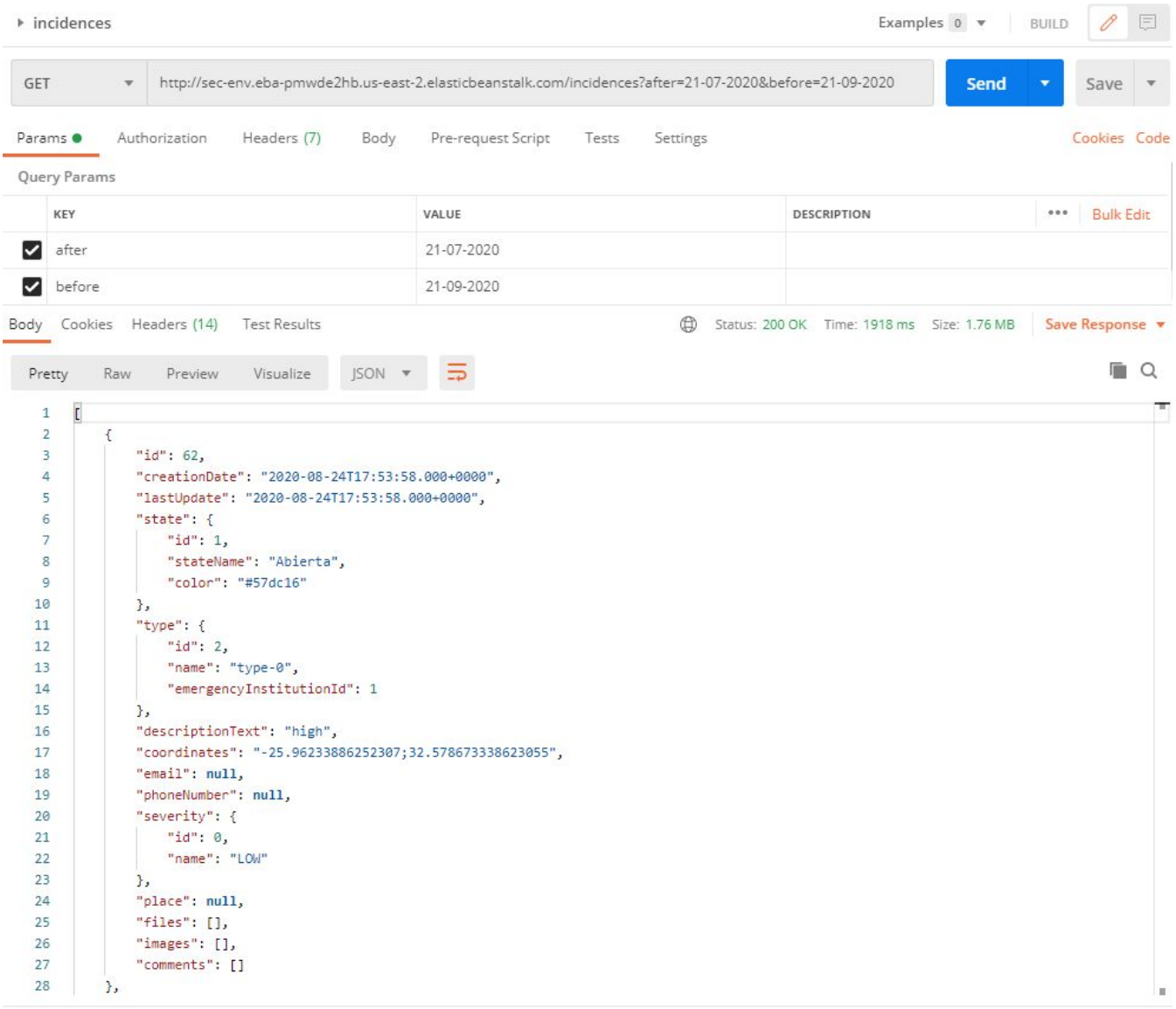
7.4.3- API Endpoints

Los Endpoints son las definiciones de las llamadas HTTP que nuestro backend va a poder recibir. Éstos han sido listado en la herramienta Postman. Desde ahí se han definido y ejemplificado cada una de las llamadas, a parte de testeado. Cada una de estas llamadas serán recibidas desde los controladores del backend, donde está programado el comportamiento deseado en cada caso. A continuación muestro el listado de llamadas que podemos encontrar organizado en carpetas en nuestro Postman:



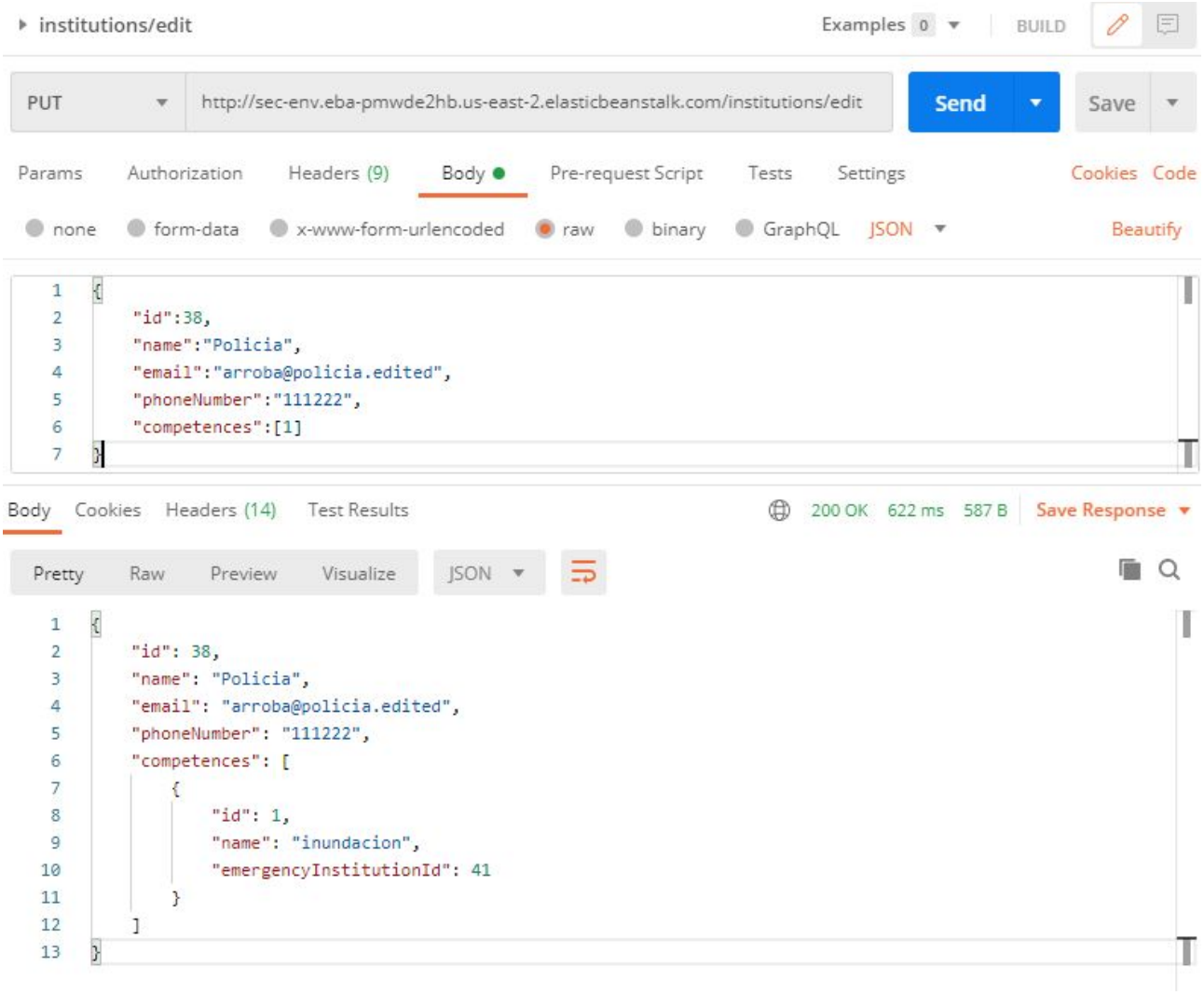
Postman ha sido una herramienta clave para el desarrollo, ya que no solo se ha podido testear el Backend haciendo llamadas HTTP de una forma sencilla, sino que se ha podido documentar el listado de llamadas mientras se ha ido desarrollando. La funcionalidad de compartir en Postman ha permitido que Eynar tuviera siempre el listado de llamadas y los ejemplos (como los que mostramos a continuación) actualizados para saber cómo comunicarse con el backend.

A continuación mostraremos ejemplos de los endpoints usando Postman:



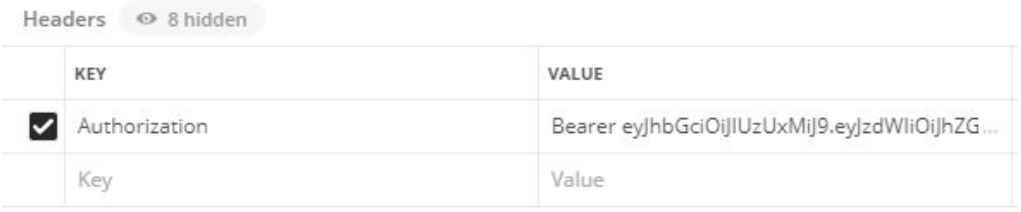
En este caso hemos ejecutado la llamada **GET /incidences**, con los parámetros after y before para definir el filtro de fechas. Podemos ver en el resultado una incidencia que forma parte del listado de incidencias devuelto por la petición. Podemos observar la composición del objeto Incidencia, con todos sus parámetros.

En el segundo ejemplo, ejecutaremos la llamada **HTTP PUT institutions/edit.**

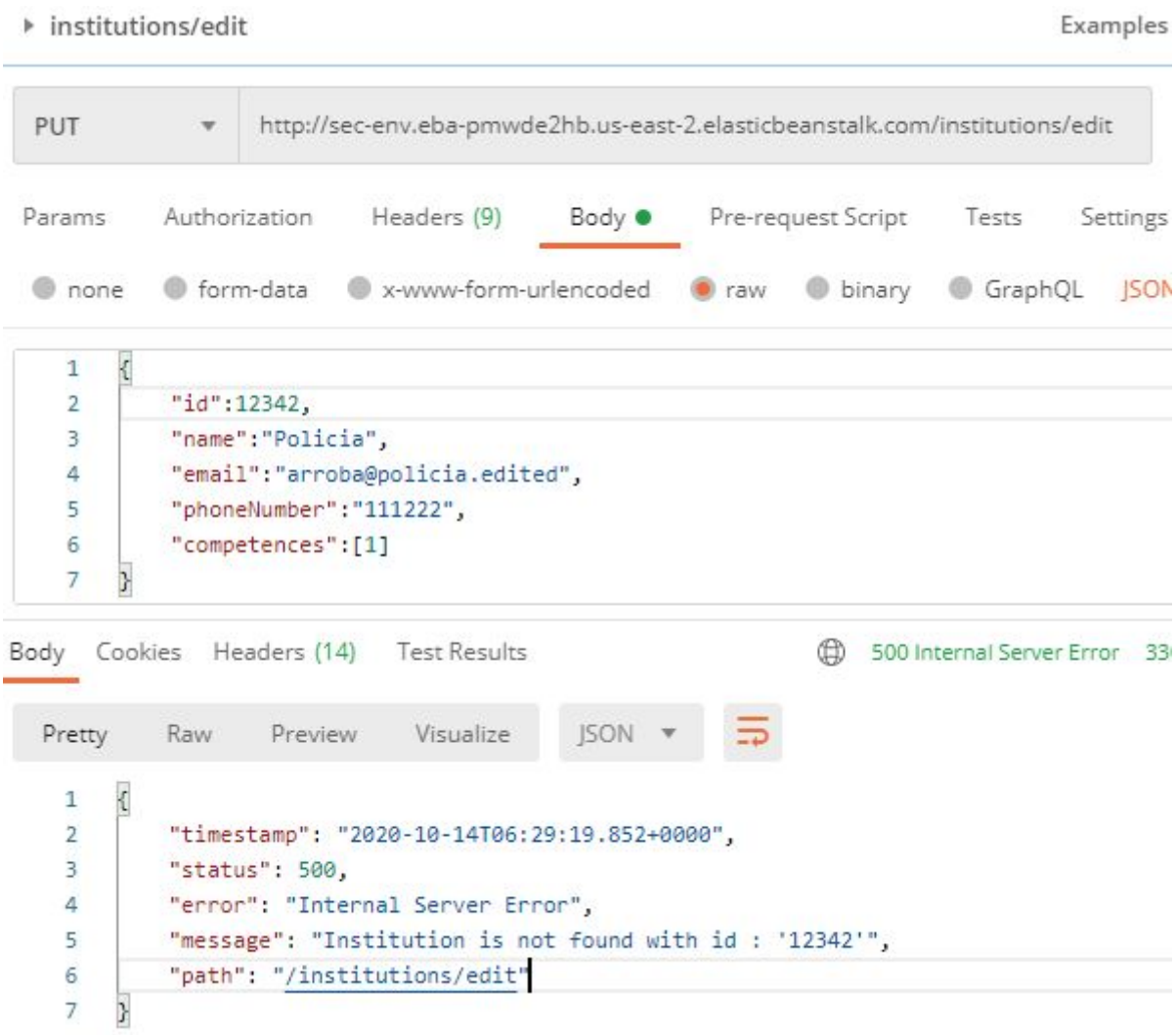


Podemos observar como hemos definido en el boyd en JSON los valores con los que queremos actualizar la institución. El apartado competences espera un listado de IDs, con los ids de las categorías de incidencia que la institución puede gestionar (las competencias de la institución). En el body de la respuesta podemos leer el objeto Institución resultante persistido en nuestra BD.

La autorización se ha añadido de la siguiente manera:



A continuación ejecutaremos la misma petición, pero cambiaremos el id de la institución a editar por un id inexistente en BD. De acuerdo con nuestro diagrama de secuencia, nos debería saltar un error del tipo InstitutionNotFound.



Efectivamente, el cambio no ha podido ser efectuado, ya que el id introducido no ha sido correspondido con ningún objeto en BD. La excepción muestra de forma clara cuál ha sido el problema por el que ha fallado la petición ejecutada.

La **colección de llamadas en Postman**, ha sido compartido con Eynar a través de este enlaze: https://www.getpostman.com/collections/898f4ddd37c26f92fcb5

El enlace es público, así que cualquiera que use Postman puede importar la colección con el enlace anterior y ejecutar las llamadas HTTP directamente hacia el backend, así como consultar todos los detalles i parámetros.

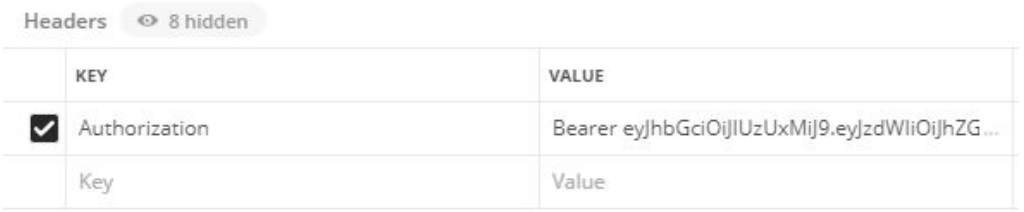
7.4.4- Autentificación

Se ha desarrollado la autentificación con la tecnología JWT (JSON Web Token).

En el Backend se han definido todas las llamadas que deben hacerse con autentificación en una clase nombrada *WebSecurityConfig.* El backend rechazará las llamadas que se hagan dentro de ese listado si no recibe ningún token, o si el token es incorrecto. En el código backend se puede encontrar un paquete *auth,* en él está definido y configurado el funcionamiento de este sistema de tokens.

El frontend, usando la llamada *authentificate*, se recibirá un token temporal. Éste token se podrá usar en las siguientes llamadas para que el backend pueda reconocer la identidad del usuario que está interactuando con el sistema.

En Postman, añadiremos el token de la siguiente manera:



7.4.5- Tests

Los tests se han desarrollado con JUnit y la ayuda de la librería Mokito, que nos permite mockear las funciones para poder testear solo las funcionalidades específicas en cada caso.

Para los Tests se ha creado un entorno seguro donde poder ejecutar las pruebas necesarias sin tener que consultar ni editar la base de datos real. Tenemos un archivo *data.sql*. Este archivo contiene un conjunto de inserts, los cuales generarán la BD de pruebas sobre la que se ejecutarán los tests. Gracias a el mapeo de las clases Entidad (comentado previamente en el apartado Dominio), nuestra aplicación ya tendrá una estructura de cómo está definida la Base de Datos, así que simplemente con los Inserts seremos capaces de crear nuestra BD de pruebas totalmente aislada de la real, donde podremos ejecutar nuestros tests sin modificar los datos reales.

7.5 Aplicación USSD

Para el desarrollo de esta aplicación hemos dedicado bastantes horas en entender la tecnología USSD y en la búsqueda de herramientas para su desarrollo. No hemos sido capaces de encontrar ningún operador que nos pudiera dar servicio en Maptuo. Por condiciones de operadores y servicios de terceros intentamos crear cualquier tipo de aplicación USSD que pueda ser simulada, pero con las herramientas que encontramos perdimos mucho tiempo sin resultados. Así que, mientras el desarrollo del frontend y backend seguía en proceso, decidimos que yo desarrollaría un ejecutable en consola que pudiera simular la comunicación con USSD.

Se ha desarrollado un prototipo de la aplicación en Java con Maven y Spring que mediante consola simula la interacción que se tendría desde un teléfono (usando sólo números como entrada), y que con dicha aplicación se llegue a dar de alta una incidencia. De esta forma la lógica desarrollada será útil para un futuro desarrollo que transforme la aplicación de consola en una aplicación USSD real.

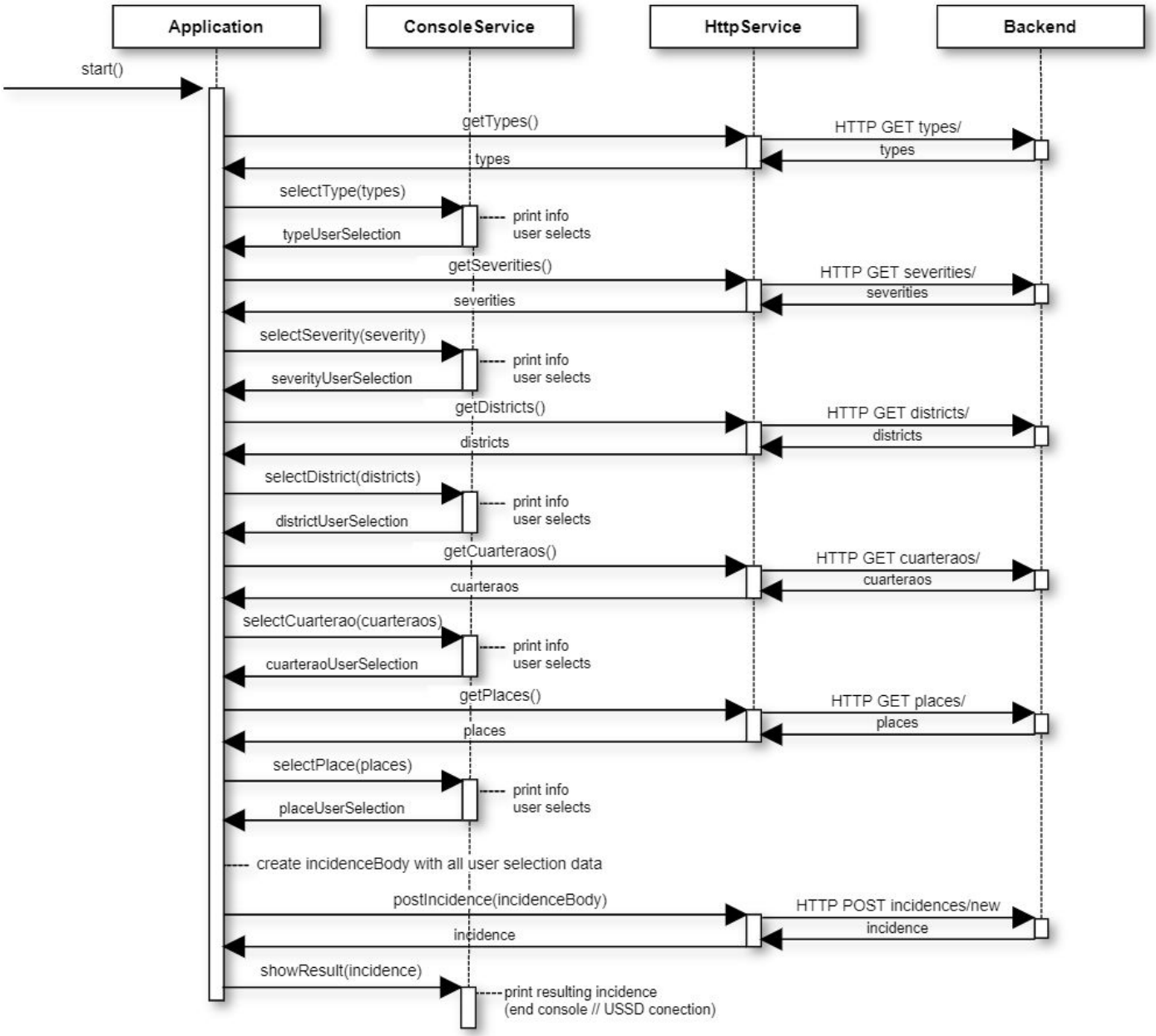
La aplicación se ha hecho con pensamiento modular. Se ha desarrollado de tal forma que sea fácil utilizar la misma aplicación con cambiar sólo la clase que se utiliza para la comunicación con el usuario desde el terminal. Si sustituimos la clase ConsoleService.java por una clase que realmente realice la interacción con el usuario a través de USSD, el resto de aplicación seguirá pudiendo ejecutar las llamadas HTTP para crear la incidencia a reportar.

Esta aplicación reutiliza las entidades tal y cómo se definieron en el Backend (sin ser mapeadas con ninguna BD, como ya se ha explicado en este documento). Con estas entidades podremos recibir todos los datos que tendremos que enseñar al usuario para que seleccione los valores que definirán la nueva incidencia.

7.5.1- Diagrama de secuencia aplicación completa

Se trata de una aplicación bastante simple, así que he intentado representar el comportamiento de nuestra aplicación en un diagrama de secuencia que explica cómo sería la ejecución normal de una llamada a nuestro servicio USSD. De acuerdo con este diagrama, si en un desarrollo futuro se pudiera sustituir la clase ConsoleService por una clase UssdService, el sistema estaría completo tal y como se deseaba desde un principio.

El sistema preguntará al usuario que tipo de incidencia quiere reportar, de que severidad se trata, y a continuación preguntará por que barrio, que quarterao (termino usado para la subdivisión de los barrios de Maputo) y que sitio de referencia específico se encuentra la incidencia. El sistema dará de alta una nueva incidencia con esos datos y enseñará al usuario los parámetros introducidos antes de finalizar la conexión.



8- Seguimiento del proyecto

Se han conseguido implementar todas las funcionalidades definidas en nuestro diseño y requisitos.

El desarrollo del backend y frontend han sido desarrollados simultáneamente, así que la comunicación con Eynar, responsable del frontend ha sido clave para el desarrollo del sistema. Eynar dependía de las llamadas que iba desarrollando yo en nuestro Backend, de tal forma yo iba proporcionándole las especificaciones de las llamadas y los objetos específicos mientras iba desplegando en el servidor los últimos avances. De esta forma Eynar podía ir pidiéndome correcciones mientras yo le iba actualizando con los últimos cambios desplegados.

Durante el desarrollo nos encontramos varios problemas que fuimos reportando en las reuniones con el director a medida que iban surgiendo.

8.1- Cambios Backend

La funcionalidad de **notificar**, se estableció que se desarrollaría en Backend. El desarrollo de esta funcionalidad ha llegado a funcionar localmente, el backend notificaba por email a los usuarios, pero tuve problemas para la configuración del backend en el servidor AWS.

No pude encontrar la manera de dar los permisos necesarios a mi aplicación para usar el servicio de envío de emails de AWS a cualquier dominio. El servicio SES de AWS me seguía devolviendo un error de Dominio, y no pude entender qué fallaba en mi aplicación y la configuración con toda la documentación consultada para que enviara un mail desde el servidor. Habiendo desarrollado ya el servicio de envío de Emails funcionando en el backend desde local, si en un futuro alguien con conocimientos sobre el servicio de mailin desde AWS, podría fácilmente usar el código que he dejado (deprecated) en el backend.

Eynar desarrolló un servicio externo en Heroku con el que el frontend podía notificar sin usar el backend. Hasta que no desarrolló él mismo esta funcionalidad, al final del desarrollo no pudo probar su frontend en ella. Debido a la intervención de Eynar, el sistema tiene implementada las funcionalidades usando notificaciones.

Debido a que el envío de mail no ha sido funcional hasta el final del desarrollo, la funcionalidad de **cambiar contraseña** a través del Mail se ha visto comprometida.

Durante el desarrollo, se propuso (al menos temporalmente), hacer que un administrador pudiera cambiar su contraseña pidiendolo a un superAdministrador. De esta manera los usuario con rol de superAdmin se encargan de gestionar el cambio de contraseñas. Por el entorno institucional en el que se usaría esta aplicación, el cambio propuesto podría ser

considerado, y sin poder enviar emails, se implementó en el backend la opción a un superAdmin de actualizar la contraseña de un usuario. Se pueden cambiar las contraseñas, de manera que el sistema podría ser funcional, aunque no corresponde con el diseño de requisitos deseado.

8.2- Cambios Aplicación USSD

El desarrollo con USSD supuso un problema, como se ha explicado en el apartado de Implementación / Aplicación USSD. No se pudo desarrollar una aplicación en USSD funcional, y se creó el ejecutable que simula la interacción que tendría el usuario usando el teléfono desde la consola de un ordenador. Nuestro objetivo era desarrollar un prototipo funcional desde teléfono, pero adaptamos el diseño del prototipo eliminando simplemente el uso de USSD, de forma que se pueda utilizar el diseño programado en un proyecto futuro.

8.3- Cambios Planificación

Por la pandemia mundial que estamos viviendo desde mediados de Marzo, se canceló el **viaje a Maputo** y las pruebas de uso en usuarios reales. Estas horas extra que nos hemos ahorrado se han utilizado en el desarrollo de nuestra aplicación.

El desarrollo no empezó a dar un sistema con alguna parte funcional hasta mediados de Mayo, y debido a que en Junio estaba prevista la entrega del trabajo, se decidió darnos más tiempo para asegurarnos que llegábamos con implementación del sistema que aún teníamos pendiente, sin dejar funcionalidades atrás por las prisas. Teníamos miedo de entregar un sistema con demasiadas carencias. Durante estos meses extra hemos estado desarrollando y probando el sistema hasta poco antes de la finalización, a finales de Octubre.

El motivo por el que la implementación del sistema no fué tan efectiva en primera instancia fue, tanto por conocimiento de las tecnologías y el entorno de desarrollo, como por el hecho de que tanto Eynar como yo estábamos trabajando a jornada completa, y la cantidad de horas que debíamos dedicar para sacar el desarrollo del sistema adelante nos hizo tener dudas de nuestra capacidad para entregar un proyecto aceptable considerando las fechas de entrega.

La mayoría de horas planificadas eran de desarrollo e implementación, y el motivo por el que aplazamos la entrega a Octubre fue por la falta de tiempo para el desarrollo e implementación. Consideramos que no hemos usado muchas horas extras para esta tarea en la extensión de la entrega, ya que no usamos todas las horas planificadas antes de junio. Sabíamos que la falta de tiempo iba a ser un riesgo para nuestro proyecto, y finalmente fue el motivo por el que se aplazó la entrega.

9- Conclusiones

9.1- Conclusiones personales

Como se ha comentado anteriormente, este trabajo ha sido desarrollado en dos partes. El trabajo ejecutado con Eynar ha sido duro, pero los resultados son suficientemente satisfactorios. Hemos sido capaces de desarrollar un sistema que cumple con los requisitos y que es funcional. En la primera fecha de entrega, tuvimos miedo de entregar un trabajo demasiado pobre, ya que la implementación estaba incompleta, pero con el tiempo extra que nos dieron y gracias a nuestra dedicación podemos decir que hemos entregado un prototipo bastante cerca de lo que debería ser el software final usado por las instituciones de Maputo.

El software resultante es capaz de registrar incidencias digitalmente, y facilita su gestión a los administradores que tengan que resolverlas. Ha cumplido con los objetivos principales que se plantearon.

Nunca había tenido la oportunidad de desarrollar un sistema de backend desde 0, y la experiencia que he ganado al invertir tanto tiempo y esfuerzo en este proyecto me será muy útil profesionalmente. Cada una de las partes del sistema de las que yo era responsable me han supuesto un desafío. Entender las tecnologías que he usado, e implementarlas exitosamente ha sido una experiencia muy enriquecedora. Ser capaz de desarrollar un sistema funcional como el que se ha desarrollado, me dará mucha seguridad y confianza en el momento de empezar futuros trabajos.

En el inicio del proyecto, se me presentaba la oportunidad de viajar a Maputo y enseñar y ayudar a los profesionales locales cómo funciona el sistema desarrollado, lo que me hubiera encantado, pero por culpa del coronavirus esta parte del trabajo no pudo ser ejecutada.

De todas formas, este proyecto puede ayudar a que en un futuro se acabe este software y sea utilizado para lo que fué diseñado: ayudar a la gente de Mozambique a poder tener una mejor vida en su ciudad.

Agradezco a Xavi, el director del proyecto, la oportunidad que nos brindó al ofrecernos este proyecto. Usar mis habilidades como desarrollador para ayudar a los que mas lo necesitan, sin ánimos de lucro, me ha dado fuerzas para poder terminar este TFG, y así cerrar mi etapa de estudiante.

9.2- Integración conocimientos

Las principales competencias trabajadas durante la realización de este proyecto han sido las siguientes:

- **CES1.1**: Desarrollar, mantener y evaluar sistemas y servicios complejos.
- **CES1.2**: Dar solución a problemas de integración en función de las estrategias, de los estándares y de las tecnologías disponibles.
- **CES1.3**: Identificar, evaluar y gestionar riesgos potenciales asociados a la construcción de software que se puedan presentar.
- **CES1.4**: Desarrollar, mantener y evaluar servicios y aplicaciones distribuidas con soporte de red.
- **CES1.5**: Especificar, diseñar y evaluar bases de datos.
- **CES1.7**: Controlar la calidad y diseño de pruebas en la producción de software.
- **CES2.1**: Definir y gestionar los requisitos de un sistema de software.

9.3- Trabajo futuro

Este trabajo ha resultado en un prototipo bastante funcional, pero haría falta algunos retoques finales para considerarlos totalmente operable.

- La tecnología USSD al final no ha sido usada para la aplicación desarrollada. Se ha facilitado a un futuro desarrollador un código modular donde modificando una clase, y utilizando la tecnología USSD desarrollada por alguien que sea capaz de usarla, la aplicación ya está es capaz de comunicarse con el Backend para dar de alta incidencias.

- Acabar los tests del sistema. Se han desarrollado varios tests del sistema, pero para completar el control de calidad del software, haría falta mejorar los tests e intentar incrementar el porcentaje de código testeado.

- Asegurar mejor el backend a nivel de seguridad. He usado la tecnología JWT para la autentificación, y he encriptado de forma bastante simple las contraseñas, pero haría falta que un experto en seguridad se asegurara que los datos de los usuarios del sistema estuvieran siendo tratados de forma segura. Estoy convencido que un hacker experimentado sería capaz de romper nuestro sistema.

- Implementar el sistema de notificaciones desde el backend. Debido al desarrollo de Eynar, esta funcionalidad ha sido cubierta, y posiblemente no sea necesario. Pero para un diseño más robusto, lo ideal sería que el backend cubriera esta funcionalidad. El código en el backend ya está implementado, haría falta encontrar el fallo de permisos con los servicios de AWS y corregir la configuración del servidor.

10- Bibliografía

Para el desarrollo de este proyecto se han consultado muchísimas webs con documentación que ha ayudado a la implementación del software. Se han registrado algunas de las webs, pero por el camino se han perdido muchos links que podrían haberse añadido a esta lista:

[1] Spring Boot Security:
https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world

[2] MySql - Save Images
https://www.digitalocean.com/community/tutorials/how-to-use-the-mysql-blob-data-type-to-store-images-with-php-on-ubuntu-18-04

[3] Images and Files with SpringBoot
https://medium.com/@rameez.s.shaikh/upload-and-retrieve-images-using-spring-boot-angular-8-mysql-18c166f7bc98

[4] USSD Africa's Talking
https://medium.com/f4life/serverless-ussd-with-africas-talking-62c97ef91fdd

[5] Software Architecture Pattern for Developing
https://www.dineshonjava.com/software-architecture-patterns-and-designs/

[6] Repository-Service Pattern
https://exceptionnotfound.net/the-repository-service-pattern-with-dependency-injection-and-asp-net-core/#:~:text=The%20Repository%2DService%20pattern%20breaks,against%20a%20single%20Model%20class.

[7] Spring Boot root project
https://spring.io/projects/spring-boot

[8] Web with a lot of Spring Boot documentation used to develop Backend
https://www.baeldung.com/

[9] MOPA
https://www.mopa.co.mz/en/

[10] Amazon Web Services
https://aws.amazon.com/