UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL, MEDIANTE APP MÓVIL, PARA REDES DE SENSORES ACÚSTICOS EN IOT

TRABAJO FIN DE GRADO

Febrero -2024

AUTOR: Adrián Sarrías Pérez

DIRECTOR/ES: Miguel Onofre Martínez Rach
Otoniel Mario López Granado

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En esta sección se expondrá la motivación que ha impulsado el desarrollo de este proyecto, además se proporcionará una visión general del proyecto desarrollado y los objetivos principales establecidos en las etapas iniciales del mismo.

1.1 MOTIVACIÓN

La dependencia de la tecnología continúa creciendo a medida que más empresas utilizan la tecnología en la gestión, procesos de fabricación, control de calidad, mantenimiento y otros aspectos industriales. La gestión de múltiples recursos y procesos empresariales les permiten a las empresas maximizar los ingresos y mejorar su capacidad de respuesta a eventos complejos mientras se reducen los costos asociados a estos eventos.

El propósito del desarrollo de este proyecto ha sido explorar un tema que cada vez está más integrado en la sociedad: la contaminación acústica. Además, es necesario entender cómo gestionar y mantener estos sistemas, como recopilan información en tiempo real y como intervenir en los procesos para mejorarlos desde diferentes perspectivas, como eficiencia energética y seguridad. Estos son, entre otros, algunos de los motivos que han impulsado el desarrollo de este proyecto.

1.2 OBJETIVOS

El objetivo principal de este proyecto es el diseño y desarrollo de una aplicación móvil, así como toda la infraestructura adicional necesaria que muestre la información que recogen un conjunto de IoTs. Para realizar este proyecto, hay que solucionar el complejo desarrollo sobre las comunicaciones entre la aplicación móvil, el servidor y el conjunto de IoTs.

El proyecto desarrollado abarca el diseño como una solución completa, capaz de atender a múltiples clientes y sistemas. Esto incluye la capacidad de gestionar, de manera eficiente, una gran cantidad de conexiones simultaneas para garantizar una experiencia de usuario óptima.

También es crucial que el sistema sea configurable y adaptable a las necesidades del entorno en el que se despliega. Este sistema debe permitir la gestión por parte de los usuarios, debe ser capaz de monitorear cada dispositivo IoT conectado a él, permitir la realización de acciones de forma manual y proporcionar información sobre su estado actual.

Para alcanzar los objetivos establecidos en el desarrollo de este proyecto, ha sido necesario llevar a cabo una serie de tareas, entre las cuales se incluyen:

- Estudio de las tecnologías existentes para el desarrollo del backend: Se requiere evaluar las tecnologías existentes que proporcionen la funcionalidad necesaria para este proyecto. Esto incluye tecnologías que utilizan los protocolos de comunicación mencionados.
- Diseño e implementación de cada subsistema tanto en el frontend como en el backend: Una vez seleccionada las tecnologías a utilizar, se debe llevar a cabo el diseño de cada subsistema, tanto frontend como backend, y posteriormente implementarlas empleando las tecnologías elegidas.

Aparte de las tareas necesarias para completar todo el proyecto, existen ciertos requisitos que se deben cumplir. En primer lugar, los usuarios deben poder autenticarse utilizando sus credenciales para acceder al contenido del sistema IoT. Además, el sistema debe ser capaz de devolver y mostrar de manera eficiente y efectiva la lista de contenidos solicitados por el usuario. Estos requisitos son esenciales para garantizar una experiencia de usuario satisfactoria y el correcto funcionamiento del sistema en su conjunto

1.3 JUSTIFICACIÓN DEL PROYECTO

Para poder comprender el proceso necesario para el monitoreo de la contaminación acústica, debemos entender algunos conceptos en los que se basa este proyecto.

La contaminación acústica es ampliamente considerada por la mayoría de la población de las grandes ciudades como un factor medioambiental muy significativo que afecta principalmente a la calidad de vida de las personas. Esta forma de contaminación, también conocida como ruido ambiental, es una consecuencia no deseada proveniente de las actividades que se llevan a cabo en entornos urbanos densamente poblados.

El término contaminación acústica se refiere al ruido cuando este se considera un contaminante, es decir, un sonido molesto que puede causar efectos fisiológicos y psicológicos perjudiciales para una persona. La actividad humana, incluyendo el transporte, la construcción de edificios y obras públicas, así como la industria, es la principal causa de la contaminación acústica. Los efectos del ruido pueden manifestarse en pérdida de audición y en irritabilidad, entre otros efectos tanto fisiológicos como psicológicos [1].

El monitoreo continuo de ruido permite evaluar el nivel acústico que puede soportar una ciudad, obteniendo datos exactos y reales se pueden llevar a cabo medidas para reducir los niveles de ruido que soportan las personas [2].

En la figura 1, se muestran las fuentes principales de ruido en entornos urbanos, cabe destacar que la principal fuente es el ruido ocasionado por los turismos, seguido de las motocicletas y vehículos pesados.

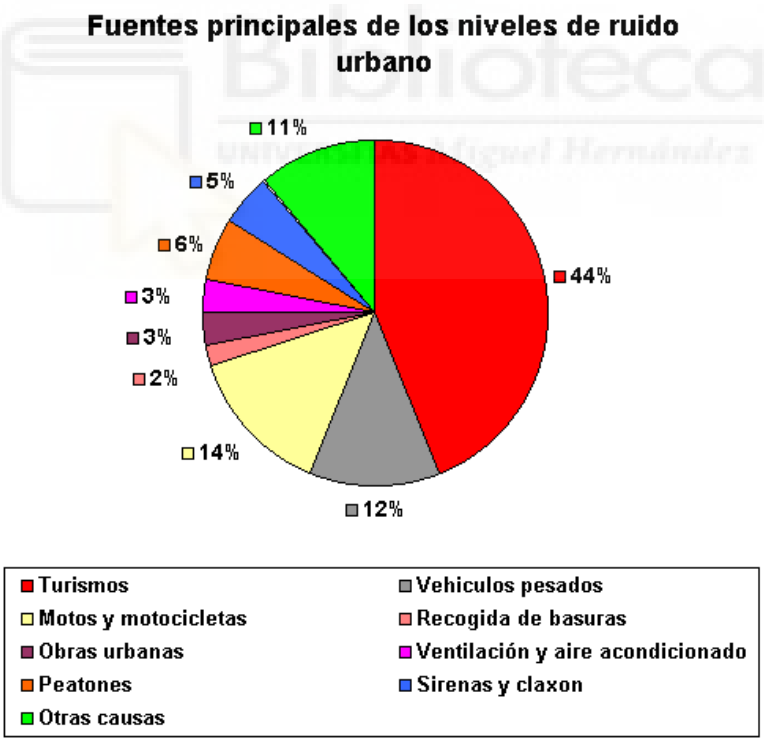

Figura 1. Fuentes principales de los niveles de ruido urbano. Fuente:Monografías [24].

2. IoT

2.1 ¿QUÉ ES IOT?

El Internet de las cosas (IoT) es el proceso que permite conectar los elementos o dispositivos físicos cotidianos a Internet: desde objetos domésticos comunes, como las bombillas o electrodomésticos, hasta recursos para la atención de la salud e incluso conectar los sistemas de las ciudades inteligentes.

Los dispositivos IoT que se encuentran dentro de estos objetos suelen pertenecer a una de estas dos categorías: son actuadores (es decir, envían instrucciones a un objeto) o son sensores (recopilan datos y los envían a otro lugar).

Los sistemas IoT tradicionales funciona de tal manera que envían, reciben y analizan los datos de forma permanente en un ciclo de retroalimentación. Según el dispositivo IoT, las personas o los sistemas de inteligencia artificial y aprendizaje automático (IA/ML) pueden analizar estos datos casi de inmediato o durante cierto tiempo [3]. En la figura 2, se muestran una serie de dispositivos cotidianos, que son IoTs.

Figura 2. IoTs. Fuente:Tecnoseguro [25].

2.2 HISTORIA Y EVOLUCIÓN DEL INTERNET DE LAS COSAS

Los primeros conceptos sobre la creación de una red de dispositivos inteligentes se discutieron en 1982, cuando una máquina de Coca-Cola modificada se convirtió en el primer electrodoméstico conectado a Internet. Durante la década de los 90, diversos artículos académicos fueron publicados sobre este tema, y el término "Internet de las cosas" se popularizó en 1999.

A partir de ese momento, el concepto de Internet de las cosas y su evolución estuvieron relacionados con la incorporación de sensores y conexiones en cualquier dispositivo que pudiera admitirlo. Básicamente se trataba de agregar "inteligencia" a diferentes objetos electrónicos que tuvieran esa capacidad.

Desde 2010, la evolución del Internet de las Cosas ha sido exponencial y se espera que el crecimiento continúe en los próximos años. En este sentido, el último avance en las tecnologías de conectividad para el IoT ha sido el 5G, una conexión inalámbrica significativamente más rápida que sus predecesores [4].

2.3 FUTURO DE LOS IoTs

A medida que avanza la tecnología, es probable que el Internet de las Cosas continúe creciendo y evolucionando. Con el surgimiento de la tecnología 5G, la capacidad de transmitir datos de manera rápida y confiable se expandirá aún más, permitiendo una mayor conectividad y aplicaciones más avanzadas. Además, la convergencia del IoT con otras tecnologías como la inteligencia artificial o el *big data* promete abrir nuevas posibilidades en áreas como la atención médica, la logística y la gestión de recursos naturales [5].

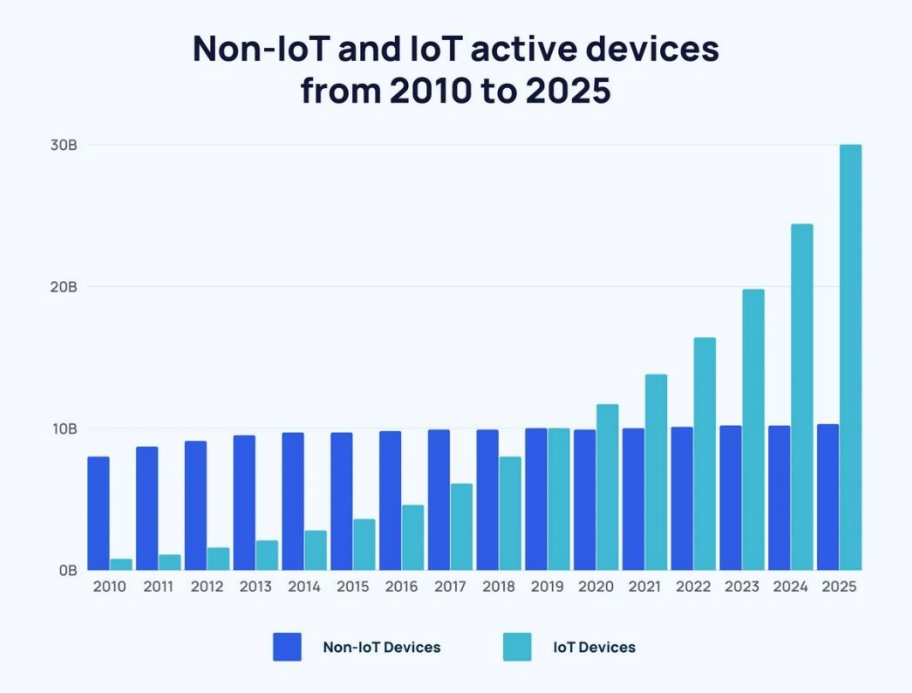

Figura 3. Comparación entre los dispositivos Iot y los que no son IoT. Fuente: ExplodingTopics [26].

Ha habido un cambio notable de dispositivos que no son IoT a dispositivos IoT durante la última década, como se ve en la figura 3. De hecho, se estima que para 2030, el 75% de todos los dispositivos serán IoT.

3. TECNOLOGÍAS

En esta sección se describirán brevemente las distintas tecnologías, servicios y aplicaciones fundamentales en el diseño y desarrollo de este proyecto.

3.1TECNOLOGÍAS APLICADAS EN EL LADO DEL CLIENTE

La parte correspondiente a la interfaz de usuario del sistema está compuesta por la aplicación Android y cada una de sus tecnologías asociadas. Esta parte permite al usuario interactuar a través de diversas pantallas con los distintos componentes, como botones, texto y listas. A través de esta interacción, el cliente puede comunicarse con el servidor, que a su vez se comunica con el IoT. Es fundamental que estas tecnologías brinden soporte para enviar datos al IoT y reflejen cualquier cambio de estado en la aplicación cuando el IoT experimenta un cambio manual.

Es importante destacar que algunas de las tecnologías del lado del cliente son innovadoras y relativamente nuevas en la comunidad, pero se espera que jueguen un papel fundamental en el futuro del desarrollo de aplicaciones Android.

3.1.1 Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 durante la conferencia Google I/O, y se presentó como el sucesor de Eclipse, que era el IDE oficial utilizado para el desarrollo de aplicaciones para Android. La primera versión estable de Android Studio fue publicada en diciembre de 2014.

Este IDE está basado en el software *IntelliJ IDEA* de *JetBrains* y se ha distribuido de forma gratuita bajo la Licencia Apache 2.0. Es compatible con varias plataformas, incluyendo GNU/Linux, macOS, Microsoft Windows y Chrome OS, y ha sido diseñado específicamente para el desarrollo de aplicaciones Android.

Desde mayo de 2019, Kotlin se ha convertido en el lenguaje preferido de Google para el desarrollo de aplicaciones de Android. Sin embargo, Android Studio también admite otros lenguajes de programación, como Java y C++ [6].

Android Studio fue toda una revolución, puesto que con Eclipse se tenía que lidiar con ciertas desventajas como la complicada actualización del SDK, problemas para reconocer librerías y dificultad a la hora de crear una interfaz de usuario. Mientras que con Android Studio no existían todos estos problemas [7].

Este entorno de desarrollo presenta ciertas características que hacen que el desarrollo de aplicaciones sea más sencillo [8]:

– *Apply Changes*: Esta funcionalidad permite realizar cambios en el código y en los recursos de la aplicación en ejecución sin necesidad de reiniciarla, y en determinados casos, sin reiniciar la actividad actual. Esta flexibilidad permite revisar que partes de la aplicación se reinician cuando se implementa pequeños cambios, al mismo tiempo que se preserva el estado actual del dispositivo. En la figura 4, se muestra donde se encuentra esta funcionalidad en Android Studio

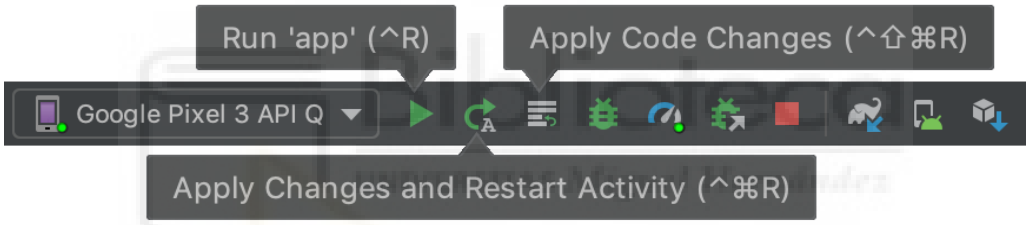

Figura 4. Apply Changes en Android Studio. Fuente: Android Studio [27].

– Editor de código inteligente**:** este editor de código permite escribir código de forma más eficiente, por lo que se trabaja con mayor rapidez siendo así más productivo. Con funciones avanzadas de completado, refactorización y análisis de código, se mejora el flujo de trabajo. Además, Android Studio ofrece sugerencias contextuales a medida que se escribe, presentadas en una lista desplegable para facilitar la selección. En la figura 5, se muestra un ejemplo de las sugerencias que ofrece este editor.

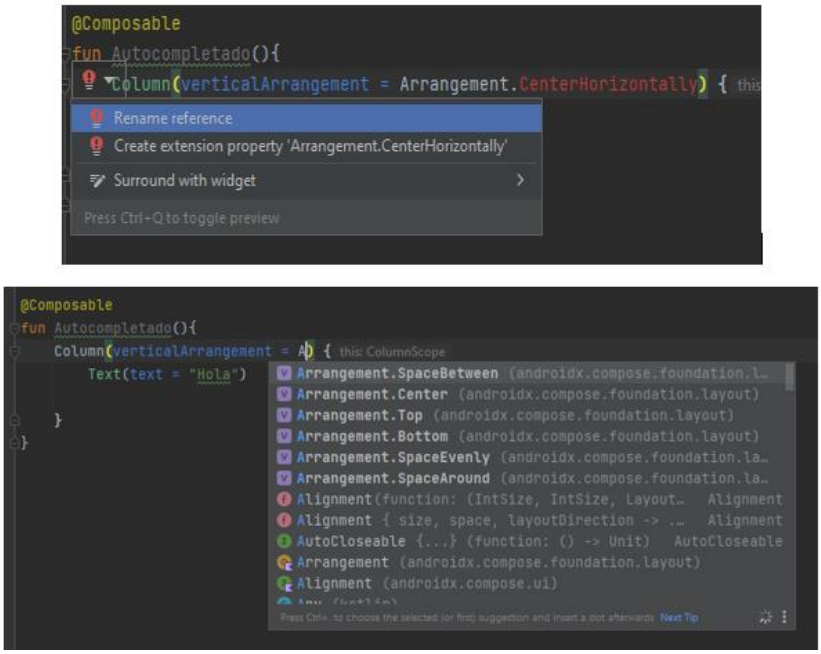

Figura 5. Autocompletado del código y sugerencias.

– Emulador Android: el emulador se instala e inicia tus aplicaciones de manera más rápida que un dispositivo real. Además, proporciona la capacidad de crear un prototipo de la aplicación y probarla en una variedad de dispositivos Android, incluyendo teléfonos, tablets, smartwatches y Android TV. Esto asegura el correcto funcionamiento de la aplicación en diferentes dispositivos. En la figura 6, se muestra el emulador utilizado en Android Studio para el desarrollo de la aplicación

Figura 6. Emulador en Android Studio.

3.1.2 Kotlin

Kotlin es un lenguaje de programación de código abierto creado por JetBrains que ha ganado bastante popularidad, especialmente debido a su capacidad para desarrollar aplicaciones Android. Aunque Kotlin puede utilizarse para una variedad de propósitos de programación, su integración fluida con las herramientas de desarrollo de Android y su sintaxis concisa lo hacen una opción atractiva para los desarrolladores de aplicaciones móviles [9].

Entre sus principales características, se destacan:

- Su curva de aprendizaje es sencilla, la sintaxis que presenta este lenguaje permite un aprendizaje fluido, intuitivo y fácil de usar. Al ser un lenguaje de código abierto, hay mucho apoyo de la comunidad de programadores de Kotlin, lo que supone una gran ventaja.
- El paradigma de programación que presenta Kotlin puede ser tanto Orientado a Objetos (POO) como funcional. Una ventaja significativa de Kotlin, es trabajar con *lambdas*, una característica de lenguajes avanzados que permite encapsular código en variables que pueden ser usados como parámetros de funciones, y que junto con una sintaxis optimizada y reducida permiten optimizar las tareas más comunes y tediosas en el desarrollo de una aplicación.
- Kotlin es capaz de interoperar completamente con la sintaxis del lenguaje Java. Esto significa que, dado un código escrito en Java, Kotlin puede interactuar correctamente, y viceversa. Esta interoperabilidad permite a los desarrolladores aprovechar el ecosistema de Java y migrar gradualmente a Kotlin sin necesidad de reescribir todo el código desde cero.
- En Kotlin, las corrutinas optimizan la programación asíncrona. Simplifican así el trabajo de las llamadas al API y acceso a las bases de datos.
- Kotlin tiene la capacidad de reducir el tiempo de programación al eliminar el código redundante. Este lenguaje es conocido por su sintaxis compacta y concisa, lo que facilita significativamente el proceso de escritura de código y ayuda a evitar la repetición innecesaria de instrucciones.

Es cierto que Kotlin ha ido ganado una gran popularidad en el desarrollo de aplicaciones Android, principalmente debido a sus ventajas en términos de productividad, interoperabilidad con Java y su sintaxis concisa. A medida que más desarrolladores eligen Kotlin como lenguaje para el desarrollo de sus proyectos de Android, es posible que Java se vuelva menos predominante en este contexto. Además, se están desarrollando tecnologías relacionadas con la interfaz de usuario que están apostando totalmente por Kotlin, lo que podría acelerar aún más su adopción y reducir la dependencia de Java en el desarrollo de aplicaciones Android. Sin embargo, es importante tener en cuenta que Java aún seguirá siendo relevante en muchos otros ámbitos fuera del desarrollo de aplicaciones móviles.

3.1.3 Jetpack Compose

Jetpack Compose surgió en julio de 2021 con una versión estable y pública, marcando un antes y un después en el desarrollo de aplicaciones nativas en Kotlin. Este *framework* ofrece una ventaja significativa al permitir la visualización en tiempo real de vistas previas del interfaz de usuario, lo que añade valor al momento de presentar y vender una aplicación. Con herramientas precisas y actualizadas que se adaptan a las necesidades actuales, Jetpack Compose se destaca como una opción atractiva para los desarrolladores, ofreciendo una experiencia de desarrollo más fluida y eficiente. Esto lo convierte en una de las principales opciones para la creación de interfaces de usuario modernas y atractivas.

Desde un punto de vista comparativo, la creación de interfaces de usuario (UI) con Jetpack Compose se asemeja a la metodología utilizada en *React Native*. Ambos *frameworks* hacen uso de componentes reutilizables para minimizar la cantidad de código necesario y evitar así la repetición. Sin embargo, es importante destacar que en el caso de Jetpack Compose, esta compilación de componentes se logra gracias a los *plugins* de compiladores de Kotlin. Estos *plugins* permiten ejecutar los componentes mediante la estructura de archivos de Kotlin, lo que contribuye a una reducción significativa de código y a una mayor eficiencia en el desarrollo de la interfaz de usuario [11].

Para comprender mejor esta herramienta, es fundamental comenzar por entender los dos paradigmas básicos que existen para construir la parte gráfica de las aplicaciones: el paradigma imperativo y el paradigma declarativo [12].

En una interfaz de usuario desarrollada con el paradigma imperativo, se tiene que especificar a cada componente cómo debe comportarse en respuesta a cada evento o interacción que ocurra, con el objetivo de modificar su estado interno. En otras palabras, construir una interfaz de usuario mediante programación imperativa implica indicar cómo debe comportarse en cada momento, con cada evento que ocurra.

Sin embargo, en un enfoque declarativo, los componentes no mantienen un estado interno, sino que cada vez que se actualiza el estado de la aplicación en respuesta a eventos o interacciones, los componentes se reconstruyen según sea necesario. Es decir, construir una interfaz de usuario mediante programación declarativa implica indicar qué se debe mostrar en función del estado actual de la aplicación, en lugar de como se debe modificar cada componente en respuesta a eventos específicos. En la figura 7, se muestra un esquema de la migración de utilizar *Views* a utilizar *Compose*, siendo esta última la mas utilizada actualmente

Jetpack Compose ofrece varios beneficios clave en comparación con el desarrollo de interfaces de usuario mediante XML y el paradigma imperativo:

- Menos código: Jetpack Compose destaca por necesitar menos líneas de código para lograr el mismo resultado que con la programación imperativa. Esta eficiencia se traduce en una clara ventaja en términos de optimización del trabajo.
- Mas intuitivo: Dado que Jetpack Compose sigue un paradigma declarativo, donde simplemente se describe la interfaz de usuario que se va a mostrar, se considera que el código es mucho más legible y comprensible para los desarrolladores.
- Acelera el desarrollo: Jetpack Compose acelera el desarrollo al ofrecer compatibilidad con el código existente en XML. Esto significa que puedes llamar al código de Compose desde las *Views* tradicionales y viceversa.
- Composable: Jetpack Compose se basa en el concepto de “composable”, lo que significa que está diseñado para construir interfaces de usuario mediante la combinación de pequeños bloques de código reutilizables e independientes. A diferencia del enfoque tradicional basado en *Fragments* o *Activities*.

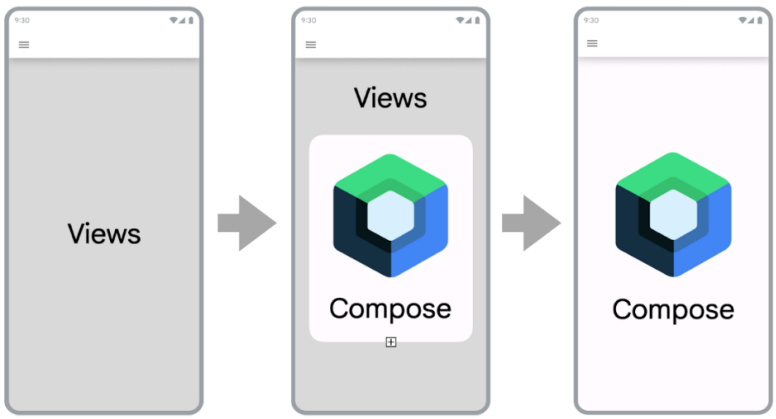

Figura 7. Migración de Views a Compose. Fuente: Android Studio [28].

3.1.4 Retrofit

Retrofit 2 es un cliente REST creado por Square para Android y Java que tiene como objetivo simplificar la interacción con servicios web *RESTful*. Utiliza *OkHttp* como su capa de administración de sistemas, aprovechando su robustez y eficiencia. Retrofit 2 serializa naturalmente las respuestas JSON utilizando objetos POJO (*Plain Old Java Objects*) que deben estar estructurados según el formato JSON correspondiente. Para llevar a cabo esta serialización, se requiere un convertidor que convierta el JSON a *Gson*.

En comparación con otras bibliotecas, Retrofit es notablemente más simple, ya que no requiere el análisis manual del JSON; en su lugar, devuelve los objetos directamente. Sin embargo, una limitación importante es que no proporciona soporte integrado para cargar imágenes desde el servidor. Para superar esta limitación, los desarrolladores pueden optar por utilizar bibliotecas adicionales como Picasso. En la figura 8, se muestra como es el funcionamiento de Retrofit, utilizando peticiones Http.

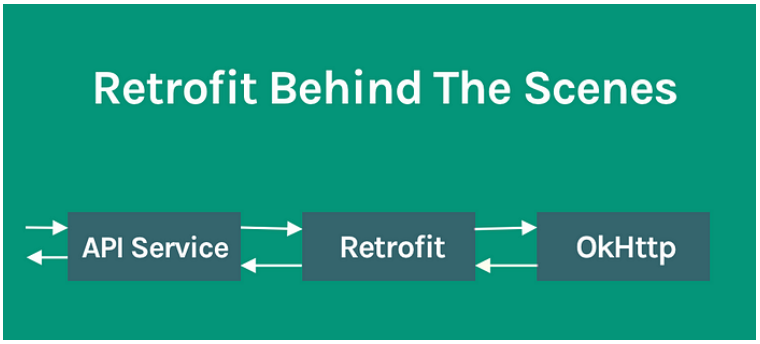

Figura 8. Funcionamiento de Retrofit. Fuente: Medium[29].

3.2 TECNOLOGÍAS DEL LADO DEL SERVIDOR

El servidor es la parte de la estructura que hace de intermediario entre los dispositivos IoT y la aplicación móvil, tiene que ser capaz de gestionar todas las peticiones y la base de datos de forma eficiente.

3.2.1 Laravel

Laravel es considerado por muchos desarrolladores el *framework* predeterminado para crear sitios web y aplicaciones basadas en PHP. Se ha convertido en uno de los *frameworks* más populares para el desarrollo de aplicaciones web, gracias a su sintaxis elegante y funcional, sus amplias funcionalidades y su enfoque más accesible para los desarrolladores [13].

Laravel proporciona a los desarrolladores un conjunto de herramientas para crear una amplia variedad de aplicaciones web, desde simples páginas hasta plataformas empresariales más complejas, de manera rápida y eficiente. Su objetivo principal es simplificar el proceso de desarrollo ofreciendo una sintaxis expresiva, una arquitectura modular y una amplia gama de funcionalidades integradas.

En el ecosistema de Laravel, Composer y Artisan juegan roles fundamentales. Laravel, se beneficia enormemente de estas dos herramientas.

- Composer [14] es un administrador de dependencias para el lenguaje de programación PHP, que administra el software de dependencias y las bibliotecas requeridas. Composer se ejecuta a través de la línea de comando. Su objetivo principal es instalar las dependencias o bibliotecas necesarias para una aplicación. Además, permite a los usuarios instalar las aplicaciones PHP disponibles en *Packagist*, donde *Packagist* es el repositorio principal que contiene todos los paquetes disponibles.
- Artisan [15] es el nombre de la interfaz de línea de comandos incluida con Laravel. Proporciona una serie de comandos útiles para su uso mientras desarrolla su aplicación. Está impulsado por el potente componente *Symfony Console*. Cada comando también incluye una pantalla de "ayuda" que muestra y describe los argumentos y opciones disponibles del comando.

3.2.2 PHP

El término PHP [16] es un acrónimo de *Hypertext Preprocessor*. PHP es un lenguaje de programación del lado del servidor diseñado específicamente para el desarrollo web. Es de código abierto, lo que significa que se puede descargar y utilizar de forma gratuita. Se trata de un lenguaje interpretado que no requiere un compilador.

Algunas características significativas que presenta este lenguaje son:

- El código PHP se ejecuta en el servidor.
- Se puede integrar con muchas bases de datos como Oracle, Microsoft SQL Server, MySQL, PostgreSQL, Sybase e Informix.
- Admite protocolos como HTTP Basic, HTTP Digest, IMAP, FTP y otros.
- PHP presenta una gran facilidad a la hora de integrarse en archivos, permitiendo también la escritura de código HTML dentro de un archivo PHP.
- Lo que distingue a PHP de los lenguajes del lado del cliente como HTML, es que los códigos PHP se ejecutan en el servidor, mientras que los códigos HTML se representan directamente en el navegador.
- Los códigos PHP se procesan primero en el servidor y luego se envía el resultado al navegador. La única información que el cliente o navegador conoce es el resultado devuelto después de ejecutar el código PHP en el servidor, y no el código real presente en el archivo PHP. Además, los archivos PHP pueden incluir otros lenguajes de programación del lado del cliente, como CSS y JavaScript.

3.2.3 PostgreSQL

PostgreSQL [17] es una base de datos gratuita y de código abierto que originalmente se conocía como Postgre. Está desarrollada y mantenida por el Grupo de Desarrollo Global PostgreSQL, que se centra en voluntarios. No se requiere ninguna tarifa de licencia, y los usuarios pueden utilizarla y modificarla según sus requisitos. PostgreSQL ha ganado una gran popularidad y actualmente es el cuarto motor de base de datos más utilizado. Está disponible para casi todas las distribuciones de Linux y otros sistemas operativos como Windows y macOS.

Esta base de datos es compatible con consultas relacionales a través del lenguaje de consulta estructurado (SQL) y también con consultas JSON no relacionales. Esta particularidad permite manejar una amplia gama de tipos de datos que van más allá de las simples cadenas y números. Por ejemplo, PostgreSQL es adecuado para almacenar medios como imágenes, audio y vídeo. Al operar en el modelo relacional, almacena datos en tablas, filas y columnas, lo que facilita a los usuarios la transición desde otros sistemas de base de datos hacia una arquitectura relacional de objetos. PostgreSQL también admite la herencia, donde las tablas pueden heredar propiedades de una tabla principal. Los usuarios pueden definir sus propios tipos de datos y funciones sin necesidad de modificar el código base.

Cuando hablamos de bases de datos, podemos diferenciar tipos de sistemas de gestión de bases de datos que se utilizan en el contexto de bases de datos relacionales: RDBMS *(Relational Database Management System*) y ORDBMS (*Object-Relational Database Management System*)

La principal diferencia es que RDBMS se refiere a un sistema de gestión de bases de datos que se basa en un modelo relacional para almacenar y gestionar datos. En un RDBMS, los datos se organizan en tablas relacionadas entre sí, donde cada tabla tiene filas y columnas, y las relaciones entre las tablas se establecen mediante claves primarias y externas. Por otro lado, ORDBMS es una extensión del concepto de RDBMS que combina características de los sistemas de bases de datos relacionales y los sistemas de bases de datos orientados a objetos. En un ORDBMS, se pueden definir tipos de datos y operaciones basadas en objetos, lo que permite representar de manera más eficiente las complejas relaciones entre tablas.

3.2.4 PgAdmin

PgAdmin4 [18] destaca como la herramienta GUI (*Graphical User Interface*) de código abierto más avanzada diseñada para la gestión de bases de datos relacionales. En conjunto con PostgreSQL, ofrece una combinación poderosa para la administración de bases de datos.

Algunas características de esta herramienta son:

- Funciona en todos los sistemas operativos ya que es compatible con Windows, Mac y Linux.
- Es compatible con todas las versiones de PostgreSQL y EDB *Postgres Advanced Server*.
- Existe una amplia gama de documentación en la que se detalla su instalación y sus diferentes usos.
- Se puede implementar en modo escritorio o modo servidor según las necesidades de la base de datos
- Cuentas con herramientas para realizar copias de seguridad, restaurar, aspirar y analizar.
- Permite la visualización, creación y edición de todos los objetos estándar en PostgreSQL.

4. DISEÑO

Antes de comenzar con el desarrollo del proyecto es necesario llevar a cabo la fase de diseño, en la que es necesario definir cada parte que participará en el proyecto final. Además, se debe definir la funcionalidad de cada parte, la jerarquía entre las secciones y el flujo de datos que debe cumplirse para poder realizar una correcta implementación posteriormente.

En esta fase es crucial tener en cuenta los requisitos principales, ya que esto permitirá determinar si son viables y compatibles entre sí, o si podrían surgir incongruencias y contradicciones no detectadas anteriormente.

Para realizar el diseño completo, se ha optado por dividir el proyecto en 2 grandes grupos diferenciados:

- Diseño del backend: Esta sección abarca toda la funcionalidad que debe ser implementada y ejecutada en un servidor, el cual brindará servicio a los dispositivos que funcionen como clientes y a aquellos dispositivos que consideremos IoTs.
- Diseño del frontend: En este apartado se tomarán en cuenta las especificaciones para el desarrollo de la aplicación móvil, la cual permitirá a los usuarios finales hacer uso del servicio.

4.1DISEÑO DEL BACKEND

Cuando se habla del diseño del backend, se refiere a la arquitectura y la lógica de programación que se implementará en el servidor para satisfacer las necesidades de los usuarios y dispositivos IoT.

El diseño del sistema backend se puede fragmentar en las siguientes partes:

- Diseño de la base de datos: Para gestionar toda la información generada por las comunicaciones y otros eventos, como los datos captados por los dispositivos IoT, es fundamental almacenar esta información de forma persistente. Para realizar esta tarea de manera eficiente, es necesario hacer uso de una base de datos. Esta

se encargará de estructurar la información y almacenarla para su posterior consulta y manipulación.

– Diseño del API RESTful: La mayor parte de la lógica del servicio final recae en el backend, y principalmente en el API. Esta parte se encarga de gestionar las comunicaciones que se realicen mediante llamadas HTTP. Además, es capaz de interconectar tanto el backend como el frontend, entre sí, ya sea de manera directa o indirecta.

La arquitectura del servidor estará formada principalmente por el *framework* PHP Laravel y la base de datos de PostgreSQL. Laravel se dedicará principalmente a gestionar las conexiones, mediante el uso de la API para las autentificaciones de usuario o para consultar datos, por lo cual tiene que estar conectado a la base de datos y la base de datos contendrá toda la información necesaria para el desarrollo del proyecto. En la figura 9 se muestran un esquema de la arquitectura usada en este proyecto.

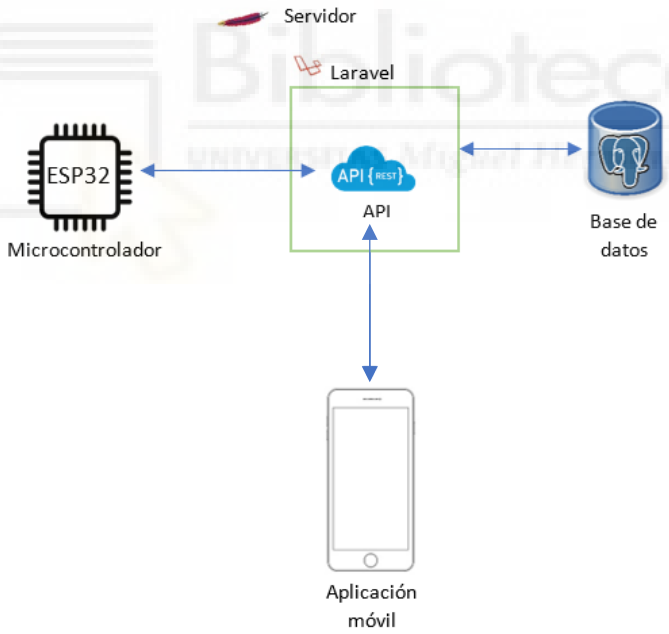

Figura 9. Arquitectura del proyecto.

4.1.1 Base de datos

Todos los datos con los que trabaja el sistema desarrollado se almacenan en una base de datos. Para ello, ha sido necesario diseñar un modelo entidad-relación que especifique

cada entidad participante en el proyecto, así como su interacción entre ellas. En la Figura 10 el modelo de la base de datos que se ha usado en este proyecto.

Figura 10. Modelo de la base de datos del proyecto.

La base de datos consta de 44 tablas, siendo cada una de ellas una entidad, o producto de ciertas relaciones. Todas las tablas poseen un identificador autoincremental, es decir, que su valor será establecido automáticamente por el gestor de base de datos, y su uso es únicamente interno, así se permite una mejor gestión y manejo de cada registro y se facilita la diferenciación entre ellos.

Para la creación de la base de datos se han utilizado las migraciones de Laravel, estos archivos contienen instrucciones para modificar la estructura de la base de datos, ya sea para crear, actualizar, añadir o crear tablas y columnas. Una vez que se han definido las migraciones, se pueden ejecutar mediante el uso de comandos de *Artisan*, la interfaz de línea de comandos incluida en Laravel.

4.1.2 API RESTful

El término API [19] es una abreviatura de *Application Programming Interfaces*, que en español significa interfaz de programación de aplicaciones. Se trata de un conjunto de funciones y procedimientos que se utilizan para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones a través de un conjunto de reglas establecidas.

La API desarrollada es de tipo RESTFUL, basado en la arquitectura *REpresentational State Transfer* (REST). Este tipo de APIs emplean *JavaScript Object Notation* (JSON) como formato de texto para realizar las comunicaciones y el intercambio de mensajes.

ARQUITECTURA REST

La transferencia de estado representacional (REST) [20] es una arquitectura de desarrollo que puede ser utilizada en cualquier cliente HTTP. Además, es mucho más simple que otras arquitecturas ya existentes, como pueden ser *XML-RPC* o *SOAP*. Esta arquitectura impone condiciones sobre cómo debe funcionar una API y es posible utilizar esta arquitectura para admitir comunicaciones confiables y de alto rendimiento.

A continuación, se presentan algunos de los principios que presenta la arquitectura REST:

- Es una tecnología sin estado, debido a que no se guarda la información en el servidor. Es decir, toda la información es enviada por el cliente en cada mensaje

HTTP, consiguiendo un ahorro en variables de sesión y almacenamiento interno del servidor.

- Presenta un conjunto de métodos bien definidos, siendo los más comunes *GET*, *POST*, *PUT* y *DELETE*, que se emplea en todos los recursos.
- Esta arquitectura admite el almacenamiento en caché, que es el proceso de almacenar algunas respuestas en la memoria caché del cliente o de un intermediario para mejorar el tiempo de respuesta del servidor
- Emplea *hipermedios* para representar la información, los más comunes son HTML, XML o JSON.

En la Figura 11 se observa cómo sería la arquitectura del servidor. Donde un usuario realiza una petición usando algún método de los antes mencionados, esta petición se direcciona al controlador correspondiente según venga establecido en la hoja de rutas del servidor. El controlador será el encargado de buscar la información en la base de datos usando el modelo necesario, y una vez que ya se haya obtenido la información, se la devolverá al usuario en formato JSON.

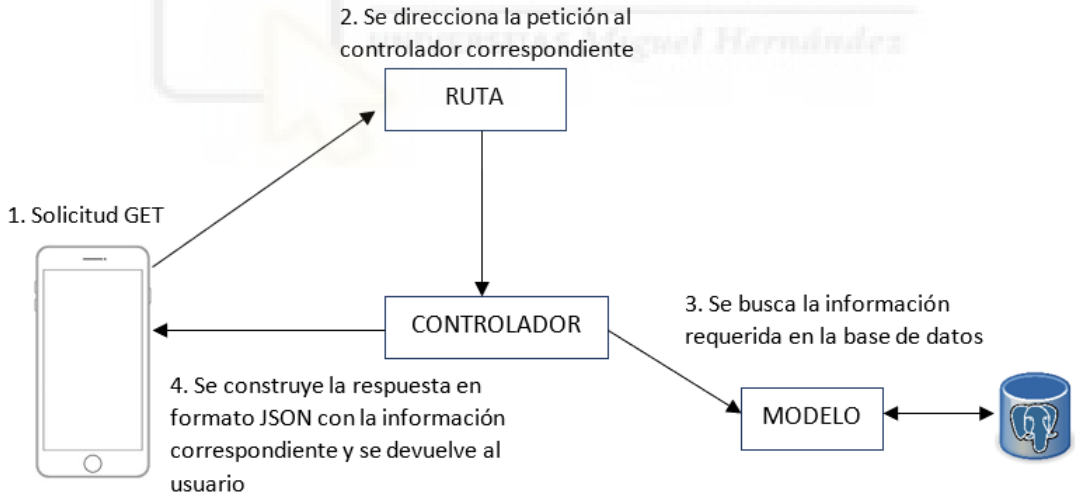

Figura 11. Arquitectura del servidor.

El archivo de rutas define las URIs para poder acceder a cada uno de los recursos. Una URI es un identificador que sirve para definir la identidad de un objeto, independientemente del método utilizado. Cada ruta hará uso de un controlador, los cuales tendrán que ser importados.

Algunas rutas de la API estarán protegidas por "*Sanctum*", que es un sistema de autenticación muy ligero para APIs basadas en tokens. Sanctum permite a cada usuario de la aplicación generar múltiples tokens API para su cuenta. A estos tokens se les pueden otorgar habilidades / ámbitos que especifican qué acciones pueden realizar los tokens. De esta manera, proporciona la lógica necesaria para proteger el acceso a los recursos, para acceder a estos recursos será necesario un token que autorice el acceso. Este token se debe de enviar en la cabecera de la petición HTTP

Todas las rutas a excepción del "*login*"," *closeSession*" y "*register*" están protegidas con "*Sanctum*", por lo que será necesario un token para poder acceder a ellas. La función del "*login*" sirve para autentificar a un usuario en sistema, si las credenciales son correctas, entonces se le entregará un token. La función "*closeSession*" sirve para cerrar una sesión de un usuario, solo aquellos usuarios con un rol específicos podrán hacer uso de esta función. Por último, la función "*register*" es una función para que un usuario se pueda registrar desde la aplicación. Todas las rutas vienen definidas en la Figura 12.

Rutas			
Nombre	Método HTTP	Tipo de Ruta	Descripción
login	POST	Pública	Autentifica al usuario
logout	DELETE	Privada	Cierra la sesión del usuario
signUp	POST	Pública	Registra un usuario nuevo
users/devices/sensors	GET	Privada	Obtiene los sensores del usuario autenticado
cities/Elche/districts	GET	Privada	Obtiene los distritos de la ciudad de Elche

cities/Elche/districts/ {districtId}/neighbourhoods	GET	Privada	Obtiene los barrios de un distrito de la ciudad de Elche
neighbourhood/{neighbourhoodId} /deviceSensors	GET	Privada	Obtiene los sensores de un barrio en concreto
users/favorites/{user_id}	GET	Privada	Obtiene los sensores favoritos del usuario
users/{user_id}/addFavoriteSensor/ {device_sensor_id}	GET	Privada	Añade un sensor a la lista de sensores de ese usuario
users/{user_id}/removeFavoriteSensor/ {device_sensor_id}	GET	Privada	Elimina un sensor a la lista de sensores de ese usuario
measures/getLastMeasures/ {device_sensor_id}	GET	Privada	Obtiene las 500 últimas medidas de un sensor

Tabla 1. Hoja de rutas.

4.2 IMPLEMENTACIÓN DEL BACKEND

Una vez se ha realizado el diseño de todas las partes de las que componen el backend, se procede a realizar su implementación. En este apartado se describe y detalla el procedimiento llevado a cabo para la implementación de cada parte, que conforman el backend. Este procedimiento consta de las siguientes partes:

- Configuración del sistema operativo del servidor. El sistema operativo será Linux y se ubicará en un servidor virtual dedicado con herramientas de acceso *ssh* y *xrdp* configuradas adecuadamente.
- Servidor Web Apache: Para gestionar el backend de una aplicación móvil utilizando Laravel, es necesario instalar Apache. Este servicio actúa como un

servidor web que proporciona alojamiento para la aplicación, ejecutándola cuando recibe una solicitud dirigida a ella y dirigiendo estas peticiones hacia la aplicación para su procesamiento.

- Instalación de Composer y Laravel: Para poder comenzar con la implementación, es necesario descargar los paquetes dependientes necesarios y generar el proyecto Laravel inicial en el que trabajar. Para ello se usa la herramienta de Composer.
- Instalación y configuración de PostgreSQL: Uno de los procesos de implementación del backend, implica la instalación y configuración de la base de datos diseñada para almacenar todos los datos necesarios para el funcionamiento de la aplicación. Como se ha comentado el modelo de datos presentado, se implementará en base a las migraciones de Laravel.
- Implementación del API RESTFUL: En este caso, es necesario implementar el API RESTFUL descrito en la etapa de diseñado mediante las herramientas ofrecidas por Laravel.

4.2.1 Configuración del entorno

El servicio desarrollado se encuentra desplegado en un servidor que utiliza el sistema operativo Linux, específicamente la distribución Debian. Para acceder a este servidor, se requiere acceso directo, ya sea físicamente o a través de una conexión *Secure Shell* (SSH). Una vez que se obtiene acceso, es necesario actualizar los repositorios y los paquetes existentes en el sistema para mejorar la compatibilidad con las aplicaciones utilizadas y garantizar la estabilidad del sistema.

4.2.2 Instalación y configuración de Apache

A continuación, se inicia la instalación de Apache [21] desde el terminal de comandos. Esta aplicación proporciona el servicio de alojamiento para el proyecto y enruta las solicitudes entrantes a los puertos correspondientes. Una vez instalado Apache, se generará el directorio /var/www/html, que es donde se debe ubicar el servicio desarrollado para que sea accesible. Además, es importante verificar si se está utilizando algún cortafuegos en la distribución instalada y, en caso afirmativo, asegurarse de que el puerto 80 esté abierto, ya que Apache utiliza este puerto para las comunicaciones HTTP.

Una vez que Apache esté configurado, se deberá iniciar para que esté disponible para los usuarios. Una vez que Apache esté instalado y configurado, deberá verificar si se ha instalado correctamente. Esto se puede comprobar abriendo un navegador web y navegando hasta la dirección IP del servidor. Si aparece una página que dice "*It works!*", significa que Apache se ha instalado correctamente.

4.2.3 Instalación y configuración de Composer y Laravel

Con Apache instalado, es necesario proceder con la instalación de Composer [22], el cual es el gestor de paquetes que ofrece una forma directa para la instalación y creación de proyectos Laravel, además de las dependencias requeridas en dichos proyectos. Para su instalación es necesario descargar el archivo ejecutable desde la página oficial y ejecutarlo con permisos de administrador. Una vez instalado, se realizará la instalación de Laravel desde Composer. En este caso la instalación se realizará de forma global, así será posible crear el proyecto de Laravel en el directorio deseado. [22]

Con Composer instalado, ya es posible crear el proyecto inicial de Laravel. Para crear el proyecto es necesario posicionarse en el directorio /var/www/html del servidor. Ahí se crearán todas las aplicaciones Laravel gestionadas por Apache. Introduciendo en el terminal la siguiente instrucción:

"*composer create-project laravel/laravel <nombre_proyecto>"*

Se creará la carpeta que contiene un proyecto inicial de Laravel, y descargará, si es necesario, dependencias adicionales para el funcionamiento de dicho proyecto.

Una vez creado el proyecto no es posible hacer uso de él directamente porque no contiene los permisos y propiedades necesarios para acceder al almacenamiento y a la cache del servidor. Para garantizar dichos permisos y propiedades es necesario ejecutar los siguientes comandos:

- *sudo chown -R $USER:www-data storage*
- *sudo chown -R $USER:www-data bootstrap/cache*
- *chmod -R 775 storage*
- *chmod -R 775 bootstrap/cache*

Una vez ejecutados todos estos comandos, es necesario reiniciar el servicio de Apache para que los cambios surjan efecto:

sudo systemctl restart apache2

Para comprobar que todo lo que se ha realizado funciona correctamente, hay que abrir un navegador y poner la dirección del servidor seguido del nombre del proyecto y seguido de "public" y debe de salir la página por defecto de una aplicación Laravel recién instalada.

4.2.4 Implementación de la base de datos

Una vez se disponga del proyecto base de Laravel, lo recomendable es comenzar con la implementación de la base de datos. Para ello se debe empezar por instalar el gestor de bases de datos, es este caso se ha optado por emplear PostgreSQL.

El proceso de instalación es bastante sencillo, lo primero es actualizar repositorios e instalar últimas actualizaciones, escribiendo en la consola:

- *sudo apt update*
- *sudo apt upgrade*

A continuación, se instalan los paquetes para PostgreSQL:

sudo apt install postgresql postgresql-contrib

Y ya estaría instalado en el servidor, se puede ver que versión hay instalada introduciendo en la consola:

psql --version

Una vez que ya se ha instalado el gestor PostgreSQL, se procede a conectar este gestor y Laravel. Para empezar a modificar la base de datos hay que configurar los parámetros de conexión a la base de datos. Laravel almacena las variables de entorno (ver Figura 13) en el fichero ".env", allí se encuentran las siguientes variables relacionadas con la conexión a la base de datos, en este caso se utiliza PostgreSQL, alojado en el mismo servidor.

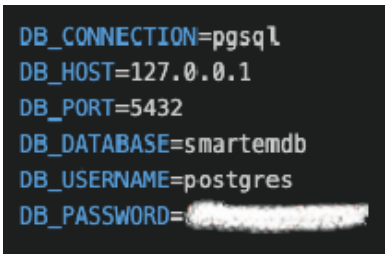

Figura 12. Variables de entorno Laravel.

Para crear cada una de las tablas de la base de datos (ver Figura 14), se utiliza el comando de Artisan "*php artisan make:migration nombre_tabla*". Esto generará una nueva clase que deberá modificarse según las necesidades específicas de esa tabla, como los atributos, las claves principales, las claves foráneas, etc.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('neighbourhoods', function (Blueprint $table) {
            $table->id();
            $table->foreignId('district_id');
            $table->string('name',45);
            $table->timestamps();

            $table->foreign('district_id')->references('id')->on('districts')->onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('neighbourhoods');
    }
};
```

Figura 13. Ejemplo de una migración (tabla).

Las migraciones están compuestas por dos funciones: "*up*" y "*down*". La función "*up*" se ejecuta durante la migración, es decir, cuando se inicia el proceso de transferencia de tablas y atributos a la base de datos. Todo el código contenido en esta función se ejecutará en este evento, por lo que generalmente se utiliza para crear tablas y definir sus atributos. Por otro lado, la función "*down*" se ejecuta en caso de un *rollback* o reversión de la migración. En esta función, típicamente se incluye la lógica para eliminar tablas o atributos.

Una vez se hayan creado todas las tablas que se encuentran en el modelo de la base de datos, se ejecutará el comando de *Artisan* "*php artisan migrate*" para iniciar la migración a la base de datos en PostgreSQL.

Una vez que la base de datos está configurada y con todas sus tablas creadas, se deben preparar los accesos a estas tablas. En este proyecto, se ha utilizado el ORM (*Object-Relational Mapper*) Eloquent, (ver Figura 15) que consiste en una técnica de programación que permite a los desarrolladores trabajar con bases de datos relacionales usando objetos en lugar de escribir consultas SQL directamente.

Eloquent simplifica la interacción con la base de datos al proporcionar una capa de abstracción que mapea objetos de PHP a filas en tablas de la base de datos y viceversa.

Además de las operaciones básicas de CRUD (Crear, Leer, Actualizar, Eliminar), Eloquent también proporciona relaciones, lo que permite definir relaciones entre diferentes modelos de manera sencilla y expresiva.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class SensorType extends Model
{
    use HasFactory;

    /**
     * Get all device sensors that are of a sensor type
     *
     * @return \Illuminate\Database\Eloquent\Relations\hasMany
     */
    public function device_sensors(){
        return $this->hasMany(DeviceSensor::class);
    }
}
```

Figura 14. Ejemplo de modelo Eloquent.

Una vez estén implementados todos los modelos necesarios se puede hacer uso del constructor de consultas de Eloquent. Para poder construir las consultas se deberá importar los modelos necesarios.

A continuación, en la Figura 16, se muestra en el recuadro en rojo la importación de los modelos que se van a usar en esta consulta, y en el recuadro verde se muestra la construcción de la consulta, haciendo uso del constructor de consultas de Eloquent.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\ResponseStatus;
use App\Models\DeviceSensor;
use App\Models\Measure;
use Illuminate\Support\Facades\DB;


class MeasureController extends Controller
{
    public function getMeasuresFromDeviceSensor($device_sensors_id){
        $currentUser=auth()->user();

        if (!$currentUser->HasPrivilege('list_devices')){
            $rdata=[
                "status" => ResponseStatus::UNAUTHORIZED,
                "message" => "Not enough privileges"
            ];
            return response()->json($rdata,403);
        }

        $measures = Measure::where('device_sensors_id',$device_sensors_id)->get();
        $rdata=[
            "status" => ResponseStatus::OK,
            "measures" => $measures
        ];

        return response()->json($rdata, 200);
    }
}
```

Figura 15. Ejemplo de consulta con Eloquent.

4.3 DISEÑO DEL FRONTEND

La arquitectura más usada para aplicaciones que usan Jetpack Compose es la de MVVM [23] (*Model-View-ViewModel*), el principal objetivo de esta arquitectura es separar el apartado de la interfaz de usuario (*View*) de la parte lógica (*Model*). El ViewModel, se encarga de servir como enlace entre la interacción de la vista (*View*) y el Modelo (*Model*)

Dentro de las características relevantes del patrón de arquitectura MVVM, destaca su capacidad para separar de manera clara en de una aplicación la lógica del negocio de su interfaz de usuario. Esta separación facilita el abordaje de diversos problemas de desarrollo, simplificando los procesos de prueba, mantenimiento y evolución del sistema.

Además, con el patrón MVVM, permite el trabajo simultáneo e independiente de los desarrolladores y diseñadores en sus respectivas áreas durante el desarrollo de la aplicación. Mientras los diseñadores se centran en la vista, los desarrolladores pueden

ocuparse de los componentes del Modelo y del Modelo de Vista en la arquitectura MVVM para Android. En la figura 17 se muestra de manera gráfica como funciona esta arquitectura.

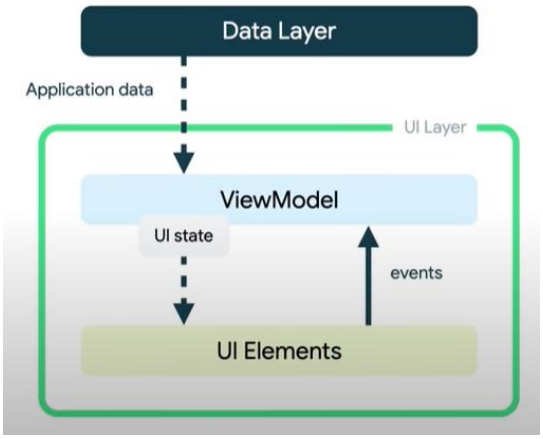

Figura 16. Arquitectura de la aplicación. Fuente: Miguel Martínez [30].

En esta sección se tratará del diseño completo realizado de la parte del frontend del servicio.

- Inicio de sesión: Uno de los requisitos para poder acceder a la información contenida en el servicio es estar autenticado, por lo que el primer paso para el usuario es autenticarse.
- Obtención de la lista de sensores: Una vez se ha autenticado un usuario, podrá acceder a lista de sensores existentes en el sistema.
- Información de cada sensor: Al pulsar en un sensor, se navegará a una página donde se muestra información más detallada de ese sensor.
- Mapa de posición: Se dibujará un mapa usando Google Maps en el que se vea la localización de cada sensor y su valor.
- Información de las medidas: Para conocer si el valor que posee cada sensor es bueno o malo se ha realizado una pantalla detallando una clasificación de los posibles resultados

4.3.1 Diseño de Pantallas y Navegación

En la tabla 1 se muestra una breve descripción de cada pantalla, que, en su conjunto, conforman la aplicación.

Pantalla	Breve descripción
Bienvenida	Esta es la primera pantalla que se muestra al abrir la aplicación, en ella se observa el nombre de la aplicación y un botón para navegar al Login. Si el usuario ya está autenticado no verá esta pantalla.
Login	Esta pantalla sirve para hace el inicio de sesión en el sistema, para ellos se deberá introducir las credenciales necesarias. Si el proceso se ha completado con éxito se navega a la pantalla donde se muestran los sensores.
Sensores	Una vez autenticado el usuario, se muestra una lista con todos los sensores que ese usuario puede observar, así como navegar a una pantalla donde se muestre un mapa con la posición del sensor.
Información adicional	Al pulsar el botón de información adicional de un sensor, se navega a esta pantalla donde te muestra más información del sensor que se ha pulsado
Gráficas	Cuando un usuario pulse el icono de la gráfica en la pantalla donde se encuentra la lista de sensores, navegará a esta pantalla donde se muestra una gráfica con los últimos 500 valores de ese sensor
Filtros	Esta pantalla sirve para filtrar los sensores por posición, cada sensor, está ubicado en un barrio de Elche, por lo que es posible realizar un filtrado y observar aquellos sensores que quiera el usuario.

Mapa	Esta pantalla muestra un mapa con la posición de cada sensor, además, al pinchar en el icono del mapa, se puede observar el último valor de ese sensor.
Clasificación	Aquí se muestra una clasificación con los diferentes niveles de audición y pequeños ejemplos de donde se pueden obtener dichos niveles.
Perfil	Esta pantalla muestra la información relacionada con el perfil del usuario.
Cierre de sesión	Esta pantalla se muestra al pulsar el botón “Cerrar sesión” ubicado en el menú lateral, y en ella se muestra un botón para cerrar la aplicación y orto para volver a iniciar sesión

Tabla 2.Pantallas que conforman la aplicación.

Para entender mejor como se navega entre pantallas se ha diseñado el siguiente esquema de la figura 18, don se aprecia el flujo de navegación de esta aplicación.

Figura 17. Diagrama de navegación.

4.3.1 Bienvenida

Esta pantalla será la primera que verá el usuario al entrar en la aplicación, si el usuario esta ya autenticado, directamente verá la pantalla donde se muestran los sensores. Esta pantalla consta del título de la aplicación la cual se llama “Smart Environmental Monitor”, más abajo se encuentra el logo de la aplicación y un botón el cual, al pulsarlo, la aplicación navega a la pantalla de autenticación. En la figura 19 se muestra el resultado final.

Figura 18. Pantalla bienvenida.

4.3.2 Inicio de sesión

La primera parte del frontend consiste en el sistema de inicio de sesión. En este caso se ha decidido que consista en un formulario en el que el usuario debe introducir su correo electrónico y su contraseña, una vez que se han rellenado estos campos y se ha pulsado el botón de iniciar sesión, el sistema deberá enviar dichas credenciales al backend de forma asíncrona. El frontend generará una petición POST al backend con los parámetros del formulario, y si dichas credenciales son correctas, recibirá una respuesta, también asíncrona, con un token que identifique a este usuario. Dicho token deberá ser almacenado en la memoria del móvil para posteriores peticiones que se realicen. En la figura 20 se muestra el resultado final.

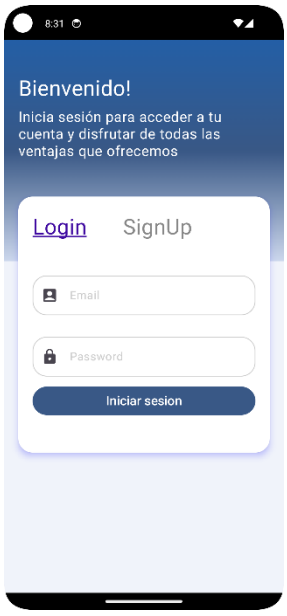

Figura 19. Pantalla de Login.

4.3.3 Sensores

Una vez se haya iniciado sesión correctamente y el usuario este autenticado, se accederá a una nueva pantalla en la que se recuperan del API todos los sensores del sistema. En esta pantalla aparece un cuadro de búsqueda junto con una opción para filtrar sensores según donde se encuentren, y una opción para ordenar la lista de sensores por orden alfabético.

Cada elemento de la lista posee un nombre, un icono para poder ver una gráfica donde se muestra el histórico de los datos y un botón para añadir este sensor a una lista de favoritos. El indicador muestra en una escala de colores el último valor que ha captado el sensor, el color del indicador muestra si es valor es aceptable o por el contrario es muy alto. Además, hay un botón en la parte superior llamada “Favoritos”, que permite ver una lista con los sensores favoritos del usuario. En la figura 21 se muestra la pantalla que contiene la lista de sensores.

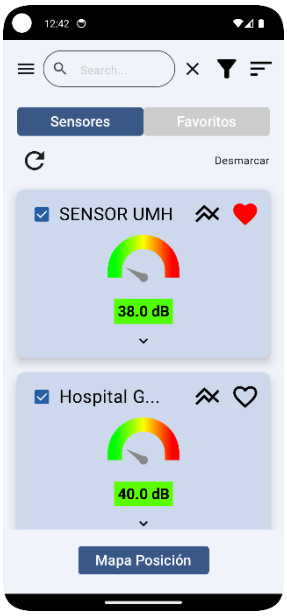

Figura 20. Pantalla con la lista de sensores.

4.3.4 Información

Al pulsar el texto que aparece en la lista de sensores, “pinche aquí para más información” se navega a una pantalla donde se muestra más información del sensor seleccionado. En la figura 22 se muestra un ejemplo de esta interfaz con un sensor.

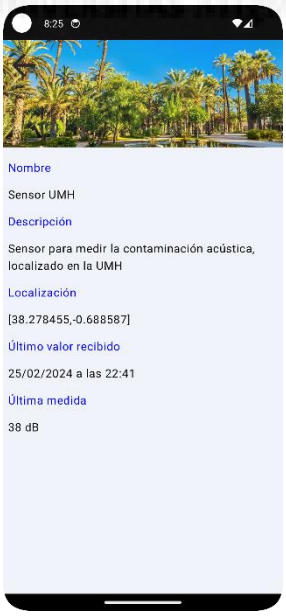

Figura 21. Pantalla con información adicional.

4.3.5 Gráfica

Al pulsar en el icono de la gráfica, se navega a una pantalla la cual muestra una gráfica con las últimas 500 medidas de ese sensor. En el eje de ordenadas se muestran los valores en decibelios y en el eje de abscisas se muestra la fecha en la que se obtuvieron esos valores. La información se muestra en un gradiente de colores, que va desde el verde hasta el rojo, siendo los valores en verde los mas bajos, y en rojo aquellos valores mas perjudiciales para el ser humano. En la figura 23 se muestra una gráfica con los valores que ha obtenido un sensor.

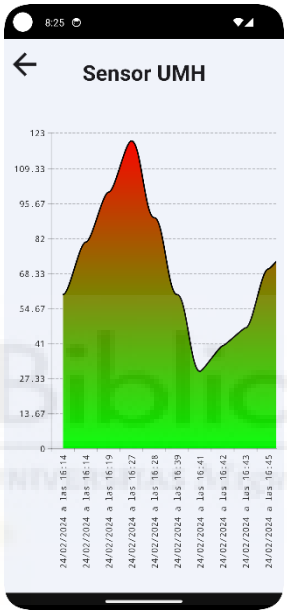

Figura 22.Pantalla con la gráfica.

4.3.6 Filtros

En esta pantalla se muestran todos los distritos de Elche, así como los barrios ubicados en estos distritos. El usuario puede elegir los barrios y ver que sensores hay en dichos barrios. Abajo se encuentra un resumen de los filtros seleccionados, es decir, los distritos y barrios seleccionados anteriormente. Al pulsar el botón “Aplicar” se realiza la búsqueda de los sensores en dichos barrios en la base de datos, y el API devuelve una lista con los sensores, si los hay, en dichos barrios. En la figura 24 se muestra la pantalla que contiene todos los filtros existentes.

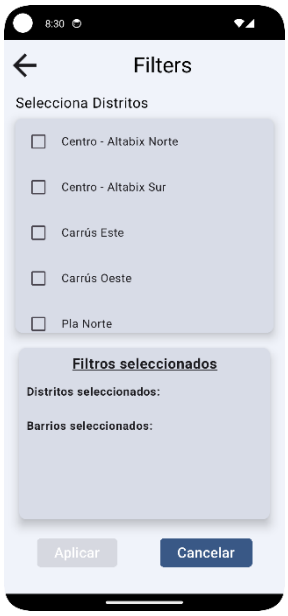

Figura 23. Pantalla de filtros.

4.3.7 Mapa

Otra funcionalidad de esta aplicación, es que permite observar en un mapa la localización de los sensores seleccionados, en la pantalla de los sensores, el usuario puede seleccionar que sensores quiere observar en el mapa. Al pulsar el icono que aparece en el mapa, aparece un cuadro de texto el cual muestra el nombre del sensor, una breve descripción y el último valor recibido. En la figura 24 se muestra un ejemplo con sensores ubicados por todo el municipio de Elche.

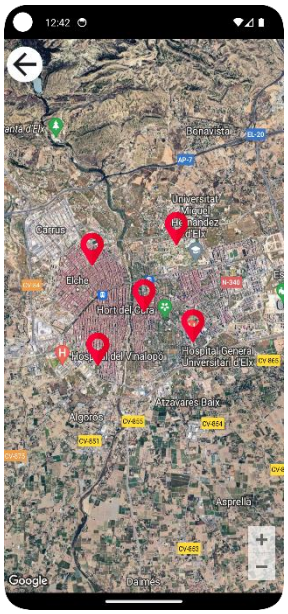

Figura 24. Pantalla del mapa.

4.3.8 Clasificación

En esta pantalla se muestra una clasificación de los diferentes niveles de audición, asi como un ejemplo de lo que puede generar dicho nivel, además se especifica los riesgos que tiene para la salud dicho nivel de intensidad sonora. La clasificación se muestra en colores, siendo el rojo el peor valor posible, y el verde el valor más favorable. En la figura 26 se muestra la clasificación en la que se ha basado esta aplicación.

Figura 25. Pantalla de clasificación.

4.3.9 Perfil

En esta pantalla se muestra la información referente al usuario, una foto de perfil junto con su nombre y el país donde vive. También se muestran opciones para cambiar el correo electrónico y la contraseña. En la figura 27 se muestra como es esta pantalla.

Figura 26. Pantalla de perfil.

4.3.10 Cierre de sesión

Esta pantalla es la última que se muestra cuando un usuario cierra sesión, en ella se muestra un mensaje de despedida, y dos botones, uno para cerrar la aplicación y otro para volver a iniciar sesión en la aplicación. En la figura 28 se muestra como es la interfaz de esta pantalla.

Figura 27. Pantalla cierre sesión.

4.4 IMPLEMENTACIÓN DEL FRONTEND

En este apartado, se explora la implementación del frontend utilizando Jetpack Compose, se centra en concreto en la navegación entre pantallas, la gestión de estados y la recomposición de la interfaz. A través de una serie de subpuntos, se examinarán los conceptos clave y las mejores prácticas para crear una experiencia de usuario fluida y receptiva en las aplicaciones Android. Desde el diseño de rutas de navegación coherentes hasta la manipulación de datos en tiempo real. Esta parte se ha dividido en los siguientes apartados

- Navegación: Es importante recalcar el uso de la navegación entre pantallas puesto que en una aplicación lo normal es que se vaya de una pantalla a otra o al menos cambien ciertos elementos de la interfaz de usuario.
- Estados y Recomposición: El estado se refiere a la representación actual de los datos en la interfaz de usuario. Este estado puede cambiar en respuesta a eventos o acciones del usuario. Por otro lado, la recomposición es el proceso mediante el cual Jetpack Compose actualiza automáticamente la interfaz de usuario cuando cambia el estado.
- Consumo de APIs: Toda la información necesaria para esta aplicación se encuentra almacenada en la base de datos, por lo tanto, será necesario realizar el consumo

4.4.1 Navegación

Hasta hace poco, era común utilizar *Intents* para cambiar de una actividad a otra en el desarrollo de aplicaciones Android. Sin embargo, en la actualidad, se recomienda encarecidamente utilizar el componente *Navigation*. Esta práctica nos permite consolidar nuestra aplicación en una sola actividad y facilita la transición entre distintas funciones "Composable".

Una ventaja significativa de usar el componente *Navigation* es que proporciona una estructura de navegación declarativa y centralizada, lo que simplifica el diseño y la comprensión del flujo de la aplicación.

La navegación se lleva a cabo mediante la API central del componente *Navigation* que tiene el nombre de *NavController*. Este elemento debe estar asociado a un *NavHost*, que contiene un gráfico de navegación que especifica los destinos componibles por los que se puede navegar. A medida que se navega por los elementos que admiten composición, el contenido del *NavHost* se reescribe automáticamente. Cada destino que admite composición en el gráfico de navegación debe estar asociado con una ruta.

Para la implementación de la navegación en primer lugar se debe agregar la dependencia en el *gradle*, en este caso es:

implementation ("androidx.navigation:navigation-compose: 2.5.3")

En primer lugar, se crea una lista con el nombre de las rutas que tendrá la aplicación, como la que se muestra en la Figura 18.

```
enum class NavigationScreen() {
    Welcome,
    Login,
    SignUp,
    Sensors,
    Filters,
    HeatMap,
    ChangePassword,
    Profile,
    Information,
    CloseSession,
    InfoSensor

}
```

Figura 28. Nombre de las rutas de la aplicación.

Una vez creada la lista de las rutas, se pasará a crear el *NavHost*, que es un contenedor el cual contiene las funciones componibles que se llamarán en el momento que se navegue a esa ruta, como se muestra en la Figura 19. En dicha figura se muestra como el componente *NavHost* tiene un parámetro de tipo *NavHostController*, este parámetro lo que hace es gestionar el flujo de navegación de la aplicación. De esta forma, el

NavController interactúa con el *NavHost* para mostrar los destinos de navegación correspondientes en la pantalla y gestionar su transición.

Figura 29. Funciones componibles contenidas en el NavHost.

Para utilizar la navegación y que cambie de un componente a otro simplemente habría que llamar al método *navigate* de la clase *NavController*, por ejemplo, para ir a la pantalla de Sensores incluiríamos la línea de la Figura 20.

```
navController.navigate(NavigationScreen.Sensors.name)
```

Figura 30. Ejemplo de navegación a una pantalla.

4.4.2 Estados y recomposición

Una vez que se ha definido el comportamiento de la interfaz y el funcionamiento de la navegación sólo quedaría describir cómo una función componible observa una variable del *ViewModel.*

En primer lugar, para que la función componible pueda hacer uso de las variables del *ViewModel*, se debe crear una instancia de ésta que utilice la propia clase *ViewModel*, esta clase almacena y administra datos relacionados con la interfaz de usuario de manera

optimizada para los ciclos de vida, permite que se conserven los datos después de cambios de configuración.

Una vez que se ha instanciado el *ViewModel* en la *Activity*, ya se pueden "observar" las variables del "*ViewModel*". En este caso, se aprecia que el *ViewModel* contiene muchas variables de estado, como se puede apreciar en la Figura 21.

```
class MainViewModel : ViewModel() {

    private val repository = Repository()

    private val _login = MutableStateFlow(Login())
    val login: StateFlow<Login> = _login.asStateFlow()

    private val _userSensors = MutableStateFlow(UserSensors())
    val userSensors: StateFlow<UserSensors> = _userSensors.asStateFlow()

    private val _favoriteUserSensors = MutableStateFlow(FavoriteUserSensors())
    val favoriteUserSensors: StateFlow<FavoriteUserSensors> = _favoriteUserSensors.asStateFlow()

    private val _districts = MutableStateFlow(Districts())
    val districts: StateFlow<Districts> = _districts.asStateFlow()

    private val _neighbourhoods = MutableStateFlow(Neighbourhoods())
    val neighbourhoods: StateFlow<Neighbourhoods> = _neighbourhoods.asStateFlow()

    private val _filteredSensors = MutableStateFlow(FilteredSensors())
    val filteredSensors: StateFlow<FilteredSensors> = _filteredSensors.asStateFlow()
```

Figura 31. Varibles de estado del ViewModel.

Cada una de estas variables representan el flujo de los datos que son relevantes para la vista y pueden cambiar durante el ciclo de vida de la aplicación.

Estas variables son muy útiles en la medida en que permiten emitir y observar cambios de estado de manera reactiva, lo que simplifica la actualización de la interfaz de usuario en respuesta a cambios en ese estado. Además, al ser mutable, se puede cambiar su valor cuando sea necesario en el ViewModel y los cambios se reflejarán automáticamente en cualquier parte de la aplicación que esté observando ese flujo de datos.

4.4.3 Consumo de APIs

Para realizar solicitudes a la API, se ha utilizado Retrofit, una biblioteca que simplifica el proceso al ofrecer objetos y métodos que facilitan el manejo de la API. Retrofit abstrae al

programador de los detalles de la comunicación, tanto para enviar solicitudes como para recibir y convertir los datos posteriormente.

Es necesario incluir dependencias en el *gradle* para hacer uso de sus métodos y para parsear las respuestas JSON.

implementation ("com.squareup.retrofit2:retrofit:2.9.0")

implementation("com.squareup.retrofit2:convertergson:2.2.0")

Para poder utilizar Retrofit dentro de Android, además de configurar el fichero *'build.gradle'*, es entrar en el fichero *'android.manifest'* de la aplicación y añadir una serie de permisos (ver Figura 22). En este caso, como necesitamos acceder a un servicio REST en un servidor en Internet, se le debe indicar al *'android.manifest'* que se necesita, por parte del usuario, permiso para conexión a Internet. Sin este permiso, no se podrá acceder a Internet y no podrán realizar las llamadas al API.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Figura 32. Permisos declarados en el Manifest.

Una vez configurado Retrofit, se tienen que definir cada uno de los *endpoints* de la API que se van a utilizar en la aplicación, estos se deben definir en una interfaz, y allí se definirán sus métodos abstractos que representarán cada una de las rutas específicas de la API. Para este proyecto, las peticiones que se realizan, usan los métodos POST, GET y DELETE. En la Figura 23 se muestran unos ejemplos de *endpoints.*

```kotlin
interface API {

    companion object {
        const val BASE_URL = "http://157.90.22.135/smartemserver/public/api/"
    }

    @FormUrlEncoded
    @POST("login")
    fun login(
        @Field("email") email: String,
        @Field("password") password: String,
    ): Call<LoginResponse>

    @HTTP(method = "DELETE", path = "login", hasBody = true)
    fun logout(@Body request: Logout): Call<LogoutResponse>

    @GET("users/devices/sensors")
    fun getSensorsAllUsers(): Call<SensorsUserResponse>

    @GET("cities/Elche/districts")
    fun getDistricts(): Call<DistrictsResponse>

    @GET("cities/Elche/districts/{districtId}/neighbourhoods")
    fun getNeighbourhoods(@Path("districtId") districtId: String): Call<NeighbourhoodsResponse>

    @GET("neighbourhood/{neighbourhoodId}/deviceSensors")
    fun getSensorOfNeighbourhood(@Path("neighbourhoodId") neighbourhoodId: String): Call<SensorsFilteredResponse>

    @GET("users/favorites/{user_id}")
    fun getFavoriteSensors(@Path("user_id") userId: Int): Call<FavoriteSensorsResponse>
```

Figura 33. Endpoints de la aplicación.

Una vez que se hayan definido todos los *endpoints* necesarios, el siguiente paso es crear el objeto Retrofit. Sin embargo, antes de hacerlo, es crucial establecer un interceptor para configurar ciertos parámetros de la solicitud. Por ejemplo, es esencial incluir en la petición el token que autorice el acceso a la API (ver Figura 24). Este interceptor se encargará de agregar las cabeceras (*headers*) necesarias para la comunicación.

```kotlin
class AuthorizationHeader:Interceptor {

    override fun intercept(chain: Interceptor.Chain): Response {
        val request=chain.request().newBuilder()
            .header( name: "Authorization", value: "Bearer "+SPApplication.preferences.getToken())
            .build()
        return chain.proceed(request)
    }
}
```

Figura 34. Interceptor.

El token esta almacenado en las *sharedPreferences*, ésta es una característica de Android que permite almacenar y recuperar pequeñas cantidades de datos de manera persistente y privada para cada aplicación.

Para crear el objeto Retrofit llamaremos al método *Builder()*, al cual le pasaremos el interceptor, la URL base y el parseador de GSON (ver Figura 25).

```
companion object {
    var client = OkHttpClient()
    val clientWithHeader = client.newBuilder().addInterceptor(AuthorizationHeader()).build()
}

private fun retrofitWithHeader(): Retrofit {
    return Retrofit.Builder() Retrofit.Builder
        .client(clientWithHeader) Retrofit.Builder
        .baseUrl(API.BASE_URL)
        .addConverterFactory(GsonConverterFactory.create())
        .build()
}
```

Figura 35. Objeto Retrofit.

Una vez que se ha creado el objeto Retrofit necesario para realizar la petición, se procede a realizar la llamada asíncrona, tal y como se muestra en la Figura 26. Para poder hacer peticiones HTTP debemos instanciar una llamada (Call) y llamar al método del endpoint que queramos usar, en este caso para hacer el "Logout" de la aplicación. Se llama al método "enqueue" que hará una llamada asíncrona. Para impedir bloqueos de la interfaz, este método obliga a sobrescribir sus dos métodos, "onResponse" si todo ha ido bien y ha habido una respuesta correcta y "onFailure" si ha habido algún problema en la llamada.

```
suspend fun logout(viewModel: MainViewModel, context: Context, request: Logout) =
    withContext(Dispatchers.IO) { this: CoroutineScope
        val call: Call<LogoutResponse> =
            retrofitWithHeader().create(API::class.java).logout(request)
        call.enqueue(object : Callback<LogoutResponse> {
            override fun onResponse(
                call: Call<LogoutResponse>,
                response: Response<LogoutResponse>
            ) {
                if (response.body()?.status == 1000) {
                    viewModel.updateStateLogin( state: false)
                    SPApplication.preferences.removeToken()
                } else {
                    Toast.makeText(context, response.message(), Toast.LENGTH_LONG).show()
                }
            }

            override fun onFailure(call: Call<LogoutResponse>, t: Throwable) {
                Toast.makeText(context,  text: "Error en el logout", Toast.LENGTH_LONG).show()
            }
        })
    }
```

Figura 36. Ejemplo de llamada asíncrona al API.

5. CASO PRÁCTICO

La primera pantalla que aparece al abrir la aplicación la de bienvenida, al pulsar el botón de “Comenzamos” se navega a la pantalla de login (ver figura 37).

Figura 37. Bienvenida aplicación.

En esta pantalla hay que introducir las credenciales para poder iniciar sesión en la aplicación tal y como se muestra en la figura 38.

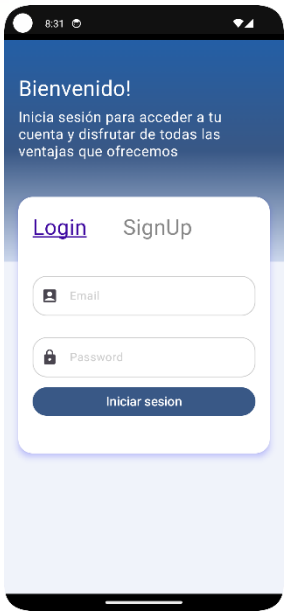

Figura 38. Login aplicación.

Si las credenciales son correctas se podrá navegar entre la lista de sensores y el mapa tal y como se observa en la figura 39.

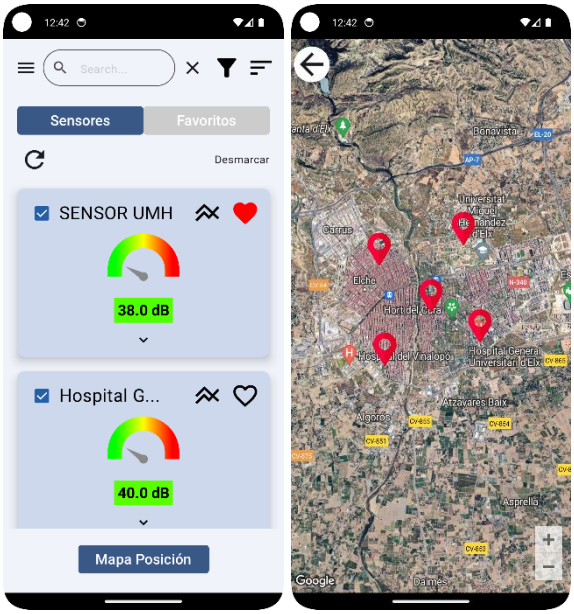

Figura 39. Navegación entre lista de sensores y mapa.

Si se quiere obtener más información de un sensor o ver la gráfica se sus valores, basta con pulsar el icono correspondiente (ver figura 40).

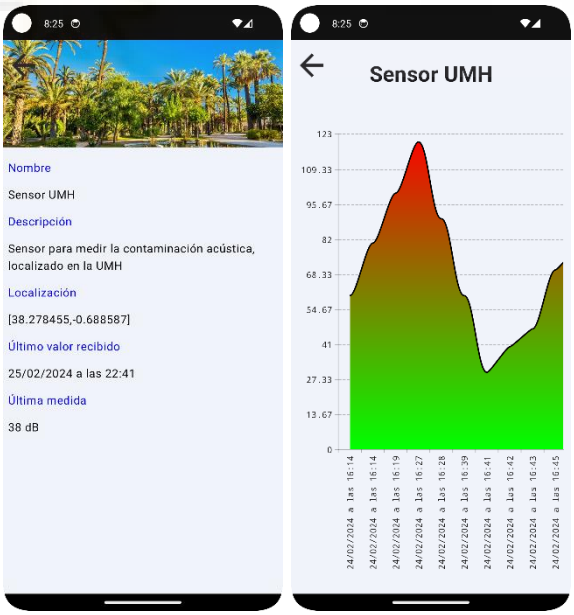

Figura 40. Información adicional y gráfica.

Si se pulsa el botón del menú lateral situado arriba a la izquierda, veremos las opciones de ir a la pantalla de perfil o a la pantalla de clasificación. También está la opción de cerrar sesión y cambiar a modo oscuro (ver figura 41).

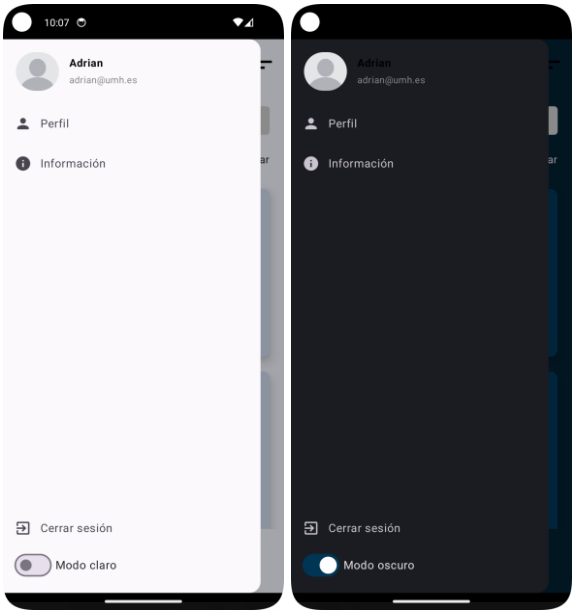

Figura 41. Menú lateral en modo claro y oscuro.

Al pulsar en Perfil o en Información se navegará hasta las pantallas que se muestran en la figura 42

Figura 42. Clasificación y perfil.

Por último, al pulsar el botón “Cerrar sesión”, ya sea en el menú lateral o en la pantalla del perfil se navegará hasta la pantalla descrita en la figura 43

Figura 43. Cierre de sesión.

6. CONCLUSIONES

Una vez concluido el proyecto, se puede afirmar que se han alcanzado los objetivos establecidos. Construir un sistema que permita la interacción entre varios usuarios no es una tarea trivial.

El proceso de desarrollo de este proyecto ha sido una experiencia enriquecedora que ha implicado aprender nuevas tecnologías y enfrentarse a diversos desafíos. Aunque contaba con experiencia previa en el desarrollo de aplicaciones Android utilizando Kotlin, el aprendizaje de tecnologías de servidor como PHP y el manejo básico del framework Laravel fue fundamental. La comprensión y dominio de APIs desde cero resultaron esenciales, ya que son ampliamente demandadas en el mercado y juegan un papel crucial en el desarrollo de sistemas modernos.

Por otro lado, la adopción de Jetpack Compose para la interfaz de usuario en la aplicación Android representó un cambio significativo. Este cambio de paradigma, junto con el uso de Kotlin, promovió la adopción de patrones de diseño como MVVM. Este enfoque estructural ordena el código y lo hace más claro, lo cual es una práctica positiva que a menudo se pasa por alto en la formación académica.

En resumen, tanto en el servidor como en la aplicación Android, el manejo eficiente de conexiones mediante APIs es esencial, ya que la mayoría de las aplicaciones requieren el consumo de datos externos. Este proyecto ha destacado la importancia de estar siempre dispuesto a aprender nuevas tecnologías y adaptarse a los cambios del mercado para desarrollar soluciones efectivas y modernas.

7. TRABAJOS FUTUROS

Para futuros desarrollos, se pueden considerar varias áreas de expansión y mejora para el proyecto. Por ejemplo, sería posible ampliar la plataforma desarrollando una interfaz web complementaria que permita a los usuarios acceder a los datos de los sensores desde cualquier dispositivo con acceso a Internet, lo que ampliaría la accesibilidad y la versatilidad de la solución.

La implementación de esta interfaz web podría beneficiarse de la integración de tecnologías como Websockets para facilitar la comunicación en tiempo real entre el servidor y los dispositivos de los usuarios, lo que garantizaría una experiencia fluida y actualizada.

También se podría explorar la inclusión de nuevos tipos de sensores IoT para monitorear diferentes aspectos ambientales o de la infraestructura, como la contaminación atmosférica o la calidad del agua, con el fin de proporcionar una visión más completa del entorno y abordar una gama más amplia de aplicaciones y necesidades.

Otra posibilidad sería implementar funcionalidades avanzadas, como algoritmos de análisis de datos y *machine learning* para identificar patrones, tendencias o anomalías en los datos recopilados, lo que podría mejorar la capacidad predictiva y analítica de la plataforma.

Además, sería importante realizar estudios de usabilidad y retroalimentación de los usuarios para identificar áreas de mejora en la interfaz de usuario y la experiencia general de la aplicación, optimizando la navegación, personalizando las preferencias del usuario y agregando características adicionales solicitadas por los usuarios.

Estas son solo algunas ideas que podrían ayudar a expandir y mejorar el proyecto, ofreciendo nuevas funcionalidades, mayor precisión en la recopilación de datos y una experiencia de usuario más enriquecedora.

8. BIBLIOGRAFÍA

[1] Contaminación acústica

https://www.monografias.com/trabajos/contamacus/contamacus

[2] Ruido Ambiental y la importancia de su monitoreo continuo.

https://hteltda.com/medicion-del-ruido-ambiental-y-la-importancia-de-su-monitoreo-continuo/#:~:text=El%20monitoreo%20continuo%20de%20ruido,ruido%20que%20soportan%20sus%20empleados

[3] ¿Qué es un IoT?

https://www.redhat.com/es/topics/internet-of-things/what-is-iot

[4] El internet de las cosas: su evolución en los últimos años

https://www.tokioschool.com/noticias/internet-de-las-cosas-evolucion/

[5] Futuro del IoT

https://bisite.usal.es/es/blog/formaci-n/23/08/23/el-internet-de-las-cosas-transformando-nuestro-mundo-conectado-bisite

[6] Android Studio

https://es.wikipedia.org/wiki/Android_Studio

[7] Android Studio vs Eclipse

https://jesicarinaldi.com.ar/blog/desarrollo-android/android-studio-vs-eclipse/

[8] Características de Android Studio

https://developer.android.com/studio/features?hl=es-419

[9] Historia de Kotlin

https://www.plainconcepts.com/es/kotlin-android/#Historia_de_Kotlin

[10] Jetpack Compose

https://keepcoding.io/blog/que-es-jetpack-compose/

[11] Jetpack Compose: el paradigma declarativo llega al desarrollo Android nativo

https://www.izertis.com/es/-/blog/jetpack-compose-paradigma-declarativo-llega-al-desarrollo-android-nativo

[12] Retrofit

https://www.geeksforgeeks.org/introduction-retofit-2-android-set-1/

[13] Introducción a Laravel

https://www.sitepoint.com/laravel-introduction/

[14] ¿Qué es Composer?

https://www.javatpoint.com/laravel-composer-installation

[15] Artisan Console

https://laravel.com/docs/5.1/artisan#:~:text=Artisan%20is%20the%20name%20of,php%20artisan%20list

[16] PHP Introducction

https://www.geeksforgeeks.org/php-introduction/

[17] What is PostgreSQL?

https://www.linode.com/docs/guides/an-introduction-to-postgresql/

[18] What is PgAdmin?

https://www.adservio.fr/post/what-is-pgadmin

[19] API: qué es y para qué sirve

https://www.xataka.com/basics/api-que-sirve

[20] Arquitectura REST: la arquitectura del momento

https://gausswebapp.com/arquitectura-rest.html

[21] Guía completa de como instalar apache en Debian 11

https://comandosenlinux.com/debian/instalar-apache-en-debian-11/

[22] Instalación de Composer

https://getcomposer.org/doc/00-intro.md#installation-linux-unix-macos

[23] ¿Qué es el patrón de arquitectura MVVM?

https://keepcoding.io/blog/que-es-el-patron-de-arquitectura-mvvm/#:~:text=El%20patr%C3%B3n%20de%20arquitectura%20MVVM%2C%20tambi%C3%A9n%20conocido%20como%20Model%20View,la%20parte%20l%C3%B3gica%20(Model)

[24] Contaminación acústica

http://www.sorolls.org/docs/CA_monografias.htm

[25] Qué es IoT y su impacto en la industria de la seguridad electrónica

https://www.tecnoseguro.com/analisis/pro/que-es-iot-su-impacto-industria-seguridad-electronica

[26] Amazing IoT Statistics

https://explodingtopics.com/blog/iot-stats

[27] Android Studio

https://developer.android.com/studio/past-releases/past-android-studio-releases/as-3-5-0-release-notes?hl=es-419

[28] Estrategia de migración

https://developer.android.com/jetpack/compose/migrate/strategy?hl=es-419

[29] Understand How does Retrofit work

https://medium.com/mindorks/understand-how-does-retrofit-work-c9e264131f4a

[30] Introducción a la Arquitectura de una Aplicación Android con Jetpack Compose

https://umhandroid.momrach.es/introduccion-a-la-arquitectura-de-una-aplicacion-android-con-jetpack-compose/