Daniel Alejandro Coll Tejeda

SOUNDLESS: PROYECTO DE CIENCIA CIUDADANA

TRABAJO DE FINAL DE GRADO

dirigido por Pedro Antonio García López

Grado de Ingeniería Informática

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2023

Resum.

En els últims anys, hi ha hagut un creixent interès per part de la comunitat científica en l'impacte del soroll generat per l'activitat humana. En 2019, l'Organització Mundial de la Salut (OMS) va publicar un estudi a Europa que confirmava i ressaltava la nova evidència sobre com el soroll generat pels mitjans de transport afecta la salut dels ciutadans europeus [1][2].

Amb l'objectiu d'abordar aquesta problemàtica que afecta els residents de Tarragona, s'ha desenvolupat i posat en producció una aplicació Android integrada al núvol, juntament amb múltiples microserveis, amb la finalitat d'augmentar la visibilitat d'aquest problema i buscar solucions.

S'ha desenvolupat un sistema de detecció d'incidències basat en les dades proporcionades pels usuaris, així com una generació de mapes que mostra la ubicació d'aquestes incidències. Aquestes funcions es van implementar en el costat del servidor. Pel que fa al client mòbil, s'ha creat una interfície i funcionalitat completament nova per a visualitzar les dades recopilades.

Després de completar aquest treball, el projecte ha finalitzat amb èxit. L'aplicació i els microserveis estan actualment en producció, sent utilitzats per usuaris reals. A més, s'ha assegurat l'escalabilitat, seguretat i eficiència de tots els aspectes del projecte.

Resumen.

En los últimos años, ha habido un creciente interés por parte de la comunidad científica en el impacto del ruido generado por la actividad humana. En 2019, la Organización Mundial de la Salud (OMS) publicó un estudio en Europa que confirmaba y resaltaba la nueva evidencia sobre cómo el ruido generado por los medios de transporte afecta la salud de los ciudadanos europeos [1][2].

Con el objetivo de abordar esta problemática que afecta a los residentes de Tarragona, se ha desarrollado y puesto en producción una aplicación Android integrada en la nube, junto con múltiples microservicios, con el fin de aumentar la visibilidad de este problema y buscar soluciones.

Se ha desarrollado un sistema de detección de incidencias basado en los datos proporcionados por los usuarios, así como una generación de mapas que muestra la ubicación de estas incidencias. Estas funciones se implementaron en el lado del servidor. En cuanto al cliente móvil, se ha creado una interfaz y funcionalidad completamente nueva para visualizar los datos recopilados.

Tras completar este trabajo, el proyecto ha finalizado con éxito. La aplicación y los microservicios están actualmente en producción, siendo utilizados por usuarios reales. Además, se ha asegurado la escalabilidad, seguridad y eficiencia de todos los aspectos del proyecto.

Abstract.

In recent years, there has been a growing interest by the scientific community in the impact of noise generated by human activity. In 2019, the World Health Organization (WHO) published a study in Europe confirming and highlighting

new evidence on how noise generated by means of transport affects the health of European citizens [1][2].

In order to address this issue affecting the residents of Tarragona, a cloud-integrated Android application has been developed and put into production, along with multiple microservices, in order to increase the visibility of this problem and seek solutions.

An incident detection system has been developed based on data provided by users, as well as a map generation showing the location of these incidents. These functions were implemented on the server side. As for the mobile client, a completely new interface and functionality was created to visualize the collected data.

After completing this work, the project has been successfully completed. The application and microservices are currently in production, being used by real users. In addition, the scalability, security and efficiency of all aspects of the project has been ensured.

Índice

1 INTRODUCCIÓN 5
1.1 MOTIVACIONES 5
1.2 DESARROLLO PREVIO 5
1.3 OBJETIVOS 6
1.4 ORGANIZACIÓN DE LA MEMORIA 7

2 TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS 8
2.1 GOOGLE CLOUD 8
2.1.1 Cloud Run 9
2.1.2 Cloud Functions 9
2.1.3 Cloud Storage 10
2.1.4 Play Console 10
2.2 FIREBASE 10
2.3 ANDROID STUDIO 11
2.3.1 SQLite 11

3 REQUISITOS DEL SISTEMA 12
3.1 REQUISITOS FUNCIONALES 12
3.2 REQUISITOS NO FUNCIONALES 13
3.3 CASOS DE USO 13

4 ARQUITECTURA Y DISEÑO 17
4.1 ARQUITECTURA DEL SISTEMA EN LA NUBE 17
4.1.1 Organización de ficheros en Cloud Storage 19
4.1.2 Formato de los datos 20
4.2 ARQUITECTURA Y DISEÑO ANDROID 21
4.2.1 Arquitectura basada en MVVM 21
4.2.2 Diseño de la interfaz 22

5 IMPLEMENTACIÓN 26
5.1 DETECCIÓN DE INCIDENCIAS 26
5.1.1 Algoritmo de z-scores 26
5.1.2 Implementación del algoritmo 28
5.1.3 Incidencias en el sueño 30
5.2 GENERACIÓN DE MAPAS 32
5.2.1 Importación de los archivos 32
5.2.2 Clustering 33
5.2.3 Generación del mapa HTML 35
5.3 APLICACIÓN ANDROID 37
5.3.1 Modelo 39
5.3.2 VistaModelo 39
5.3.3 Vista 42
5.3.4 Interfaz gráfica 43

6 JUEGO DE PRUEBAS 46
6.1 ESCALABILIDAD 46
6.2 VERIFICACIÓN DE LOS RESULTADOS OBTENIDOS 47
6.3 AUTONOMÍA 53

7 PUESTA EN PRODUCCIÓN 55
7.1 ERRORES Y SOLUCIONES 56

8 CONCLUSIONES 58
8.1 TRABAJO FUTURO 58

9 REFERENCIAS 59

Índice de tablas

TABLA 1. EQUIVALENCIAS ENTRE ESTADO DEL SUEÑO Y SU VALOR. 30
TABLA 2. EQUIVALENCIAS ENTRE NÚMERO DE INCIDENCIAS Y COLOR. 36

Índice de figuras

Figura 1. Gráfico sobre la distribución del mercado. 8
Figura 2. Distribución de cada servicio por categorias. 9
Figura 3. Caso de uso: establecer la conexión. 14
Figura 4. Caso de uso: obtener mapas de incidencias. 15
Figura 5. Arquitectura del sistema en la nube. 17
Figura 6. Organización del almacenamiento en la nube. 19
Figura 7. Patrón de diseño MVVM 22
Figura 8. Interfaz mostrando el mapa del día de ayer. 23
Figura 9. Interfaz informando sobre un mapa más actualizado. 24
Figura 10. Interfaz para seleccionar una fecha concreta. 25
Figura 11. Ejemplo de z-scores. 27
Figura 12. Ejemplo de K-means (clustering). 33
Figura 13. Mapa generado en base a datos reales 37
Figura 14. Distribución de clases dentro del patrón MVVM. 38
Figura 15. Clases y sus métodos 38
Figura 16. Vista de la interfaz gráfica y sus componentes 44
Figura 17. Número de visitas hacia el servidor. 46
Figura 18. Número de errores del servidor 46
Figura 19. Fichero con datos obtenidos por un usuario ficticio. 47
Figura 20. Nivel de dB y ritmo cardiaco del usuario de 05:00 a 06:00 48
Figura 21. Estado del sueño del usuario de 05:00 a 06:00. 48
Figura 22. Fichero resultante con las incidencias detectadas. 48
Figura 23. Mapa resultante de la ciudad de Tarragona 49
Figura 24. Mapa resultante de la ciudad de Tarragona con solo una incidencia. 49
Figura 25. Mapas resultantes de las ciudades de Tarragona, Reus y provincia de Tarragona sin incidencias. 50
Figura 26. Notificación informando de la disponibilidad de un nuevo mapa. 51
Figura 27. Interfaz sobre las diferentes fechas seleccionadas 51
Figura 28. Visualización del último mapa descargado en la memoria interna del dispositivo 52
Figura 29. Inserción de ficheros con los datos obtenidos por los usuarios la noche anterior 53
Figura 30. Inicio y finalización correcta del filtrado de datos. 53
Figura 31. Creación de los ficheros filtrados 53
Figura 32. Inicio y finalización correcta del generado de mapas. 53
Figura 33. Creación de los mapas generados. 54
Figura 34. Gráfico sobre la distribución de las incidencias detectadas. 56

Índice de código

Código 1. Código fuente del filtrado y preprocesado de los datos. 28
Código 2. Código fuente del algoritmo de z-scores. 29
Código 3. Código fuente de la detección de incidencias en el sueño. 31
Código 4. Código fuente de la descarga e importación de los datos. 32
Código 5. Código fuente del cálculo máximo de distancias. 34
Código 6. Código fuente de la generación de mapas. 35
Código 7. Código fuente de la función findLastFileInBucket. 40
Código 8. Código fuente del envío del evento CancelDownload. 41
Código 9. Código fuente de la función renderWebView. 43
Código 10. Código fuente de la interfaz gráfica de webView. 44

1 Introducción

Desde hace algunos años el ruido provocado por la actividad humana ha sido objeto de estudio de múltiples investigaciones sobre el tema. Este problema no es algo local o estatal, es un problema que lleva ocurriendo a nivel global desde hace décadas.

En 2011, la OMS advirtió sobre una nueva evidencia de la afectación del ruido producido por medios de transporte como podrían ser coches, aviones, trenes, entre otros, podía provocar graves influencias en la salud de los ciudadanos en Europa [1].

Posteriormente en 2019, la OMS publicó un estudio [2] realizado en Europa donde se habla sobre la gran problemática que se está viviendo en la actualidad con respecto el ruido y los daños que puede provocar este en la salud humana. Se habla de que niveles superiores a 45 dB durante la noche, suponiendo este, el nivel máximo permitido en horas de descanso supone más personas con insomnio, estrés, poca productividad, obesidad, ... Muchos problemas que afectan a la salud y algunos de forma irreversible.

1.1 Motivaciones

Estas fueron las motivaciones que dieron origen al proyecto Soundless, una iniciativa de ciencia ciudadana que busca medir los niveles de sonido y datos de salud de las personas. Este proyecto se ha desarrollado mediante una aplicación móvil conectada a distintos microservicios en la nube y a diversas plataformas que permiten la medición del ruido nocturno, así como la obtención de información sobre el ritmo cardíaco y el sueño mediante el uso de pulseras especializadas.

Desde las primeras etapas del proyecto, antes de su lanzamiento en la Google Play Store y la participación de usuarios reales, decidí unirme al equipo para desarrollar una de las funcionalidades clave de esta aplicación móvil junto a la nube. Mi objetivo era ayudar a visibilizar un problema que afecta a cientos de personas en la ciudad de Tarragona.

Por esta razón, he diseñado, desarrollado e implementado un proceso de detección de incidencias procesado en la nube, a partir de los datos de sonido, ritmo cardiaco y sueño obtenido mediante los usuarios. A su vez, he desarrollado una interfaz y una funcionalidad completamente nueva para poder visualizar estos mismos datos en la aplicación Android.

1.2 Desarrollo previo

Al momento de unirme al proyecto de Soundless, ya existía un proyecto avanzado donde estaba desarrollada una aplicación móvil la cual era capaz de detectar sonido, mostrarlo y almacenarlo en la nube. A su vez, esta, ya era capaz de realizar la conexión con los servidores de las pulseras de actividad para descargar los datos de salud y subirlos al almacenamiento destinado.

En cuanto a los servicios en la nube ya existentes, en el momento de incorporación existía una arquitectura inicial que contaba con un proceso de autenticación mediante la nube, una máquina virtual que procesa los datos recibidos a través de pub/sub, y un almacenamiento predefinido en la nube para almacenar todos los datos recibidos. En cuanto a la parte móvil, existía una interfaz definida junto a algunas funcionalidades como la grabación, conexión con las pulseras inteligentes y subida de estos datos a la nube.

Sin embargo, durante mi contribución a este proyecto, pude agregar procesos adicionales, como la generación de mapas, detección de incidencias y programación de eventos en el tiempo. Estos procesos se definirán y explicarán en este trabajo.

Al finalizar el proyecto habré estudiado y utilizado múltiples tecnologías para procesado y almacenamiento de datos en la nube, como serían Cloud Run, Functions, Scheduler o Storage. Todo ello conectado mediante Firebase y Play Console a un cliente Android, con la capacidad de recoger y mostrar los datos recogidos. Todo ello contando con las librerías específicas de cada proceso como serian Bokeh, Pandas, Geopandas o Matplot.

1.3 Objetivos

Mi proyecto tiene como finalidad desarrollar mis competencias informáticas a través de una serie de objetivos que suponen un reto y una oportunidad de aprendizaje. Estos objetivos abarcan diferentes ámbitos en los que tengo poca o ninguna experiencia previa.

Los objetivos son los siguientes:

- Servidor en la nube:
    - Generar una serie de mapas de ruido por zonas geográficas a partir de la información de los usuarios extraída por la aplicación.
    - Análisis de estos datos de forma precisa para encontrar las correlaciones e incidencias entre sonido y datos de salud.
    - Implementar una serie de microservicios que me permitan gestionar estos datos y automatizar el proceso.
- Aplicación móvil:
    - Desarrollar un protocolo para la conexión, descarga y gestión de los archivos almacenados en el servidor.
    - Diseñar y crear una nueva funcionalidad capaz de mostrar los datos analizados, siguiente el patrón de diseño móvil llamado MVVM[1].
    - Ofrecer una experiencia fluida y sencilla al usuario mediante una interfaz clara y precisa.

Además de estos objetivos técnicos, uno de mis grandes objetivos es aprender y obtener nuevas habilidades en áreas que nunca había explorado antes o lo había hecho en poca medida, como son las aplicaciones móviles, el sistema en la nube, la integración y distribución continua de un proyecto profesional, y el manejo de una aplicación en producción con usuarios reales que prueben la aplicación a diario.

1 Model-View-ViewModel: se hace referencia al patrón de diseño en el punto 4.2.1 de este documento.

1.4 Organización de la memoria

Este trabajo presenta el proceso de creación de un sistema que consta de una aplicación móvil y una plataforma en la nube. La memoria se estructura en 8 secciones que abordan cada una de las fases del desarrollo.

En la Sección 2 se describen las tecnologías y herramientas empleadas, tanto para la programación como para el uso de APIs[2] y servicios de las plataformas. En la Sección 3 se definen los requisitos que debe cumplir el sistema para satisfacer las necesidades del usuario. En la Sección 4 se explica la arquitectura y el diseño de la aplicación móvil y de la plataforma en la nube, así como las decisiones tomadas al respecto. En la Sección 5 se detalla la implementación y el funcionamiento de los componentes desarrollados, mostrando ejemplos de código y capturas de pantalla. En la Sección 6 se realiza un conjunto de pruebas para verificar que el sistema funciona correctamente y cumple con los requisitos establecidos. En la Sección 7 se narra el proceso de puesta en producción del sistema, comentando los desafíos e inconvenientes encontrados y cómo se han resuelto. Finalmente, en la Sección 8 se presentan las conclusiones y los resultados obtenidos en este trabajo, así como las posibles mejoras y ampliaciones futuras.

2 Application Programming Interface: diferentes herramientas que proporcionan los desarrolladores para permitir la comunicación con su sistema.

2 Tecnologías y herramientas utilizadas

En este proyecto, se han utilizado tecnologías y herramientas innovadoras con el fin de diseñar y desarrollar un producto completo en dos ámbitos clave de la informática: servicios en la nube y desarrollo de una aplicación móvil para Android. De esta manera, se ha trabajado en la implementación de soluciones que permitan una experiencia de usuario óptima y un despliegue eficiente del proyecto.

Durante el desarrollo del proyecto fue importante seleccionar un buen proveedor de servicios en la nube, ya que este iba a ser el que nos proveyera de las herramientas y servicios necesarios para la creación de este proyecto. En este caso, se optó por la plataforma de Google Cloud, una de las lideres del sector y en específico, la que ofrece un mayor número de ventajas para el desarrollo móvil.

En este caso, la plataforma de Google ofrece servicios básicos como el almacenamiento remoto, la gestión de máquinas virtuales, la ejecución de procesos de alto nivel, una gran escalabilidad y disponibilidad, entre muchos otros. Pero a su vez, ofrece servicios únicos como serian Firebase y Big Query, orientados a este tipo de proyectos de ingieneria.

Estos dos servicios se explicarán con detalle a continuación, pero han sido una pieza clave para mantener un desarrollo rápido y eficiente al a vez que seguro y automatizado. Nos ha permitido obtener finalmente una gran aplicación móvil capaz de cumplir con los requisitos esperados.

2.1 Google Cloud

Como veníamos diciendo anteriormente, Google Cloud es actualmente, uno de los principales proveedores de servicios en la nube. Un mercado que genera más de 200 mil millones de dólares al año [3] y que representa una fuente importante de ingresos para las empresas.

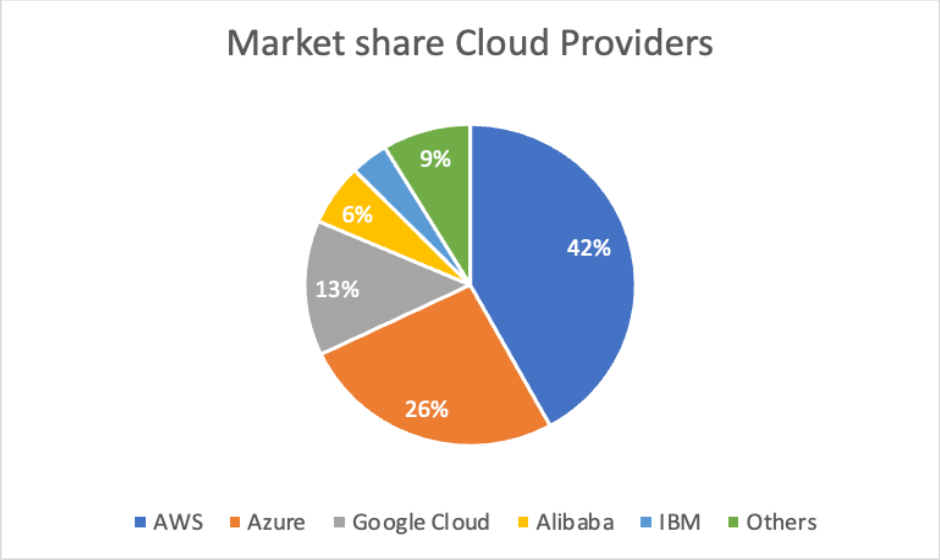


Figura 1. Gráfico sobre la distribución del mercado.

Actualmente, Google Cloud ocupa el tercer lugar en cuota de mercado [4], como se muestra en la *Figura 1*, solo por detrás de Amazon con su plataforma llamada AWS y Microsoft con su correspondiente Azure.

Google Cloud como se menciona anteriormente, en la actualidad cuenta con una gran variedad de servicios adaptados a diferentes necesidades y funciones según la necesidad de cada proyecto. En este caso, nos hemos centrado y utilizado los servicios que se mencionan en este punto.

Figura 2. Distribución de cada servicio por categorias.

Antes de entrar en la descripción y utilidad de cada uno de los servicios, se muestra en esta imagen, *Figura 2*, la distribución de cada uno de los servicios que se mencionan en tres categorías: producción, almacenamiento y computación. Representando de una forma más gráfica las distintas utilidades para estos mismos.

2.1.1 Cloud Run

Cloud Run es uno de los servicios que ofrece Google queA permite la ejecución de código en cualquier lenguaje de programación de forma escalable y contenerizada. Este servicio utiliza contenedores creados con Docker para poder ejecutar el código deseado.

Cloud Run tiene la ventaja de eliminar las restricciones que puedan tener otros servicios en cuanto a lenguajes de programación, librerías predefinidas o cualquier otra limitación. Además, se encarga de gestionar automáticamente el escalado para adaptarse al tráfico y los recursos necesarios.

Se hizo la elección de este proceso ya que era un servicio ideal para la generación de mapas ya que nos permitía tener todas las librerías y programas necesarios en un solo contenedor ejecutable.

2.1.2 Cloud Functions

Cloud Functions es un servicio diseñado para ejecutar rápidamente pequeñas porciones de código escritas en ciertos lenguajes de programación, en respuesta a eventos

específicos. Es un servicio altamente escalable que se enfoca en la ejecución rápida y eficiente de código basado en eventos.

A diferencia de Cloud Run, que está diseñado para ejecutar aplicaciones completas en contenedores, ofreciendo más flexibilidad en términos de lenguajes de programación y librerías, Cloud Functions se centra en tareas más simples y específicas.

En cuanto a este servicio, a diferencia de Cloud Run, nos ha permitido ejecutar la detección de incidencias mediante el algoritmo Z-Scores que veremos a continuación. Se escogió este servicio, ya que es ligero, rápido y no necesita de la complejidad de un contenedor para la ejecución de un programa sencillo como este.

2.1.3 Cloud Storage

Cloud Storage es uno de los servicios de Google en su nube que permite el almacenamiento de archivos o cualquier tipo de objeto en distintos contenedores llamados buckets. Este servicio permite el acceso a esos datos en cualquier momento y en cualquier ubicación geográfica.

Una de sus ventajas es la posibilidad de transferir todos los datos almacenados a otra solución de Google Cloud llamada BigQuery, que nos permite hacer un análisis de los datos obtenidos. Además, nos ofrece una solución de almacenamiento de forma económica y sencilla, integrada con el resto de los servicios que ofrece Google.

2.1.4 Play Console

La Consola de Google Play es una plataforma esencial para que los desarrolladores puedan publicar y distribuir sus aplicaciones en la tienda de Google Play. A través de esta herramienta, los desarrolladores tienen la capacidad de publicar, actualizar y gestionar la información de sus aplicaciones, además de monitorear las estadísticas de descarga y el rendimiento.

2.2 Firebase

Firebase es uno de los servicios estrella de la plataforma de Google en la nube. Este servicio ofrece una variedad de funciones e interfaces, todo centrado en poder trabajar sobre aplicaciones móviles.

Este servicio incluye autentificación de usuarios, notificaciones a distancia a los dispositivos móviles, ejecución de pruebas parciales, y más servicios centrados aplicaciones Android y iOS. Además, cuenta con una integración completa con otros servicios de Google Cloud, como Cloud Functions y Cloud Storage, lo que permite una gestión más eficiente y escalable de las aplicaciones.

Aunque en este caso los realmente importantes son su sistema de autentificación y validación de usuarios dentro de la nube de Google.

2.3 Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android. Basado en el potente editor de código y las herramientas de desarrollador de IntelliJ IDEA, Android Studio ofrece aún más funciones que mejoran tu productividad al crear aplicaciones Android, tales como:

- Un sistema de compilación flexible basado en Gradle.
- Un emulador de Android que te permite probar tu aplicación en una variedad de dispositivos Android.
- Herramientas de diseño que te permiten crear interfaces de usuario dinámicas con Jetpack Compose.
- Herramientas de depuración y análisis que te ayudan a optimizar el rendimiento y la calidad de tu aplicación.
- Una integración con los servicios de Google Cloud, como Cloud Functions, Cloud Storage y BigQuery.

Android Studio te facilita el desarrollo de aplicaciones Android al proporcionarte todo lo que necesitas en un solo lugar.

2.3.1 SQLite

SQLite es un sistema de gestión de bases de datos ligero y embebido que se integra con el sistema operativo Android. SQLite permite crear y manipular bases de datos locales en las aplicaciones móviles, usando una interfaz SQL limpia y con un bajo consumo de memoria y una buena velocidad.

Este es el motor de base de datos más usado en el mundo, utilizado actualmente por cientos de aplicaciones móviles y de escritorio, en este caso, este gestor esta incorporado dentro del propio sistema operativo Android, permitiendo una integración mucho mayor.

Este permite tener un almacenamiento de datos estructurados o semi-estructurados dentro de base de datos eficiente y rápida, permitiendo almacenar toda la información necesaria para este trabajo.

3 Requisitos del sistema

Los requisitos de un trabajo son una parte fundamental en el desarrollo de cualquier proyecto. Estos describen las condiciones o capacidades que debe cumplir el producto o sistema que se va a desarrollar o modificar. Los requisitos definen el alcance del proyecto y establecen los criterios de aceptación y validación. Los requisitos se clasifican en funcionales y no funcionales según el tipo de información que proporcionan.

3.1 Requisitos funcionales

En este apartado se describen los requisitos funcionales del proyecto, es decir, las funciones que el sistema debe realizar para satisfacer las necesidades de los usuarios. El sistema se compone de dos partes principales: un sistema en la nube y una aplicación Android.

En el sistema en la nube se describen de las siguientes tareas:

- Obtener los datos de los usuarios de Google Cloud, donde se almacenan los registros de sonido, ritmo cardiaco y sueño obtenidos por los dispositivos y pulseras inteligentes.
- A partir de estos datos, aplicar un algoritmo matemático para detectar las incidencias que puedan afectar al sonido, ritmo cardiaco o la calidad del sueño de los usuarios.
- Generar una representación en forma de mapa de calor de Tarragona, Reus y la provincia de Tarragona con los puntos con un mayor número de incidencias. Estos mapas serán generados y almacenados en la nube de forma diaria.

En cuanto a la aplicación Android se describen las siguientes tareas:

- Realizar un proceso para la autentificación y conexión con los servidores de Google Cloud y Firebase.
- Obtener el mapa de calor más reciente generado anteriormente por el sistema en la nube, en caso de no existir una versión actualizada en la nube utilizar el mapa almacenado en local si existe.
- Ofrecer una interfaz intuitiva y fácil de usar, que permita a los usuarios visualizar los mapas de incidencias de forma gráfica y poder consultar información sobre el mismo.
- Crear un algoritmo de gestión de mapas antiguos, nuevos y actuales para así optimizar el uso del espacio de almacenamiento del dispositivo. Conservando únicamente la versión más reciente de cada mapa y actualizando esta versión en caso de existir una más reciente en la nube.

3.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que describen las características de calidad del sistema, como la seguridad, la fiabilidad, el rendimiento o la mantenibilidad. Estos requisitos son importantes para asegurar el correcto funcionamiento del sistema y la satisfacción de los usuarios.

Se describen los siguientes requisitos no funcionales para el sistema en la nube descrito anteriormente:

- Disponibilidad y escalabilidad: el sistema debe estar siempre operativo y ser capaz de adaptarse al aumento o disminución de la demanda de los usuarios sin perder rendimiento ni calidad.
- Eficiencia y precisión: el sistema debe aplicar el algoritmo de detección de incidencias de forma rápida y precisa, minimizando los falsos positivos y los falsos negativos.
- Realismo y fidelidad: el sistema debe generar unos mapas de calor que reflejen fielmente la realidad de los datos obtenidos, sin distorsiones ni errores.
- Privacidad y confidencialidad: el sistema debe garantizar que los datos obtenidos por la detección de incidencias no sean accesibles ni manipulables por terceros no autorizados, respetando la legislación vigente en materia de protección de datos.

En cuanto a la aplicación Android los requisitos no funcionales que se describen son los siguientes:

- Sencillez e intuitivo: la aplicación debe tener una interfaz sencilla e intuitiva que permita al usuario acceder a las funcionalidades de forma fácil y rápida, sin necesidad de instrucciones ni ayuda externa.
- Usabilidad y navegación: la aplicación debe ofrecer una buena experiencia de usuario, permitiendo una navegación fluida y cómoda por los mapas de calor, con opciones para acercar, alejar o mover la vista, además de poder obtener información adicional sobre cada punto de calor.
- Adecuación: la aplicación debe mostrar los mapas de calor de forma adecuada al tamaño y resolución de la pantalla del dispositivo, sin perder calidad ni detalle.

3.3 Casos de uso

Este apartado describe los casos de uso del proyecto, que son las secuencias de acciones que el sistema realiza para obtener un resultado de valor para los actores. En este proyecto hay dos casos de uso principales.

El primer caso de uso consiste en establecer una conexión entre los servicios en la nube de Google y la aplicación móvil instalada en el dispositivo del usuario. Esta conexión permitirá a la aplicación acceder a los servicios en la nube y obtener información relevante para su funcionamiento.

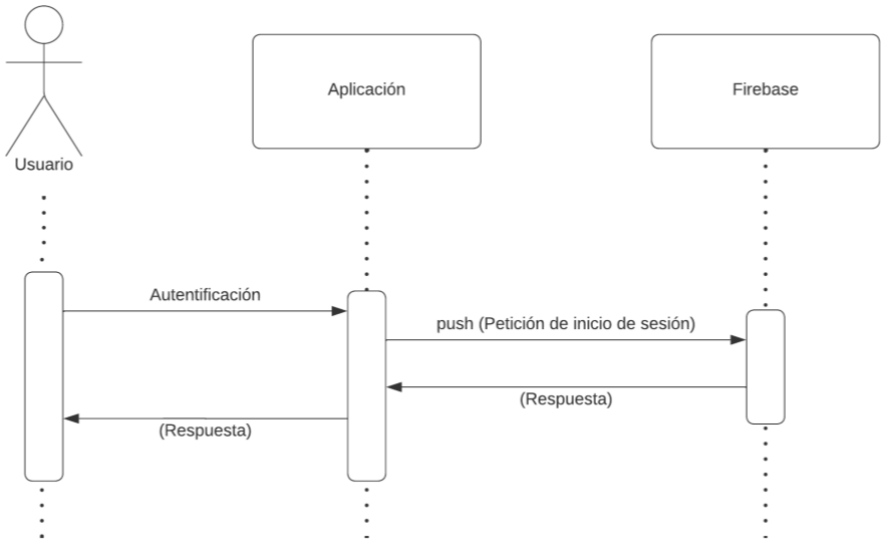


Figura 3. Caso de uso: establecer la conexión.

En la *Figura 3* se muestra el proceso por el cual se establece esa conexión entre la aplicación móvil y los servidores de Google Cloud a través de Firebase, todo ello a partir de las acciones tomadas por el usuario.

Este proceso va desde una solicitud de autentificación que se envía desde el Usuario a los servidores remotos pasando a través de la aplicación desarrollada. Una vez se genera la respuesta es enviada a través de estos elementos hasta llegar al usuario final.

Este segundo caso de uso implica la solicitud y obtención de un mapa de calor desde el almacenamiento en la nube. El usuario podrá solicitar un mapa de calor a través de la aplicación móvil, y la información se obtendrá del almacenamiento en la nube de Google. Este mapa de puntos geolocalizados se utilizará para visualizar datos relevantes para el usuario, como la ubicación y la intensidad de la actividad en un área determinada.

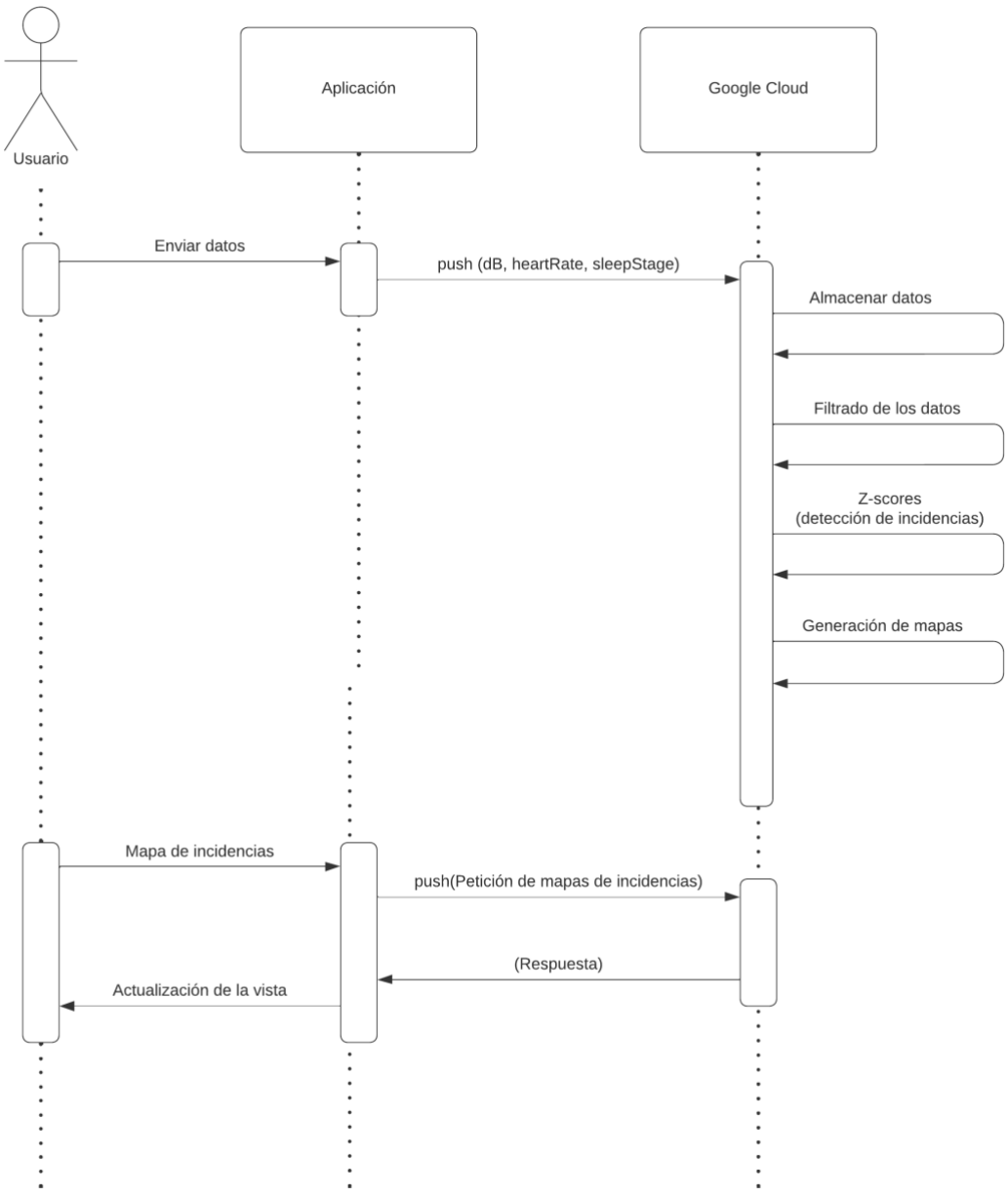


Figura 4. Caso de uso: obtener mapas de incidencias.

Tal y como se muestra en la *Figura 4*, existe una serie de procesos iniciales ejecutados dentro de Google Cloud que son los encargados de realizar la obtención de datos, detección de incidencias a partir de estos y posteriormente la generación de mapas de estas mismas incidencias.

Una vez son realizados todos estos procesos previos, cuando el usuario solicita uno de estos mapas de incidencias generados, envía la petición hacia el servidor, una vez más pasando por la aplicación para dispositivos móviles. El servidor en este caso devuelve el mapa solicitado y es devuelto al usuario en forma de vista donde se muestra el contenido de este.

4 Arquitectura y diseño

La arquitectura y el diseño son aspectos fundamentales para el desarrollo de un proyecto de ingeniería. En este apartado se describen las decisiones y los criterios que se han seguido para definir la arquitectura y el diseño de la plataforma en la nube y de la aplicación móvil que forman parte del sistema propuesto.

4.1 Arquitectura del sistema en la nube

La plataforma en la nube se compone de diferentes elementos, como podrían ser almacenamiento, activadores por eventos y pequeñas partes de código escrito en Python para poder realizar la generación de mapas de calor por incidencias en las diferentes zonas donde se ha dado el estudio.

Por ello, la arquitectura es un gran punto, ya que nos permite identificar y simplificar el proceso de entender cómo funciona el complejo sistema autónomo. Para poder entender todo este sistema se muestra la siguiente vista general de la arquitectura y todos sus componentes:

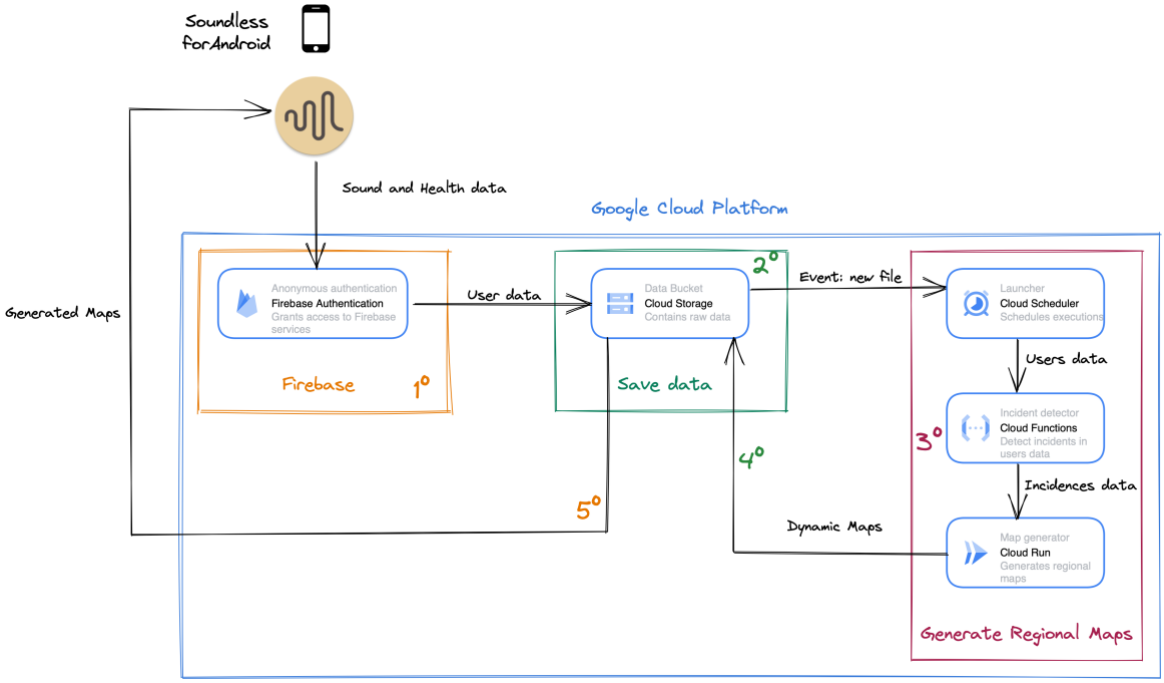


Figura 5. Arquitectura del sistema en la nube.

Tal y como vemos en la *Figura 5*, se hace uso de un gran número de servicios diferentes, como el almacenamiento remoto, funciones o autentificación de los usuarios, entre otros. Todo ello con el objetivo de recolección de datos, detección de incidencias y generación de mapas de incidencias de forma automatizada.

Para poder simplificar el proceso y el entendimiento de todos los servicios utilizados se ha hecho uso de unos pequeños números al lado de cada uno de los bloques, con tal de poder mostrar y explicar el ciclo de vida que ocurre a lo largo del sistema.

En el punto número 1 podemos observar un gran bloque llamado Firebase Authentication. Este es el componente encargado de realizar la autentificación del usuario dentro del sistema, permitiendo poder intercambiar información con el almacenamiento del proyecto.

Tras realizar esta autentificación el dispositivo de forma manual, realizado por el usuario, se hace la transferencia de todos los datos recogidos hacia la nube. Los datos en cuestión son los datos de sonido y salud capturados por el dispositivo móvil y la pulsera inteligente respectivamente.

En el segundo gran bloque observamos el servicio Cloud Storage, este es el que nos permite el almacenamiento de los datos recibidos por el usuario y posteriormente, de los mapas generados Esta es la pieza central del sistema ya que es la que recibe todas las peticiones de carga y descarga de archivos de forma constante.

Una vez, se reciben nuevos datos por parte del dispositivo móvil, se activa un nuevo evento capturado gracias, Cloud Scheduling. Este es el componente encargado detectar que se ha hecho la inserción de un nuevo archivo en una ubicación determinada dentro del almacenamiento remoto. Este se configura para que, una vez detectada la inserción, active el siguiente servicio que será el encargado de realizar la detección de incidencias. Este elemento en concreto nos permite poder automatizar de forma inteligente el proceso de procesado de los datos sin necesidad de realizarlo de forma manual.

Tras esta activación, empieza la ejecución de las llamadas Cloud Functions. Como se menciona en el punto 2.1.2, este servicio permite la ejecución de pequeñas partes de código que realizan tareas sencillas y rápidas. Este componente en específico es el encargado de realizar la detección de incidencias a partir de los datos entrantes detectados por el Scheduling. Para la realización de esta tarea se utiliza un algoritmo matemático llamado z-scores, algoritmo que explicaremos en el punto 5.1.1.

Una vez ha finalizado la ejecución de las Cloud Function, obtenemos una serie de archivos cuyo contenido son las incidencias detectadas en el sonido, ritmo cardiaco y sueño. Al obtener estos archivos, automáticamente, se realiza la activación del último de los servicios, Cloud Run.

Este elemento ya mencionado de forma sencilla en el punto 2.1.1 de este documento, es el encargado de la ejecución del generador de mapas, un programa complejo y encapsulado en forma de contenedor Docker [5]. Este proceso, recibe como entrada los datos obtenidos de la detección de incidencias mencionados anteriormente.

A partir de estos mismos datos se realizan los cálculos y generaciones correspondientes para finalmente poder almacenar nuevamente en el Storage, los mapas que serán descargados y visualizados por la aplicación móvil.

Pasando finalmente, al punto 5 de la figura, donde se muestra el ultimo recorrido de estos datos, que es la descarga de los mapas obtenidos por parte de la aplicación móvil. Siendo este paso el que marca el final de un proceso automatizado que se ejecuta diariamente sin necesidad de ningún tipo de interacción manual.

Una vez explicados los elementos y sus funciones dentro del sistema orquestado en la nube es importante mencionar y explicar la organización de ficheros y el formato realizado en estos para poder procesar y gestionar los datos contenidos en estos dentro de los diferentes componentes del proyecto.

4.1.1 Organización de ficheros en Cloud Storage

La organización de ficheros en distintos almacenamientos nos permite dar unas características y propiedades al contenedor que los guarda de forma individualizada y personalizada en función de las necesidades que se necesiten en cada uno de ellos.

Para poder entender esta gestión se necesita conocer el termino bucket, estos son los contenedores básicos donde se permite el almacenamiento de datos, estos nos permiten tener una división por secciones y usos de los ficheros mucho más eficiente y entendedora.

Cada uno de estos buckets nos ha permitido hacer la distribución de los datos capturados por el usuario, los mapas generados, utilidades necesarias para la generación de mapas entre otros archivos necesarios para todo el proceso. Esta división a su vez nos ha permitido hacer una mejor gestión del acceso y privacidad que tienen cada uno de estos contenedores, evitando que los usuarios puedan acceder a información sensible o archivos los cuales no deberían de tener acceso.

☐	soundless-data	8 ago 2022 12:08:31	Region	europe-west1	Standard	12 ene 2023 09:48:20	No público
☐	soundless-filtered-data	19 sept 2022 09:21:37	Region	europe-west1	Standard	19 sept 2022 09:21:37	No público
☐	soundless-maps	12 sept 2022 14:16:47	Region	europe-west1	Standard	12 sept 2022 14:16:47	No público
☐	soundless-reports	13 oct 2022 11:56:52	Region	europe-west1	Standard	13 oct 2022 11:56:52	No público
☐	soundless-utils	2 nov 2022 09:24:00	Region	europe-west1	Standard	12 ene 2023 09:48:36	No público

Figura 6. Organización del almacenamiento en la nube.

La siguiente imagen *Figura 6* muestra la arquitectura de almacenamiento de la aplicación cloud desarrollada para el análisis de datos acústicos. Se han utilizado cinco buckets de Google Cloud Storage, cada uno con una función y un tipo de archivos específicos. Los buckets son los siguientes:

- soundless_data: Este bucket contiene los archivos de datos en bruto de cada usuario, organizados por fecha y ciudad de origen. Se utiliza un sistema de control de versiones para garantizar la integridad y la recuperación de los datos en caso de pérdida o corrupción.
- soundless-filtered-data: Este almacen guarda los archivos de datos temporales que se generan durante el proceso de detección de incidencias y limpieza de datos. Actúa como un intermediario entre las funciones cloud (Cloud Functions) y los servicios cloud (Cloud Run) que realizan finalmente la generación de los mapas de calor.
- soundless-maps: Una vez ya generados los ficheros son almacenados en este apartado independiente y le permite a la aplicación móvil conocer la ubicación exacta del mapa que se desea, sirviendo como única puerta de acceso a la aplicación móvil a los datos sobre incidencias detectadas.
- soundless-reports: Este apartado dentro del almacenamiento del proyecto, se utiliza para guardar los informes y reportes que se elaboran a partir de los resultados del

análisis acústico. Estos documentos presentan las conclusiones y las recomendaciones sobre la gestión del ruido urbano. Este aspecto no se aborda en este trabajo de fin de grado.

- soundless-utils: Este bucket contiene los archivos auxiliares que se necesitan para la generación de mapas, como, por ejemplo: información demográfica y geográfica de las ciudades objetivo, plantillas y formatos para los mapas, etc. Este bucket permite tener una mejor organización y acceso a estos archivos que se usan frecuentemente.

Una vez obtenida una visión sobre como se han distribuido los diferentes archivos en el almacenamiento en la nube de Google, es hora de hablar del formato de los archivos utilizados, y el reto que ha supuesto diseñar un formato de datos adecuado para que los distintos programas puedan intercambiar y procesar la información de forma eficiente y coherente. A continuación, se describe el formato de datos utilizado y los desafíos que se han afrontado para su creación.

4.1.2 Formato de los datos

Para este trabajo se exploraron diferentes opciones y posibilidades para facilitar el intercambio de información entre los distintos programas involucrados. El formato de los datos era un aspecto clave, ya que determinaba cómo iban a ser interpretados los datos por los distintos componentes del sistema.

Tras realizar varios análisis sobre las distintas alternativas de formatos de archivo, las variables que debían o no incluirse en estos datos, y la forma de presentar estos datos en dichas variables, se establecieron las siguientes condiciones.

El formato de los datos elegido para este proyecto ha sido el de ficheros CSV (*comma-separated values*), que consisten en archivos de texto plano que contienen una lista de datos separados por comas. Este formato tiene la ventaja de ser fácil de leer y escribir tanto por humanos como por máquinas, y de ser compatible con la mayoría de las herramientas de análisis de datos.

Los ficheros CSV se han dividido en dos tipos: los ficheros enviados por el usuario y los ficheros con las incidencias detectadas. Los ficheros enviados por el usuario contienen los datos recogidos por la aplicación móvil durante el sueño del usuario, como el identificador anónimo y único de cada grabación (*uuid*), las coordenadas geográficas (*lng* y *lat*), la fecha y hora (*timestamp*), el nivel de ruido ambiental en decibelios (*dB*), la frecuencia cardíaca en pulsaciones por minuto (*heartRate*), la fase del sueño (*sleepStage*) y la ubicación aproximada de la grabación (*geometry*).

Los ficheros con las incidencias detectadas contienen los resultados del análisis de los datos enviados por el usuario, como el identificador anónimo y único de cada grabación (*uuid*), la fecha y hora exactas de la incidencia (*timestamp*), y los indicadores de incidencia en el nivel de ruido ambiental (*incidenciadB*), la frecuencia cardíaca (*incidenciaHeartdB*) y la fase del sueño (*incidenciaSleepdB*). Estos indicadores son valores binarios, con valores como 0 o 1, que indican si se ha detectado o no una anomalía en los datos correspondientes. Además, estos ficheros también incluyen la ubicación aproximada de la grabación (*geometry*) para poder conocer el lugar aproximado donde se ha producido la incidencia detectada.

El formato de los datos ha sido diseñado con el objetivo de preservar la privacidad y el anonimato de los usuarios, ya que no se almacena ningún dato personal o identificable que pueda vincular una grabación con una persona real. El identificador anónimo y único de cada grabación (*uuid*) se genera de forma aleatoria y no tiene ninguna relación con el dispositivo o la cuenta del usuario. Las coordenadas geográficas se utilizan únicamente para obtener la ubicación aproximada de la grabación, que se expresa en términos generales como el barrio o ciudad donde se encuentra el usuario, sin revelar la dirección exacta. La fecha y hora se almacenan en formato UTC (*Coordinated Universal Time*), sin tener en cuenta la zona horaria del usuario. Los datos de nivel de ruido ambiental, frecuencia cardíaca y fase del sueño se almacenan tal y como se recogen por la aplicación móvil, sin aplicar ningún tipo de transformación o normalización. Asegurando así en todo momento una mayor privacidad y seguridad de los datos obtenidos.

El formato de los datos ha supuesto un reto para el desarrollo del proyecto, ya que ha requerido una coordinación entre los distintos programas utilizados para recoger, almacenar, analizar y visualizar los datos. Para ello se han escogido estos dos formatos, proporcionando la información necesaria e imprescindible para poder llevar a cabo los distintos procesos, de una forma eficiente y simple para que el proyecto pueda tener una continuidad en el futuro.

4.2 Arquitectura y diseño Android

En cuanto a la arquitectura de la aplicación móvil, se ha buscado seguir siempre un patrón de diseño de acorde a lo que se espera en unos estándares de calidad de una aplicación móvil. También se ha intentado en todo momento realizar un diseño de interfaz cómodo y sencillo para que cualquier usuario pueda navegar cómodamente por los menús.

4.2.1 Arquitectura basada en MVVM

MVVM es un patrón de diseño que separa la lógica de negocio de la presentación, facilitando así el mantenimiento y la reutilización del código. MVVM significa Model-View-ViewModel, y consiste en tres componentes principales:

- El **Modelo** representa los datos y la lógica de negocio de la aplicación. Es el encargado de acceder a las fuentes de datos, como bases de datos o servicios web, y proveer los datos a los demás componentes.

- La **Vista** es la capa de presentación que muestra los datos al usuario. Es la responsable de definir la apariencia y el comportamiento de la interfaz gráfica, mediante elementos visuales como botones, listas o gráficos.

- El **ViewModel** es el intermediario entre el Modelo y la Vista. Es el encargado de exponer los datos del Modelo a la Vista de forma que sean fácilmente consumibles por esta. También gestiona los eventos y las acciones del usuario, y actualiza el Modelo según sea necesario. Es decir, es el encargado de procesar los datos obtenidos por el modelo y entregarlos a la vista para visualizarlos al usuario.

Figura 7. Patrón de diseño MVVM.

En la *Figura 7* nos encontramos lo comentado anteriormente, la separación en tres componentes del patrón de diseño, y a su vez el intercambio de datos que se realiza entre los distintos componentes de este sistema.

La principal ventaja de usar MVVM es que se logra una separación clara entre las responsabilidades de cada componente, lo que facilita el desarrollo, la depuración y las pruebas de la aplicación. Además, se favorece el uso de técnicas como eventos o el enlace de datos, que permite sincronizar automáticamente los cambios entre el Modelo y la Vista sin necesidad de escribir código adicional.

La estructura principal de la aplicación es la siguiente, en primera instancia la aplicación se comunica con la API de Google Cloud para acceder al almacenamiento en la nube y descargar los archivos solicitados por el usuario, guardándolos en el Modelo. Para transmitir los datos entre la Vista y el ViewModel se utilizan eventos y LiveData.

Los eventos son objetos que representan una acción o un cambio de estado que ocurre en la aplicación y que pueden ser observados por otros componentes. LiveData en cambio, es una clase que almacena datos observables y que respeta el ciclo de vida de los componentes de la Vista, evitando así fugas de memoria o actualizaciones innecesarias.

Este proyecto se ha realizado siguiendo las buenas prácticas recomendadas por Android y el patrón MVVM, con el objetivo de crear un código limpio, modular y mantenible.

4.2.2 Diseño de la interfaz

El diseño de la interfaz se ha basado en el estilo principal de la aplicación, que se caracteriza por el uso de colores anaranjados, una cantidad reducida de botones y un aspecto minimalista, pero sin descuidar la presentación de la información necesaria.

La interfaz consta de tres componentes principales: un webView que carga el mapa HTML correspondiente, un botón que permite alternar entre los mapas disponibles (ayer, hace una semana y hace un mes) y un desplegable que notifica al usuario cuando hay nuevos mapas disponibles para descargar. El objetivo de esta interfaz es facilitar al usuario la consulta de los mapas de forma rápida y sencilla, sin necesidad de tener conocimientos previos ni experiencia en el uso de la aplicación.

A continuación, se muestran dos imágenes de la interfaz: una con el webView mostrando el mapa de ayer *Figura 8* y otra con el desplegable abierto informando de que hay nuevos mapas listos para descargar *Figura 9*.

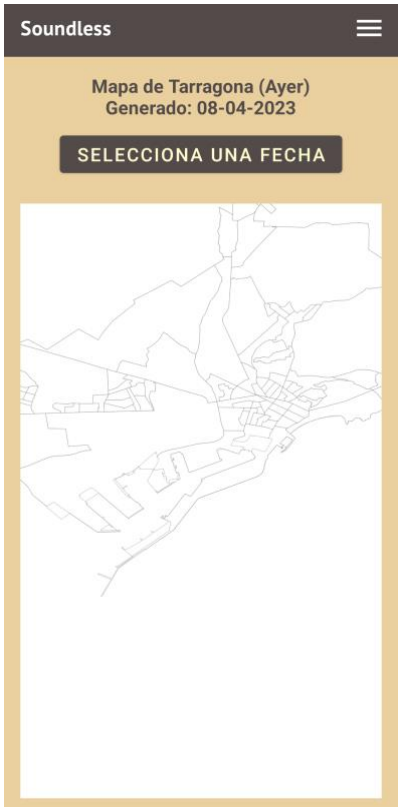


Figura 8. Interfaz mostrando el mapa del día de ayer.

Figura 9. Interfaz informando sobre un mapa más actualizado.

Finalmente, en esta última *Figura 10*, se muestra el desplegable seleccionable que se muestra cuando el usuario decide cambiar los datos que se muestran en el mapa. Aquí se muestran datos sobre el mapa generado con los datos de la noche anterior, de lo acumulado en los últimos 7 días y lo acumulado en los últimos 30 días.

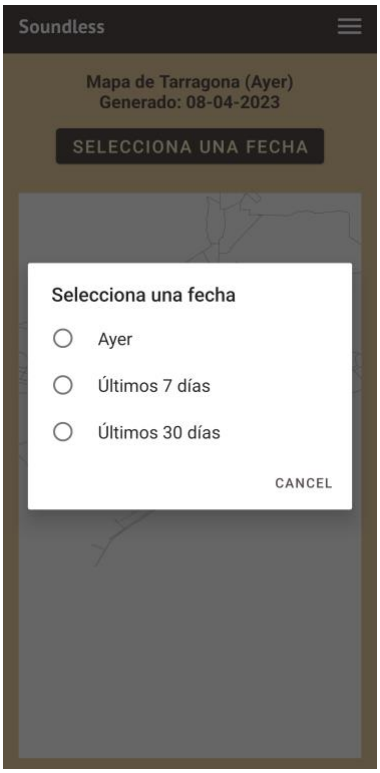


Figura 10. Interfaz para seleccionar una fecha concreta.

5 Implementación

La implementación de este proyecto se estructura en 3 grandes apartados principales, en primer lugar, la detección de incidencias en el sonido y datos de salud de los usuarios, en segundo lugar, la generación de mapas a partir de estos datos y finalmente la integración de estos dos sistemas con la aplicación Android y la visualización y gestión de los mapas creados.

5.1 Detección de incidencias

Para empezar, una incidencia se entiende como la subida repentina y de forma anómala de una variable, es decir, si en una grabación de 2 horas, con una media de 30 dB, existe un momento de la noche donde este valor aumenta en pocos segundos hasta los 50 dB, se entiende como una incidencia y una infracción de la normativa actual de ruido nocturno.

El objetivo de este trabajo era desarrollar un sistema capaz de detectar incidencias en los datos de sonido y salud recogidos por una aplicación móvil. Para ello, se exploraron diferentes alternativas que permitieran identificar valores anómalos o fuera de lo normal en las grabaciones de sonido y los parámetros fisiológicos de los usuarios. Las alternativas consideradas fueron las siguientes:

- Usar un modelo de inteligencia artificial, como ARIMA+ de Google [6], implicaría una mayor capacidad de adaptación y autonomía del sistema ante posibles cambios en los datos, lo que mejoraría su rendimiento. Sin embargo, esta opción requeriría un entrenamiento previo de la red neuronal, lo cual supondría un alto costo económico y temporal, además de una preparación exhaustiva de los datos. Debido a estas limitaciones, no se pudo implementar esta opción.

- Aplicar un algoritmo simple que considerara una incidencia como un valor que superara la media más la desviación estándar de cada grabación. Esta opción sería económica y rápida en términos de recursos y tiempo, pero presentaría problemas de robustez y fiabilidad en los resultados obtenidos, ya que no tendría en cuenta las variaciones temporales ni el contexto de los datos.

- Utilizar un algoritmo intermedio que combinara elementos estadísticos y temporales para realizar la detección de incidencias con un nivel adecuado de fiabilidad y sensibilidad. Este algoritmo debía ser capaz de reflejar la realidad cotidiana de los usuarios y captar las anomalías relevantes para su salud o bienestar, teniendo en cuenta variaciones temporales y el contexto que se le presentan.

Tras evaluar las ventajas e inconvenientes de cada alternativa, se optó por la tercera opción, que consistía en emplear un algoritmo conocido como z-scores [7], que se explica a continuación.

5.1.1 Algoritmo de z-scores

El algoritmo de z-scores es un algoritmo matemático y estadístico que permite detectar anomalías en una serie temporal de datos. Este algoritmo se basa en el principio de

dispersión, que mide el grado de variabilidad de los datos respecto a un valor central, como la media o la mediana.

El algoritmo calcula la media móvil y la desviación estándar de los datos en una ventana temporal definida. Luego, compara cada nuevo dato con la media móvil y determina si se aleja demasiado de ella, es decir, si supera un umbral de z-scores establecido por el usuario. Si el dato supera el umbral, el algoritmo lo considera una anomalía o incidencia y genera una alerta

De forma muy simplificada, este algoritmo funciona de la siguiente forma, al introducir un nuevo punto en los datos, si este se encuentra a una distancia determinada de la media móvil, una media que varía en el tiempo, el algoritmo lo detecta como una incidencia.

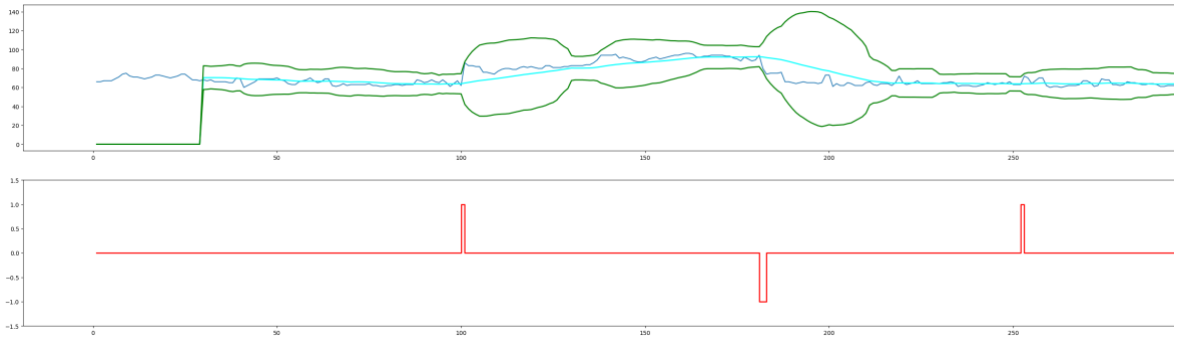


Figura 11. Ejemplo de z-scores.

La *Figura 11* es un claro ejemplo del comportamiento del algoritmo Z-scores, en la parte superior observamos una línea de color azul oscuro, representando el nivel de sonido en cada momento. A su vez, una línea azul claro, que representa la media móvil en cada momento. Tras estas, dos líneas verdes que representan la variación hacia arriba o hacia debajo de la variable de estudio en el tiempo definido.

En la parte inferior, en este caso, nos encontramos con varios picos, suponiendo los picos positivos, una incidencia. Esta incidencia viene dada por un gran aumento del sonido en el espacio de tiempo definido.

Ejemplificando un caso de uso real, si en una grabación en un momento dado, los decibelios empiezan a subir lentamente por cualquier razón externa, el algoritmo reconocerá este patrón y no lo detectará como una incidencia, ya que es un aumento gradual. Pero en cambio, si el aumento es muy pronunciado en un periodo corto de tiempo, lo detectará como incidencia y lo marcará.

Este algoritmo es ideal para nuestro caso, ya que presenta varias ventajas. En primer lugar, es un algoritmo simple y eficiente que se puede implementar fácilmente con herramientas informáticas. También, es un algoritmo adaptable que se ajusta a las variaciones temporales y contextuales de los datos, ya que la media móvil y la desviación estándar se actualizan constantemente con los nuevos puntos. Y finalmente, este es un algoritmo flexible que permite modificar el nivel de sensibilidad y fiabilidad del sistema modificando el tamaño de la ventana temporal y el valor del umbral de z-scores. De esta

forma, se puede adaptar el algoritmo a las características y necesidades específicas de cada variable de estudio, sea sonido, ritmo cardiaco o sueño.

5.1.2 Implementación del algoritmo

La implementación del algoritmo de detección de incidencias requiere de varios procesos previos para garantizar un análisis preciso y una detección fiable de las incidencias. Antes de llevar a cabo la aplicación del algoritmo, se debe realizar una limpieza, filtrado y preprocesado de los datos obtenidos para asegurar la integridad y calidad de los mismos y evitar el análisis de datos erróneos o incompletos.

```
## Filtering
# Delete duplicates
df = df.drop_duplicates()

# Delete uuids without heartRate
df = df[df['heartRate'] != -1]

# Delete uuids with low number of rows
uuid_counts = df['uuid'].value_counts()
to_keep = uuid_counts[uuid_counts >= 100].index
df = df[df['uuid'].isin(to_keep)]

# Delete uuids that have a value lower or equal than 30 as the average
value of the dB column
uuid_group = df.groupby('uuid')['dB'].agg(['mean'])
uuid_group = uuid_group[uuid_group['mean'] > 30]
df = df[df['uuid'].isin(uuid_group.index)]

# Delete uuids without valid coordinates
df = df[df['geometry'].apply(is_point_valid)]

# Use the remove_initial_neg funtion for every uuid
df.groupby('uuid').apply(remove_initial_neg)
```

Código 1. Código fuente del filtrado y preprocesado de los datos.

Como se puede apreciar en este código *Código 1*, se realiza el filtrado de los siguientes datos:

- Se descartan las grabaciones duplicadas, puede existir dos versiones de la misma grabación, una incluyendo algunos datos y otra sin incluirlos, se cogerá la grabación con un mayor número de valores.

- Se descartan a su vez todas las grabaciones que no tienen datos de salud como ritmo cardiaco y sueño, ya que en esta ocasión no nos sirven para este propósito.

- Se descartan todas aquellas grabaciones que sean inferiores a 20 minutos seguidos de grabación, ya que estas no suponen una prueba concluyente.

- Se descartan también todas aquellas grabaciones donde su media a lo largo de toda la grabación sea inferior a 30 dB, ya que son valores anómalos e imposibles en condiciones normales.

- Se descartan todas aquellas grabaciones que contengan datos de ubicación erróneos o no posibles, ya que impiden su asignación geográfica y su inclusión en los mapas geoespaciales.

- Se ha realizado un preprocesado de las grabaciones válidas, eliminando los valores negativos al inicio de cada grabación. Esto se ha hecho para asegurar que el análisis se centra en los datos correspondientes al periodo de sueño del usuario, que es el objetivo del proyecto.

Finalmente, como resultado, se obtienen unos datos limpios, verificados y listos para ser analizados.

```
def thresholding(y, lag, threshold, influence):
    signals = np.zeros(len(y))
    filteredY = np.array(y)
    avgFilter = np.zeros(len(y))
    stdFilter = np.zeros(len(y))

    avgFilter[lag - 1] = np.mean(y[:lag])
    stdFilter[lag - 1] = np.std(y[:lag])

    for i in range(lag, len(y)):
       # If the difference between the current point and the
       moving average is greater than the threshold times,
       it means incidence.
        if abs(y[i] - avgFilter[i-1]) > threshold * stdFilter[i-1]:
             signals[i] = 1 if y[i] > avgFilter[i-1] else 0

             # Set the filtered value as the influence of
             the current point
             filteredY[i] = influence * y[i] + (1 - influence) *
             filteredY[i-1]

       # If not, there isn't an incidence.
        else:
             signals[i] = 0
             filteredY[i] = y[i]

        # Recalculate the mean and the desviation
        avgFilter[i] = np.mean(filteredY[(i-lag+1):i+1])
        stdFilter[i] = np.std(filteredY[(i-lag+1):i+1])

    return signals
```

Código 2. Código fuente del algoritmo de z-scores.

Este código realiza la detección de incidencias a partir del algoritmo z-scores, para poder llevar a cabo esta detección se le pasan los siguientes parámetros:

- y: es una lista o array de valores que contiene los datos a analizar.

- lag: el número de puntos que se usan para calcular la media y la desviación estándar móviles.

- threshold: es un número que indica el umbral de z-score que se usa para determinar si un punto es una incidencia o no. Un valor más alto significa que se requiere una mayor desviación para considerar una incidencia.

- influence: es un número entre 0 y 1 que indica el grado de influencia que tiene un punto detectado como incidencia sobre los cálculos posteriores de la media y la desviación estándar. Un valor más alto significa que se le da más peso al punto como incidencia y se reduce el efecto de los puntos normales.

El algoritmo compara cada punto de la serie temporal con la media y la desviación estándar móviles de los lag puntos anteriores. Si la diferencia entre el punto y la media es mayor que el umbral multiplicado por la desviación estándar, se considera que el punto es una incidencia y se le asigna un valor de 1. Además, se aplica un factor de influencia al punto para reducir su impacto en el cálculo de la media y la desviación estándar móviles siguientes. Si el punto no es una incidencia, se le asigna un valor de 0 y se usa tal cual para actualizar la media y la desviación estándar móviles. Este finalmente, devuelve un vector de enteros donde se indican con 1 o 0 si, en ese momento del tiempo, se ha detectado una incidencia o no respectivamente.

Este algoritmo se ha aplicado a la detección de incidencias de sonido y de ritmo cardiaco, dos variables con características distintas que requieren ajustar los parámetros de entrada del método. Para el sonido, se ha utilizado un rango de tiempo menor y un umbral más bajo para así ajustar la detección a cambios más precisos en un espacio corto de tiempo. En cuanto al ritmo cardiaco, se han aumentado estas dos variables ya que se tiene en cuenta un mayor número de puntos anteriores para la detección de la incidencia actual. Así, se ha logrado una detección eficaz de las incidencias en ambas variables.

5.1.3 Incidencias en el sueño

Tras la realización de la detección de incidencias de sonido y ritmo cardiaco, es hora de hacer lo propio con los diferentes estados del sueño de la persona. En este caso, aplicar un algoritmo como z-scores no obtendría los resultados esperados, ya que, los estados del sueño no se distribuyen de forma normal, sino que tienen los siguientes valores:

Estado del sueño	Valor
Despierto	-1
Sueño ligero	1
Sueño profundo	2
Fase REM	3

Tabla 1. Equivalencias entre estado del sueño y su valor.

La tabla anterior, *Tabla 1* muestra la correspondencia entre el valor obtenido y el estado del sueño asociado. Los estados del sueño no son aleatorios, sino que durante la noche siguen un ciclo natural que varía de forma cíclica en el tiempo [8][9]. Por lo tanto, una persona pasa por todos los estados del sueño de forma continua sin que esto suponga que ha habido ninguna incidencia. Sin embargo, se considera una incidencia cuando una persona pasa de un estado de sueño profundo o fase REM a un estado despierto de forma abrupta.

Por esta razón, el algoritmo utilizado para detectar las incidencias del sueño es diferente al de los casos anteriores.

```
def sleep_incidences (data):
    # Create a list to store the final data
    data_list = []
    # Create a variable to store the position of the records
    position = 0
    # Create a variable to store the initial position of the incidence
    initial_incidence_position = 0

    for row in sleepStage:
        # Know if the user is in deepSleep or higher
        if row >= 2:
            # If the user is sleeping, no incidence
            data_list.append(0)
            # Update the initial incidence position
            initial_incidence_position = position

        elif row <= 0 and initial_incidence_position >= position - 3
and initial_incidence_position != 0:
            # If the record have not sleepStage data and the initial
incidence position is less than or equal to 3 records of the current
position, add a 1 to the final data list (detected incidence)
            data_list.append(1)
            # Update the initial incidence position to 0
            initial_incidence_position = 0

        position += 1

    return data_list
```

Código 3. Código fuente de la detección de incidencias en el sueño.

Una vez realizadas las detecciones de incidencias de sonido, ritmo cardiaco y sueño, se formatean los datos según el apartado 4.1.2 y se exportan a archivos distintos para ser procesados por el generador de mapa.

5.2 Generación de mapas

Tras la anterior explicación sobre la detección de incidencias, es hora de empezar a trabajar con los datos obtenidos en el punto anterior y obtener unos mapas de calor a partir de estas mismas incidencias.

Para la realización de esta tarea se ha hecho uso de la librería de Python llamada Bokeh [10]. Esta librería es capaz de realizar a partir de unos datos obtenidos, unos mapas de calor, pudiendo configurar y personalizar una gran variedad de opciones.

5.2.1 Importación de los archivos

Pero antes de realizar cualquier paso es necesario realizar la importación de los archivos con los datos de los usuarios para poder trabajar con ello. Para realizar esta tarea primeramente se realizan una serie de cálculos para obtener en un formato YYYY-MM-DD, donde *Y* significa el año, *M* el mes y *D* el día exacto, los archivos exactos que descargar. Este proceso es necesario ya que se realizan tres tipos de mapas por ciudad: mapas basados en los datos recogidos la noche anterior, los últimos 7 días y los últimos 30 días. Por lo tanto, es necesario saber exactamente los días que corresponden a los últimos 30 días.

Una vez obtenidos los nombres de los archivos correspondientes se realiza la descarga de los archivos a través de las librerías para Python de Google Cloud se realiza una importación a GeoPandas [11].

```
def read_data(cities: list, dates: list) -> GeoDataFrame | str:
    record_data = pd.DataFrame()

    # Read all files in the bucket
    list_bucket = list(bucket_data.list_blobs())

    for city in cities:
       for day in dates:
            for name in list_bucket:
                if name.name.startswith(city + "_" + day + ".csv"):

                   bucket_data.blob(name.name).download_to_filename(
                   name.name)
                   record_data       =       pd.concat([record_data,
                   pd.read_csv(name.name)], ignore_index=True)

    if not record_data.empty:
       record_data.pop('uuid')
       record_data['geometry']                                          =
       record_data['geometry'].apply(wkt.loads)
             return GeoDataFrame(record_data)
    else:
       return ""
```

Código 4. Código fuente de la descarga e importación de los datos.

GeoPandas es una librería que nos permite la lectura de los archivos, realizar operaciones sobre estos y consultas con tal de facilitar toda la gestión de los mismos. Esta

librería, a diferencia de Pandas, nos permite también realizar ciertas operaciones únicas con la ubicación geoespacial que contienen estos datos.

5.2.2 Clustering

Una vez realizada la importación de los datos, es necesario un proceso previo llamado *clustering*. Este proceso consiste en la agrupación de ciertos puntos cercanos en una serie de *clusters* o zonas cercanas.

Este proceso ha sido necesario ya que nos ha permitido unificar cientos de puntos que se encontraban a un ratio muy cercano del mapa, en un único punto, permitiendo evitar tener cientos de puntos unos encima de otros en el propio mapa.

Esto nos permite tener una vista mucho más simple e intuitiva de comprender para un usuario dentro de una aplicación móvil.

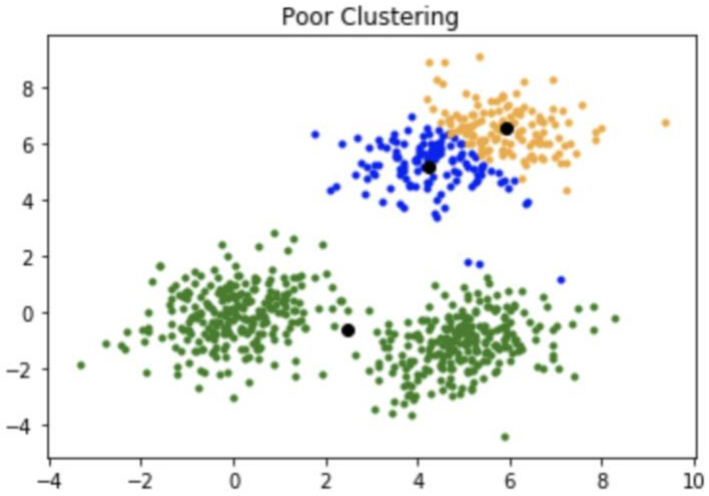


Figura 12. Ejemplo de K-means (clustering).

El proceso de clustering se ha realizado mediante el algoritmo de K-means [12], este algoritmo nos permite encontrar un centroide en una agrupación de puntos. Este centroide representa el punto medio de todos los puntos encontrados en esa zona. Estos centroides están representados como los puntos negros de la *Figura 12*.

Con tal de poder obtener centroides excesivamente alejados de las agrupaciones de puntos reales (tal y como ocurre en la zona verde de la *Figura 12*), se realiza una operación auxiliar que evita obtener centroides más alejadas de 1 Km a la redonda.

```
def get_max_distance(centroide, points: list) -> float:
    # For every point in the list of point
    distance = 0
    for point in points:
        new_distance = centroide.distance(Point(point))
        if new_distance > distance:
            distance = new_distance

    distance = distance * 4000
    if distance < 11:
        distance = 13
    return distance
```

Código 5. Código fuente del cálculo máximo de distancias.

Finalmente, tras la realización de todos estos cálculos, se inicia la generación del archivo HTML donde se mostrarán gráficamente todas las incidencias detectadas.

5.2.3 Generación del mapa HTML

Tras todo este proceso nos encontramos con los datos filtrados, importados y organizados de forma que nos permitan obtener un mapa sencillo y representativo con toda la información obtenida por los usuarios. Para la generación del mapa HTML se utiliza una librería de Python llamada Bokeh [10], que permite generar una vista dinámica de los datos.

```
def plot_map(geo_maps: list, geo_points: list):
    p=figure(height=500,width=1000, axis_visible=False)

    for geo_map in geo_maps:
        # Add patch renderer to figure.
        p.patches('xs', 'ys', source=geo_map)

    # If there are any point to show
    if geo_points != "":
       # Description of the different types of incidences
       tooltips = ['Incidències per soroll', 'Incidències al ritme
       cardíac', 'Incidències al somni']

       for geo_point in geo_points:
           i = len(geo_point) - 1
           while i > -1:
               # If is a Polygon to representate
               if 'Polygon' == geo_point['geometry'][i].geom_type:
                   # Get the peak hour in this cluster
                   hours_list=geo_point['timestamp'][i].split(',')
                   peak_hour = max(hours_list)

                   # If the polygon only have 2 points print orange
                   if len(geo_point['geometry'][i]) == 2:
                         shape=p.circle(position=geo_point,
                         size=geo_point.size, color='orange')
                   else:
                        centroid = geo_point['geometry'][i].centroid
                        distance=get_distance(centroid,geo_point[i])

                         shape=p.circle(position=centroid,
                         color='red', size=distance)

                    p.add_tools(HoverTool(renderers=[shape]))

                i = i - 1

           # Print the rest of the points
           points=p.circle(x='x',  y='y',  size=9,  color='yellow',
           source=geo_point))
       p.add_tools(HoverTool(renderers=[points],tooltips=[tool])))
```

Código 6. Código fuente de la generación de mapas.

Para poder entender este código lo dividiremos en varias secciones con tal de poder explicar mejor cada una de sus funciones y su implementación.

En las primeras líneas de código, se define como será el mapa, sus dimensiones y características para que su visualización sea la óptima para una vista móvil. Posteriormente, se agrega a la vista HTML el contorno del mapa de la ciudad correspondiente, sea Tarragona, Reus o la provincia de Tarragona.

Una vez, se ha definido el mapa y la vista principal es hora de añadir los puntos con las incidencias detectadas.

Para ello se realiza un bucle, donde por cada uno de los puntos geoespaciales detectados en nuestros datos se realizan las siguientes comprobaciones. Si este está definido como un Polígono, es decir, un conjunto de incidencias en una misma zona próxima, se calcula la hora de mayor número de incidencias en base a las incidencias detectadas. Este último proceso es un paso adicional para así poder facilitar más información sobre lo ocurrido en esa zona al usuario final.

Una vez se ha comprobado que el polígono correspondiente tiene más de 1 punto en su interior, se realiza una última comprobación sobre el número de incidencias que contiene este punto. Este último paso nos permite a partir del número de incidencias que contiene ese el polígono asignarle un color representativo con respecto a la gravedad de la zona. Estos van desde rojo a verde, pasando por naranja. La correspondencia sería la siguiente:

Número de incidencias	Color
$x = 1$	Verde
$1 < x < 5$	Naranja
$x > 5$	Rojo

Tabla 2. Equivalencias entre número de incidencias y color.

Todo este proceso trata de generar un mapa con la información importante en su interior, es decir, hora de máxima incidencia, e intentando que sea lo más representativo posible gracias a su escala de colores. Un claro ejemplo de este proceso es la *Figura 13*, donde se muestra la generación de un mapa con datos aleatorios en la ciudad de Tarragona.

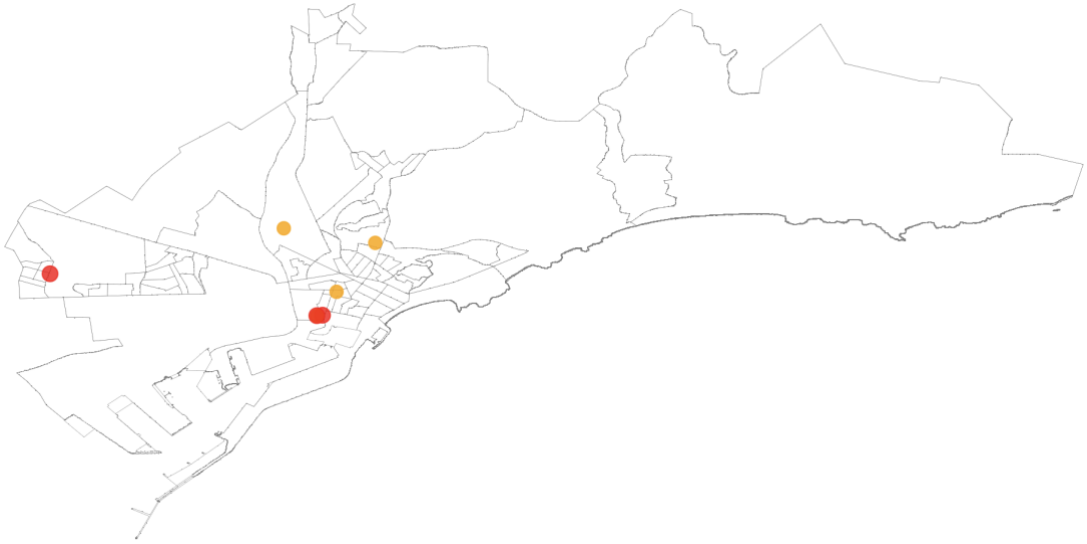


Figura 13. Mapa generado en base a datos reales.

Finalmente, tras la realización de estos cálculos y la posterior generación del mapa en formato HTML, se almacena en un archivo en el disco local y finalmente se sube al Cloud Storage para poder ser leído por la aplicación móvil.

5.3 Aplicación Android

En cuanto a la aplicación móvil desarrollada para Android, podemos decir que esta es la pieza central del proyecto. Esta es la que mantiene le contacto directo con el usuario al permitir la recopilación de datos de salud y sonido, todo ello de forma segura y anónima, y a su vez permite recibir informes sobre las incidencias ocurridas gracias a los mapas de ruido generados.

Por ello, esta es la pieza fundamental, al tener que realizar un diseño y desarrollo de forma robusta y escalable, para que cualquier tipo de persona con cualquier tipo de dispositivo pueda ser parte de este proyecto.

En cuanto al desarrollo de esta aplicación móvil, como mencionamos en el punto 4.2.1, está realizado mediante el patrón de diseño llamado Modelo-Vista-VistaModelo. Este patrón permite separar y distribuir las funcionalidades entre los componentes de vista, modelo y el modelo de negocio. Permitiendo así una mayor comprensión y posibilidad de mejorar y expandir el código en un futuro.

Por ello, la siguiente figura nos permite hacer una vista general de la función de cada una de las clases desarrolladas en la aplicación móvil y la parte a la que corresponde dentro del patrón de diseño mencionado.

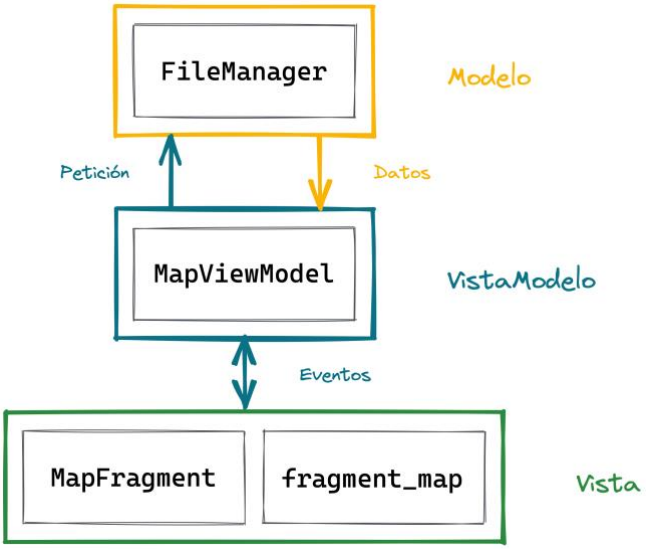


Figura 14. Distribución de clases dentro del patrón MVVM.

Como observamos en la *Figura 14*, la obtención de los mapas generados, gestión de ellos y mostrado de los mismos se realiza en base a estas cuatro clases, dos de ellas en relación con la vista y a la forma de mostrar estos datos.

Para poder obtener un poco más de información y así obtener una visión más detallada se muestra en la siguiente imagen los métodos que contienen cada una de estas clases, excepto *fragment_map*, ya que corresponde a un fichero XML cuya función es simplemente estructurar los elementos visuales y como se mostrarán en la pantalla del usuario.

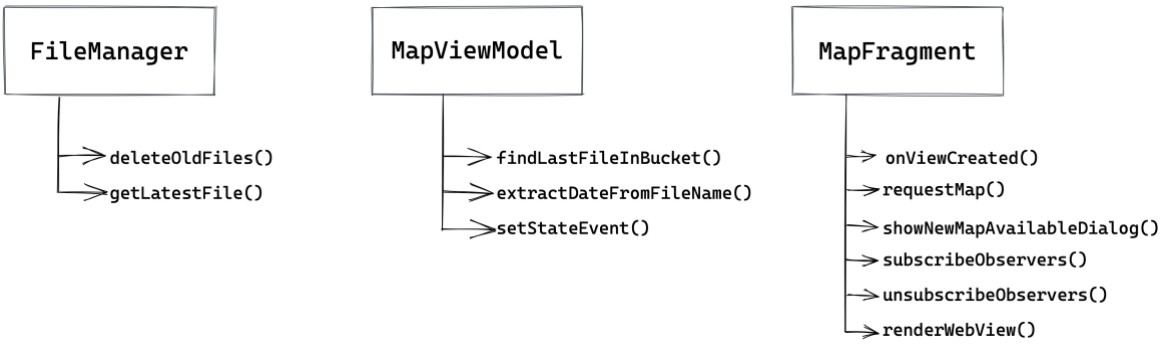


Figura 15. Clases y sus métodos.

Al observar en estas dos figuras, *Figura 14* y *Figura 15*, podemos separar las funcionalidades de cada una de las clases en tres componentes. Modelo, Vista y VistaModelo, haciendo cada uno de estos componentes a una parte específica de la aplicación. Para explicar mejor el funcionamiento de estas se ha decidido dividir la información en tres apartados, cada uno correspondiente a uno de los componentes mencionados anteriormente.

5.3.1 Modelo

Este componente es el correspondiente a los datos, es decir, como se estructuran, gestionan y convierten estos datos dentro de la aplicación.

En este caso, nuestros datos y, por tanto, nuestro modelo, es un fichero HTML con la información sobre los mapas de ruidos generados anteriormente. Mapas que previamente han sido subidos a Google Cloud Storage para poder ser descargados por los usuarios desde sus dispositivos.

La clase correspondiente al Modelo, *FileManager*, contiene dos métodos: *deleteOldFiles()* y *getLatestFile()*. Cada uno de estos métodos se encarga de la gestión de los ficheros HTML para eliminarlos u obtenerlos correspondientemente.

La primera función corresponde a la de eliminar ficheros antiguos, esta es una función de gran importancia y de las grandes funcionalidades que obtienen estos mapas de forma invisible para el usuario. Esta función nos permite tener una gestión eficiente del almacenamiento de la aplicación con respecto a los mapas descargados, pudiendo eliminar contenido antiguo cuyo uso ya no será necesario.

Esta función realiza la eliminación de los mapas antiguos, solamente existe un mapa con fecha más reciente. Esta condición nos permite que en caso de que el usuario abra la aplicación sin conexión a internet, poder obtener los mapas ya descargados hasta el momento en el que se realice la obtención de uno nuevo, evitando así tener pantallas de carga infinitas.

En cuanto a la segunda función, esta corresponde a la gestión y obtención del fichero más reciente en el almacenamiento interno de la aplicación, siendo de vital importancia para los siguientes componentes que explicaremos a continuación.

Una vez explicado toda la gestión de los ficheros locales por parte de la aplicación, es hora de explicar el componente VistaModelo o también llamado “modelo de negocio”.

5.3.2 VistaModelo

En cuanto al componente correspondiente del modelo de negocio, nos encontramos con una clase con un número de métodos limitados, ya que la función de este es la comunicación y el paso de mensajes entre los ficheros locales obtenidos por el Modelo y la representación de estos en la Vista. En este caso, al no tener que tratar los datos obtenidos, simplemente se realiza la descarga y representación de estos, no se necesita un gran modelo de negocio.

Para realizar toda la comunicación entre ambos componentes se realiza mediante eventos y llamadas a los diferentes métodos que contiene esta clase. Estos eventos son capturados y emitidos mediante los tipos de datos *MutableLiveData*. Este tipo de datos son una serie de colas donde se escuchan y envían los diferentes eventos que suceden durante la ejecución.

```
private fun findLastFileInBucket(
        mapDate: MapDate,
        location: String,
        onSuccessListener: (String) -> Unit
    ) {
        storage.listAll()
            .addOnSuccessListener { (objects, _) ->
                val regex = "${location}_\\d+-\\d+-\\d+".toRegex()
                val fileName = objects
                .filter {
                    it.name.contains(location) &&
                    it.name.contains(mapDate.alias) &&
                    regex.find(it.name)?.value != null
                }
                .map{        file        ->        file.name        to
             extractDateFromFileName(file.name) }
                .filter { (fileName, date) -> date != null }
                .maxByOrNull { (fileName, date) -> date!! }
                ?.first
                if (fileName != null) onSuccessListener(fileName)
            }
    }
```

Código 7. Código fuente de la función findLastFileInBucket.

El primero de los métodos, también llamado *findLastFileInBucket()*, como se observa en el *Código 7*, es el encargado de realizar la conexión con la base de datos de Google Cloud Storage, mediante Firebase, mencionados anteriormente. Para finalmente obtener el nombre del fichero más reciente en el almacenamiento en la nube.

Este método nos permite saber cuándo se necesita realizar una actualización de los mapas actuales o cuando el usuario ya cuenta con el fichero más reciente, y, por tanto, no necesita más ficheros.

A continuación, nos encontramos con el método *extractDateFromFileName()*. Tal y como indica su nombre, es un método simple que nos permite extraer la fecha en un formato *LocalDateTime*, para así poder trabajar sobre ella y ser utilizada en otros métodos como *findLasFileInBucket()* o *setStateEvent()*.

Finalmente, el último método llamado *setStateEvent()* es el encargado de recibir todos los eventos que ocurren por parte de la vista y realizar las comprobaciones, obtenciones y gestiones necesarias para poder satisfacer los datos que se piden.

En este caso contamos con tres eventos posibles:

- GetMapFile: este evento corresponde a la petición de obtener el fichero correspondiente al mapa a mostrar. Al recibir esta petición, el modelo de negocio es el encargado de obtener el último mapa sea en local y comprobar si el mapa corresponde al último en el almacenamiento en remoto y en caso contrario notificar a la vista para que haga una solicitud para la descarga del nuevo mapa.
- DownloadNewFile: esta solicitud corresponde a la situación mencionada anteriormente. Nos permite, en caso de que exista una nueva versión de los mapas

generados en el almacenamiento en la nube, realizar la descarga de este y almacenarlo en la memoria interna.

Para ello hace uso de las APIs proporcionadas por Google, donde nos permite a través de unas claves privadas, realizar la conexión con los servidores del proyecto y descargar el fichero correspondiente al último mapa generado.

- CancelDownload: finalmente, este evento es el que le permite al modelo de negocio saber cuándo el usuario no desea actualizar a un nuevo mapa, sino que desea visualizar el mapa ya descargado.

```
setNegativeButton(getString(R.string.cancel)) { _, _ ->
        viewModel.setStateEvent(
                MapStateEvent.CancelDownload(true)
        )
 }
```

Código 8. Código fuente del envío del evento CancelDownload.

A la hora de generar una respuesta y devolver los datos necesarios desde la VistaModelo hacia la Vista, también se realiza mediante la emisión y recepción de eventos, pero esta vez de forma inversa.

Si los eventos anteriores siempre eran unidireccionales, es decir, desde la vista al modelo de negocio, en este caso son en sentido contrario, desde el modelo de negocio a la vista.

Para la realización de respuestas se especifican los siguientes eventos que pueden ocurrir:

- MapUpdated(file): en este caso, al realizar este evento, se le especifica a la vista que existe un nuevo mapa descargado y almacenado en la memoria local, listo para ser mostrado, y a su vez se le proporciona en el propio cuerpo del evento, el fichero con el mapa correspondiente.
- NoMapAvailable: este evento corresponde a la situación donde el usuario ya contenga el mapa más reciente y no sea necesario obtener ninguno.
- NewFileAvailable: esta solicitud es la que, al ser escuchada por la vista, activa automáticamente el Pop-up mostrado en la *Figura 9*, donde se le notifica al usuario que existe un nuevo mapa más reciente y se le da la opción para realizar la descarga o no.
- NoState: para concluir, la función de este evento es la de devolver una respuesta de confirmación a la vista en caso de recibir la cancelación de la descarga o la respuesta a no encontrar ningún fichero ni en local ni en remoto que mostrar.

Como podemos observar, toda la comunicación realizada entre la vista y el modelo de negocio se hace a través de eventos hacia un sentido y hacia el inverso. Proporcionando así una forma de comunicación asíncrona, permitiendo la ejecución de otras tareas a la espera de las respuestas necesarias.

Esta es una forma de realizar las aplicaciones más escalables y mantenibles, pudiendo así implementar nuevas funcionalidades de forma sencilla, simplemente implementando nuevos eventos y observadores que escuchen las llamadas de estos métodos.

Para la gestión necesaria entre la VistaModelo y el Modelo, simplemente se hacen llamadas a las funciones correspondientes, ya que son métodos rápidos y que no requieren de una conexión hacia internet, haciendo así el código más sencillo.

5.3.3 Vista

Finalmente, la vista es la encargada de realizar las peticiones necesarias al modelo de negocio, y a partir de las respuestas recibidas por este, realizar o notificar de las acciones necesarias al usuario.

En este caso nos encontramos con una clase con múltiples métodos, suponiendo esta, la clase con mayor importancia, ya que es la encargada de toda la gestión y comunicación visual con el usuario.

En este caso, nos encontramos con el primer método llamado *onViewCreated()*, cuya función es la definición de los elementos y variables necesarios para la ejecución del modelo de negocio entre los componentes. Además, se define los pasos que se deben realizar en caso de que el usuario desee cambiar entre los mapas generados en base a la noche anterior, en base a los datos recopilados en los últimos 7 días o en los últimos 30 días.

A continuación nos encontramos con un método llamado *requestsMaps()*, este método simplemente llama al evento *GetMapFile* hacia el modelo de negocio para que así, este le devuelva una respuesta.

El método *showNewMapAvailableDialog()* es el encargado de crear y generar el Pop-up mostrado en la *Figura 9*, donde a partir de la opción escogida por parte del usuario se emite un evento hacia el modelo de negocio. Estos eventos son: *DownloadNewFile* en caso de querer actualizar el mapa actual por el mapa más reciente o *CancelDownload* en caso de no querer realizar esta acción.

Para ir finalizando nos encontramos con dos métodos más llamados *suscribeObservers()* y *unsuscribeObservers()*, cuya función es suscribirse a los eventos respuesta que puedan ser generados por la VistaModelo o cancelar la suscripción. Realizar una acción u otra lo que nos permite es especificar si queremos que el programa mantenga un canal donde recibir estas respuestas o, por el contrario, no es necesario y podemos cerrar estos mismos canales.

```
fun renderWebView(file: File) {
        val filepath = file.absolutePath
        val webView = binding.webView

        webView.settings.javaScriptEnabled = true
        webView.settings.allowFileAccess = true
        webView.setInitialScale(250)
        webView.scrollTo(200, 125)
```

```
            // Disallow the touch request for parent scroll on touch of
child view
            webView.setOnTouchListener { v, _ ->
                v.parent.requestDisallowInterceptTouchEvent(true)
                false
            }

            webView.loadUrl("file:///$filepath")
        }
```

Código 9. Código fuente de la función renderWebView.

Y finalmente, nos encontramos con el ultimo método llamado *renderWebView()*. Como se puede observar en el *Código 9*, este método es el encargado de, a partir del fichero HTML obtenido por el Modelo a través del modelo de negocio, mostrar el mapa correspondiente en base a las configuraciones definidas.

Como se puede observar en el código, se definen parámetros sobre cómo se debe mostrar la vista inicial del mapa, en este caso con la vista puesta en el centro de este, parámetros como se debe comportar la vista al recibir una pulsación en la pantalla por parte del usuario, entre otras configuraciones relacionadas con la lectura del fichero y sus funciones JavaScript.

5.3.4 Interfaz gráfica

Tras la explicación de cada uno de los componentes del patrón MVVM, es momento de explicar cómo funciona y como se ha definido la interfaz gráfica. Hasta ahora habíamos hablado de Vista, pero desde el punto de vista de la programación y como está realiza las acciones con código, pero no se explica cómo está diseñada la interfaz. Por este motivo, en este punto es donde se abarcará todo lo relacionado con los componentes visuales.

Para la realización de las vistas en Android se utiliza una serie de ficheros XML (*Extensible Markup Language*). Este consiste en un metalenguaje con el que a partir de una interfaz gráfica y/o código programático proporcionado por Android Studio, se definen los componentes que se mostrarán en pantalla.

Tal y como se puede observar en las *Figuras 8, 9 y 10*, esta es una interfaz sencilla, con un simple botón y una serie de elementos para mostrar un WebView[3] donde se renderizará el mapa de incidencias correspondiente.

3 WebView: navegador web con capacidad de renderizar archivos HTML integrado en una aplicación nativa.

Figura 16. Vista de la interfaz gráfica y sus componentes

Como se puede observar en la *Figura 16*, tras el desarrollo de esta interfaz obtenemos tres elementos principales: un texto donde se mostrará la información en cada uno de los idiomas soportados por la aplicación (español, catalán e inglés), un botón donde abrirá un desplegable para seleccionar las fechas y el mapa correspondiente.

Cada uno de estos elementos está diseñado para poder ser adaptado a cualquier tipo de pantalla, sea de cualquier tamaño. Esto se define gracias a las propiedades *layout_constrain* donde se le especifica las posiciones que tiene que tomar cada elemento. Tal y como se muestra en el siguiente código:

```
<WebView
        android:id="@+id/webView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="16dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/linearLayoutMap">
    </WebView>
```

Código 10. Código fuente de la interfaz gráfica de webView.

A su vez, se le especifican una serie de márgenes y espacios entre los distintos elementos para obtener una interfaz más simple y visual hacia el usuario final.

Finalmente, tras la ejecución e implementación de cada una de las partes de cada uno de estos componentes, se obtiene el resultado final. Este resultado se define como la obtención, gestión y visualización de los mapas y opciones dadas en la interfaz de forma eficiente, sencilla, escalable, mantenible y extensible a lo largo del tiempo.

6 Juego de pruebas

Una vez explicado todo el proyecto, su funcionamiento y su desarrollo, es hora de realizar la verificación del sistema.

En este juego de pruebas se harán verificaciones sobre escalabilidad, autonomía y una comparación entre los resultados esperados y los resultados obtenidos en cada uno de los procesos del sistema. Para ello haremos una separación en diferentes secciones a verificar.

6.1 Escalabilidad

Para comprobar la escalabilidad del sistema observaremos el día con mayor pico de peticiones y los errores devueltos por los servidores de Google.

El día de mayores solicitudes hacia el servidor fue el 30 de noviembre de 2022, como se observa en la *Figura 17*. En este día se realizaron más de 300 solicitudes hacia el servidor y se devolvieron 0 errores, reflejado en la siguiente figura, la *Figura 18*.

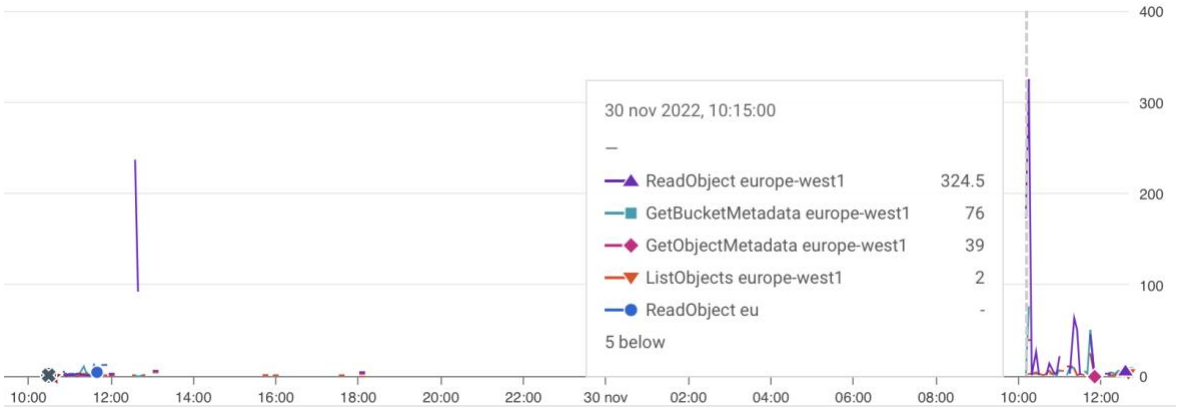


Figura 17. Número de visitas hacia el servidor.

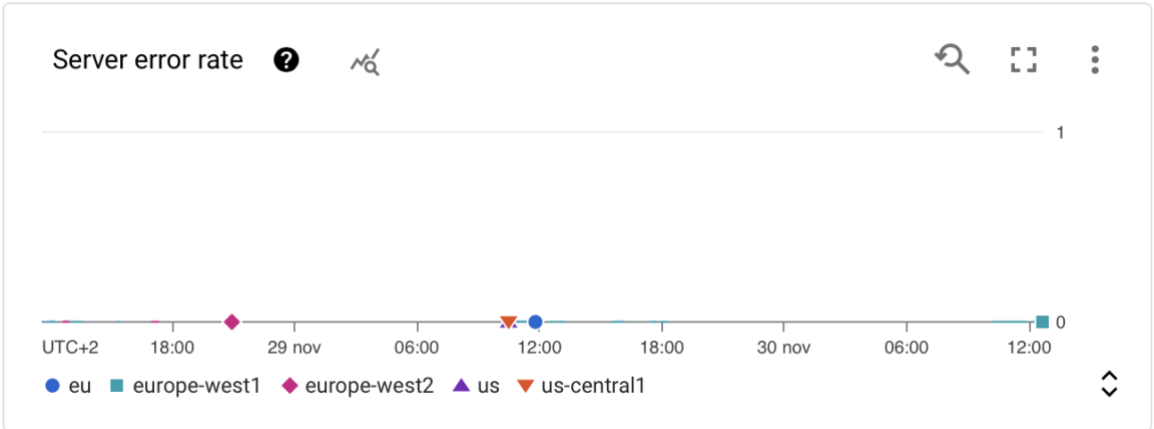


Figura 18. Número de errores del servidor.

6.2 Verificación de los resultados obtenidos

Una vez realizada la verificación sobre la escalabilidad del sistema, se realiza lo propio con los datos obtenidos y sus resultados.

```
uuid,lng,lat,timestamp,dB,heartRate,sleepStage,geometry
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474312369,30,51,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474323381,30,51,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474334390,30,52,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474345399,30,54,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474356408,30,52,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474367419,30,52,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474378427,30,50,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474389437,30,51,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474400446,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474411456,30,55,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474422467,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474433476,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474444489,30,54,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474455499,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474466509,30,51,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474477517,30,52,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474488526,30,55,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474499538,30,54,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474510544,30,54,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474521549,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474532552,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474543555,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474554560,30,52,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474565560,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474576560,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474587565,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474598572,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474609583,30,54,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474620593,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474631602,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474642610,30,55,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474653621,30,54,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474664629,30,54,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474675635,30,53,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474686646,30,54,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474697653,30,54,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474708659,30,54,2,POINT (1.2514325 41.124332)
b343a22a-df67-442a-9963-6e6f0227dd4a,1.2514325,41.124332,1668474719668,30,54,2,POINT (1.2514325 41.124332)
```

Figura 19. Fichero con datos obtenidos por un usuario ficticio.

En este caso, en la *Figura 19* nos encontramos con un ejemplo de fichero con datos obtenidos de forma aleatoria, gracias a un usuario ficticio. En este caso nos encontramos con una ubicación en la ciudad de Tarragona, con unos dB, ritmo cardiaco y sueño aleatorios.

Tras obtener los datos del usuario, se realiza la detección de incidencias de z-scores con la siguiente configuración: lag=20, threshold=3, influence=0.25. Siendo esta, la ideal para nuestro caso de uso.

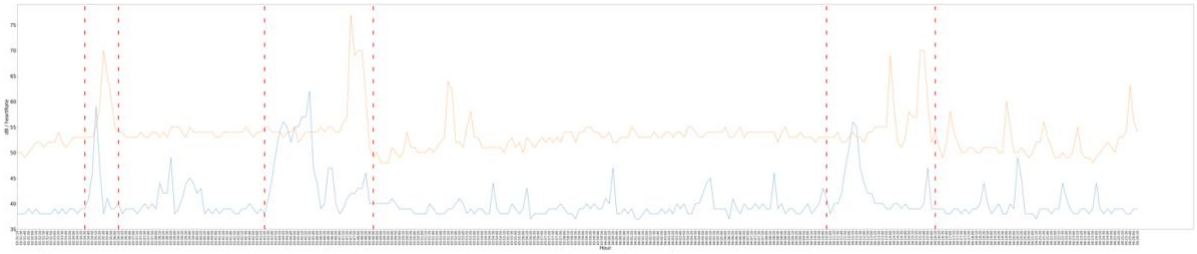


Figura 20. Nivel de dB y ritmo cardiaco del usuario de 05:00 a 06:00.

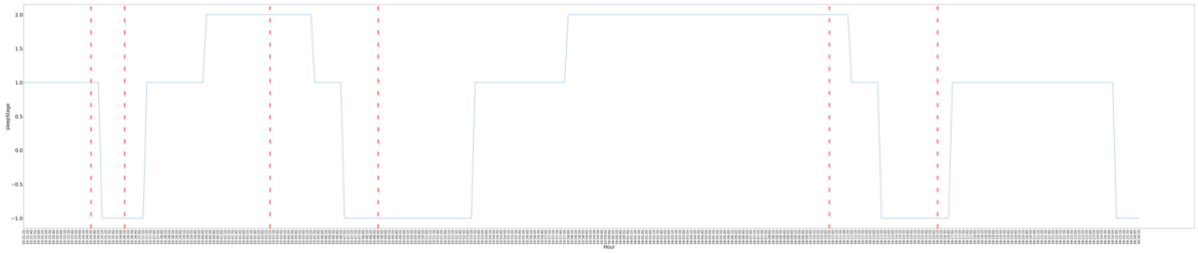


Figura 21. Estado del sueño del usuario de 05:00 a 06:00.

Como se puede apreciar en la *Figura 20*, el nivel de decibelios captados por el micrófono del dispositivo y el ritmo cardiaco detectado por la pulsera de actividad son valores muy parejos. Estos siempre cumplen la premisa de "acto-consecuencia", suponiendo el ruido un problema en la salud.

En la *Figura 21*, vemos estos niveles durante el mismo periodo, pero con respecto al estado del sueño, suponiendo valores altos, un nivel de sueño profundo, y niveles bajos, un sueño ligero o un despertar del usuario.

También se apreciar unas líneas verticales rojas para ejemplificar mejor los momentos en los que una incidencia de ruido afecta a la salud en cuanto al ritmo cardiaco y al sueño.

```
uuid,timestamp,incidenciadB,incidenciaHeartdB,incidenciaSleepdB,geometry
b343a22a-df67-442a-9963-6e6f0227dd4a,1668476738,1,0,0,POINT (1.2352637 41.127293)
b343a22a-df67-442a-9963-6e6f0227dd4a,1668475949,1,0,0,POINT (1.2352637 41.127293)
b343a22a-df67-442a-9963-6e6f0227dd4a,1668475222,0,1,0,POINT (1.2352637 41.127293)
b343a22a-df67-442a-9963-6e6f0227dd4a,1668476800,0,0,1,POINT (1.2352637 41.127293)
b343a22a-df67-442a-9963-6e6f0227dd4a,1668475354,0,1,0,POINT (1.2352637 41.127293)
```

Figura 22. Fichero resultante con las incidencias detectadas.

Una vez obtenidos los datos sobre las incidencias detectadas como se muestra en la *Figura 22*, es hora de realizar la generación de los mapas de ruido sobre estos datos.

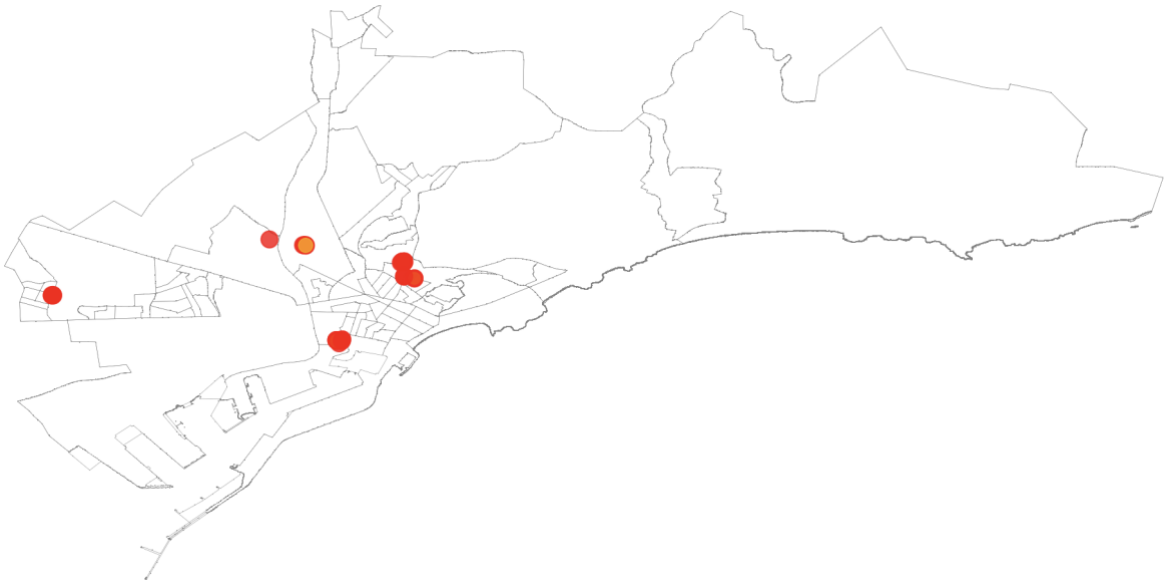


Figura 23. Mapa resultante de la ciudad de Tarragona.

Figura 24. Mapa resultante de la ciudad de Tarragona con solo una incidencia.

→ Ciudad de Tarragona.

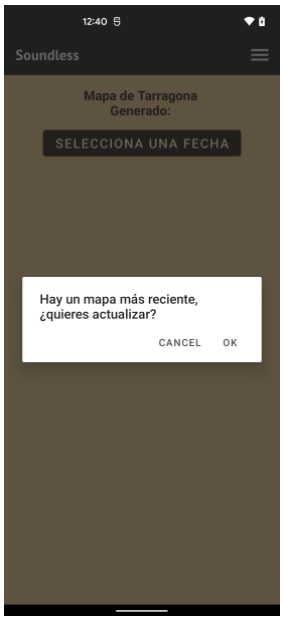
→ Ciudad de Reus.

→ Provincia de Tarragona.

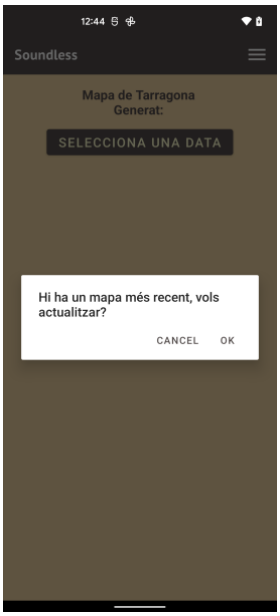
Figura 25. Mapas resultantes de las ciudades de Tarragona, Reus y provincia de Tarragona sin incidencias.

En este caso nos encontramos con tres imágenes distintas para verificar la generación de mapas en cada una de las situaciones propuestas. En el primer caso, la *Figura 23* representa el mapa de la ciudad de Tarragona con los datos aleatorios mencionados anteriormente. En la *Figura 24* se muestra esta ciudad, pero con un solo conjunto de incidencias de nivel medio (es decir, menos de 4 incidencias en la misma zona). Finalmente, la *Figura 25* viene a ejemplificar y a dar una visión de los mapas resultantes de las ciudades de Tarragona, Reus y la provincia de Tarragona, pero en este caso, sin mostrar ninguna incidencia.

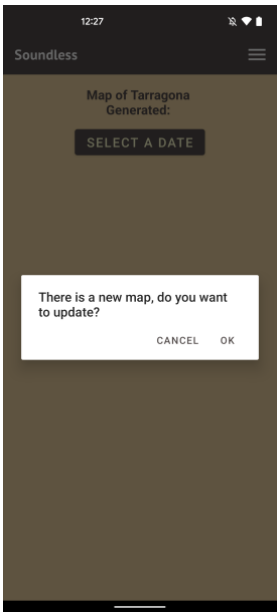
Tras la comprobación de los mapas generados, es hora de realizar las verificaciones necesarias con la aplicación móvil y sus distintas funcionalidades.



Notificación en castellano.



Notificación en catalán.



Notificación en inglés.

Figura 26. Notificación informando de la disponibilidad de un nuevo mapa.

Mapa de ayer.

Mapa de los últimos 7 días.

Mapa de los últimos 30 días.

Figura 27. Interfaz sobre las diferentes fechas seleccionadas.

En este caso nos encontramos en la *Figura 26* la notificación que se muestra para informar al usuario sobre un nuevo mapa disponible, en todos los idiomas disponibles. Posteriormente, en la *Figura 27* se muestra la interfaz de cada uno de los mapas correspondientes al día de ayer, últimos 7 días o últimos 30 días.

Finalmente, y para finalizar esta sección se comprueba el visualizado del mapa de ruido obtenido por la memoria local del dispositivo, es decir, comprobar si al no tener conexión a internet, el dispositivo sigue visualizando el ultimo mapa del almacenamiento interno.

Como se aprecia en esta imagen, *Figura 28*, el dispositivo está en modo avión y aun así permite la visualización del último mapa descargado.

Figura 28. Visualización del último mapa descargado en la memoria interna del dispositivo.

6.3 Autonomía

Finalmente, en este último apartado se busca hacer un juego de pruebas comprobando la autonomía del sistema a la hora de realizar todos los procesos de forma automática. Este es uno de los mayores puntos de la implementación ya que nos permite tener un sistema autónomo funcionando sin necesidad de interacción manual.

Para ello se comprueba que todos los procesos se inician en el orden correcto, a partir de las condiciones correctas y devolviendo los resultados esperados. Estas comprobaciones se realizarán desde la inserción de un nuevo documento en el Storage, *Figura 29*, pasando por el inicio del filtrado de datos y finalmente, la generación de mapas.

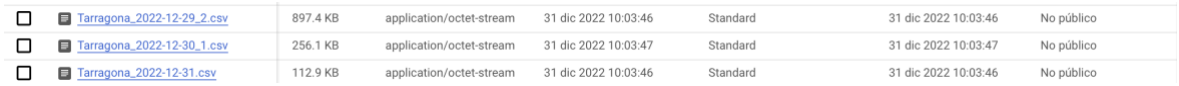


Figura 29. Inserción de ficheros con los datos obtenidos por los usuarios la noche anterior.

A continuación, se debe iniciar el filtrado de datos, mostrado en la *Figura 30*, como el proceso inicia y finaliza con éxito. Y tras esto, el resultado serán unos ficheros .csv completamente nuevos con los datos obtenidos, mostrados en la *Figura 31*:

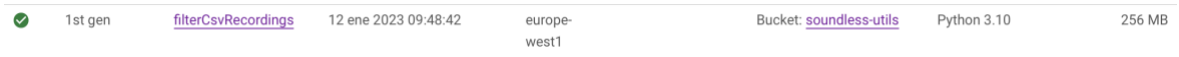


Figura 30. Inicio y finalización correcta del filtrado de datos.

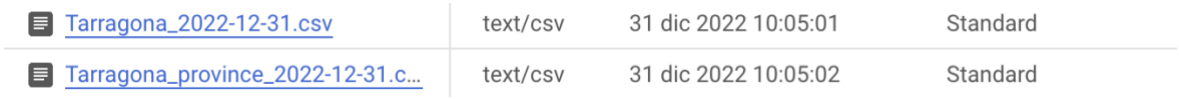


Figura 31. Creación de los ficheros filtrados.

Tras el filtrado de datos y la posterior creación de ficheros con los resultados obtenidos mostrados en las figuras anteriores, se inicia un proceso para la generación de los mapas de ruido a partir de estos mismos datos, correspondiente al proceso mostrado en la *Figura 32* y el resultado obtenido finalmente de la *Figura 33*:

Figura 32. Inicio y finalización correcta del generado de mapas.

Reus_2022-12-31_month.html	478.6 KB	text/html	31 dic 2022 12:01:02
Reus_2022-12-31_week.html	478.6 KB	text/html	31 dic 2022 12:01:02
Reus_2022-12-31_yesterday.html	478.6 KB	text/html	31 dic 2022 12:01:02
Tarragona_2022-12-31_month.html	2 MB	text/html	31 dic 2022 12:01:51
Tarragona_2022-12-31_week.html	1.3 MB	text/html	31 dic 2022 12:01:07
Tarragona_2022-12-31_yesterday....	1.2 MB	text/html	31 dic 2022 12:01:02
Tarragona_province_2022-12-31_...	5.3 MB	text/html	31 dic 2022 12:01:49
Tarragona_province_2022-12-31_...	4.6 MB	text/html	31 dic 2022 12:01:07
Tarragona_province_2022-12-31_...	4.5 MB	text/html	31 dic 2022 12:01:02

Figura 33. Creación de los mapas generados.

Tal y como se puede apreciar en este juego de pruebas exhaustivo, todos los resultados y procesos del sistema han concluido con éxito, demostrando que es un sistema escalable, fiable y autónomo.

7 Puesta en producción

Tras el costoso desarrollo de Soundless y sus diferentes componentes, finalmente el proyecto fue lanzado a producción, permitiendo el acceso a cualquier usuario externo a los desarrolladores.

Esta tarea verificó la escalabilidad, robustez y seguridad implementada en cada uno de los procesos que existen tanto en el sistema en la nube como en la aplicación móvil. Por ello, de forma previa, se realizó un exhaustivo proceso de ensayos y pruebas de integración de las diferentes partes, para tratar de lanzar el mejor producto posible.

A continuación de esta etapa de pruebas exhaustivas en diferentes dispositivos, desde diferentes lugares y utilizando herramientas especializadas para ello, se hizo un primer acercamiento al público. Este primer acercamiento, contó con el apoyo de distintas asociaciones de vecinos de los barrios con una mayor problemática de ruido de la ciudad de Tarragona.

Durante los contactos se les propuso a los usuarios obtener una de las pulseras de actividad compradas por el grupo de investigación a cambio de su participación activa en el proyecto durante los meses de noviembre y diciembre de 2022.

Al finalizar esta etapa de publicación y captación de usuarios se obtuvo un número considerable de usuarios para realizar las pruebas y obtención de datos oportunos. Desde el lanzamiento de la aplicación móvil en la *Google Play Store*[4] [13] el 26 de octubre de 2022, ha sido instalada en un total de 41 dispositivos móviles.

El resultado final del proyecto ha sido un conjunto de datos mucho más grande de lo previsto. Se han obtenido 4.320 horas de grabación, repartidas en 736 grabaciones diferentes, cada una conteniendo muestras de ruido y datos de salud registrados cada 10 segundos, generalmente durante una noche completa. Esta información ha permitido a los investigadores tener una comprensión más detallada y completa del ruido y la salud de los habitantes de Tarragona.

En cuanto a las incidencias detectadas, se obtuvieron en todo el periodo mencionado, un total de 10.887 incidencias relacionadas con el ruido, el ritmo cardíaco y el sueño. En particular, se identificaron 6.129 incidencias relacionadas con el ruido, 3.071 incidencias relacionadas con el ritmo cardíaco y 1.687 incidencias relacionadas con el sueño. Estos datos están representados en la *Figura 34* para obtener una visión global de los tipos de incidencias detectados.

4 Google Play Store: es una plataforma de distribución digital de aplicaciones móviles para los dispositivos con sistema operativo Android.

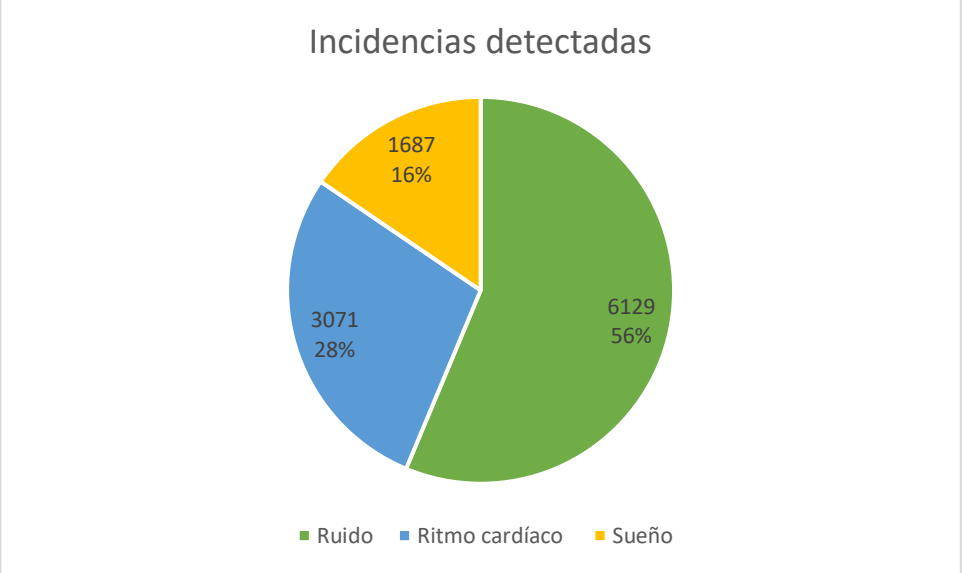


Figura 34. Gráfico sobre la distribución de las incidencias detectadas.

Junto a estos datos, se generaron, descargaron y obtuvieron un total de 549 mapas generados de las ciudades de Tarragona, Reus y la provincia de Tarragona. 9 mapas generados diariamente durante 61 días, de forma ininterrumpida, sin errores y de forma automática y autónoma.

Mostrando así la robustez del sistema y la fiabilidad de los datos obtenidos por el trabajo realizado en este proyecto.

7.1 Errores y soluciones

En cuanto a los errores detectados en esta fase de producción del proyecto fueron finalmente dos, los cuales ambos fueron relacionados con los diferentes tamaños de pantalla de los dispositivos Android.

En primera instancia nos encontramos con que los mapas mostrados en los dispositivos de los usuarios no se mostraban correctamente. Este problema era debido a los diferentes tamaños de pantalla de los dispositivos utilizados.

Al tener un tamaño de pantalla demasiado grande o pequeño, provocaba que los mapas se vieran desplazados y de forma incorrecta, dificultando así la utilización de esta funcionalidad en los dispositivos.

Para solucionar este error se modificó la función *renderWebView()*, mostrada en el *Código 9*, las modificaciones fueron con respecto al zoom y posicionamiento inicial del mapa. Se cuadró el mapa para que se mostrase en el centro de la pantalla.

A continuación, nos encontramos con más problemas relacionados con el tamaño de pantalla ya que, en pantallas de pequeño tamaño, los elementos no se mostraban en el orden correcto.

Para ello se realizaron algunas modificaciones en el archivo *XML* mostrado en el *Código 10*. Estas modificaciones permitían a los elementos recolocarse dinámicamente dependiendo del tamaño de pantalla del dispositivo, eliminando las posiciones fijas utilizadas anteriormente.

Tras estas modificaciones en el código, la aplicación pudo continuar durante el periodo de producción sin problemas ni errores detectados en lo visto en este trabajo de ingeniería informática.

8 Conclusiones

Tras la evolución y desarrollo de este trabajo podemos decir que ha cumplido con sus expectativas de forma satisfactoria. Este proyecto se presenta como la implementación de una funcionalidad desde 0 en un sistema Android-Cloud en sus fases de preproducción y producción.

En este trabajo se ha dado solución a una de las principales carencias que existían anteriormente, y es la información a tiempo real de las incidencias existentes hacia los usuarios. Se ha creado un sistema en la nube autónomo, robusto y escalable en todos los sentidos, capaz de detectar incidencias a partir de los datos de los usuarios y proporcionar mapas de ruido de estos mismos datos. También se ha implementado una interfaz y una lógica Android para la gestión y visualización de estos mapas.

A nivel personal, ha sido satisfactorio y de gran aprendizaje, el formar parte de un grupo de investigación como este y poder crear algo nuevo para ayudar a las personas en el mundo real. Este trabajo ha tenido muchos retos y dificultades, como la toma de decisiones, entender cómo funciona un proyecto en producción y el aprendizaje de múltiples servicios y herramientas totalmente nuevas.

Realizar este trabajo me ha hecho darme cuenta de lo que conlleva crear un proyecto informático y realizar toda una funcionalidad nueva desde cero. Todo ello es un proceso complejo, el cual requiere un análisis previo, mucho aprendizaje y trabajar muchas horas para que el resultado sea lo mejor posible.

Me gustaría seguir trabajando en este tipo de áreas que permitan a las personas poder dar solución a un problema existente a través de la tecnología y la informática. Esta ha sido la mayor satisfacción de este trabajo.

8.1 Trabajo futuro

El sistema implementado es perfectamente funcional e informático, pero se podrían incorporar algunas mejoras al diseño actual:

- Mejorar la visualización de la información mostrada en los mapas, haciéndolo aún más visual y intuitivo.
- Extraer información redundante de los mapas y precargarla en la aplicación móvil, ahorrando transferencia de datos entre servidor y cliente.
- Realizar un estudio más exhaustivo y con un mayor volumen de datos para mejorar la precisión en la detección de las distintas incidencias.
- Desarrollar el total de la aplicación móvil para el sistema operativo iOS y de esta forma poder abarcar a una mayor cantidad de usuarios.

9 Referencias

[1] World Health *Organization. Regional Office for Europe, Night noise guidelines for Europe*. WHO Regional Office Europe, 2009.

[2] «New evidence from WHO on health effects of traffic-related noise in Europe», *Who.int*. [En línea]. Disponible en: https://www.who.int/europe/news/item/30-03-2011-new-evidence-from-who-on-health-effects-of-traffic-related-noise-in-europe. [Accedido: 12-ene-2023].

[3] F. Richter, "Big three dominate the global cloud market", *Statista*, 26-jul-2019. [En línea]. Disponible en: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/. [Accedido: 30-mar-2023].

[4] R. Miller, "Even as cloud infrastructure market growth slows, Microsoft continues to gain on Amazon," *TechCrunch*, 06-Feb-2023.

[5] «Overview», *Docker Documentation*, 12-may-2023. [En línea]. Disponible en: https://docs.docker.com/get-started/. [Accedido: 01-abr-2023].

[6] «Realiza previsiones con ARIMA+», *Google Cloud*. [En línea]. Disponible en: https://cloud.google.com/vertex-ai/docs/tabular-data/forecasting-arima/overview?hl=es-419. [Accedido: 10-abr-2023].

[7] Wikipedia contributors, «Standard score», *Wikipedia, The Free Encyclopedia*, 29-mar-2023. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Standard_score&oldid=1147224272.

[8] E. Suni, *Stages of sleep: What happens in a sleep cycle | sleep foundation*. Sleep Foundation, 2021.

[9] «Sleep phases and stages», *NHLBI, NIH*. [En línea]. Disponible en: https://www.nhlbi.nih.gov/health/sleep/stages-of-sleep. [Accedido: 14-abr-2023].

[10] B. Van de Ven, "Bokeh", *Bokeh.org*. [En línea]. Disponible en: https://bokeh.org. [Accedido: 14-mar-2023].

[11] "GeoPandas 0.7.0 — GeoPandas 0.7.0 documentation," *Geopandas.org*. [En línea]. Disponible en: https://geopandas.org/. [Accedido: 18-abr-2023].

[12] Wikipedia contributors, «k-means clustering», *Wikipedia, The Free Encyclopedia*, 22-feb-2019. [En línea]. Disponible en: https://en.wikipedia.org/wiki/K-means_clustering.

[13] "Soundless - Apps on Google Play," *Play.google.com*. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=cat.urv.cloudlab.soundless. [Accedido: 30-abr-2023].