UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA DEL SOFTWARE

Sistema móvil para la monitorización de contaminación acústica

Mobile system for monitorization of noise pollution

Realizado por
Jesús Martín Ruiz

Tutorizado por
Bartolomé Rubio Muñoz
Daniel Garrido Márquez

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2020

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

Sistema móvil para la monitorización de contaminación acústica

Mobile system for monitorization of noise pollution

Realizado por
Jesús Martín Ruiz

Tutorizado por
Bartolomé Rubio Muñoz
Daniel Garrido Márquez

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2020

Fecha defensa: julio de 2020

Resumen

A día de hoy, según múltiples estudios de las más importantes organizaciones mundiales en el sector de la salud, se ha descubierto que la contaminación acústica es uno de los factores más perjudiciales en la vida humana y que puede derivar en enfermedades de todo tipo e incluso provocar la muerte.

Por ello, se dedican programas para estudiar la cantidad de decibelios a los que nos enfrentamos diariamente y comprobar que se encuentran en una escala buena parea nuestra salud.

El principal inconveniente de estos programas es que suelen ser bastante caros, ya que se necesitan equipos especializados para las lecturas de los correspondientes datos y éstas se realizan muy espaciadas temporalmente pudiendo llegar a pasar años entre lectura y lectura.

Para solucionar este problema, se ha desarrollado un sistema que permite a cualquier persona aportar su granito de arena en la lectura y medición de datos, mediante un sistema que sigue el esquema del Internet de las Cosas haciendo uso de componentes muy movibles y baratos.

Palabras clave: IoT, sensor, sonido, contaminación

Abstract

Nowadays, several researchs from the most important organizations in the health sector have ended up in the same point, acoustic contamination is one of the most harmful factors in the human daily life and that can produce any type of illness to us, even the death.

That is why, lots of new programs are being created to study the quantity of sound we heard every day and check that they are healthy for our system.

The main inconvenience to those programs is that they’re usually very expensive, because they need specialized tools and they are taken very occasionally, some times it can take years to get new sound data.

In order to find a solution, we have developed a system that allows any person to help in the process of data reading, with our project, that follows the Internet of Things structure, using very cheap and movable components.

Keywords: IoT, sensor, sound, contamination

Índice

Table of Contents

1

Introducción

1.1 Motivación

Hoy por hoy, según la Organización Mundial de la Salud, la contaminación acústica es uno de los factores ambientales que más cantidad de enfermedades produce, y que puede suponer un problema en nuestra calidad de vida, provocando todo tipo de problemas, desde sordera e insomnio hasta ataques al corazón y cada vez se está consagrando como uno de los mayores problemas de las grandes ciudades.

Actualmente, el Gobierno de España financia un programa para la medición de datos sobre contaminación acústica, que produce mapas donde se muestran los datos de ruido de cada zona medida.

No obstante, los datos recogidos por este tipo de sistemas puede que no sean todo lo útiles que cabría esperar, puesto que un problema de estas mediciones es la periocidad en que se realiza la lectura de datos, cada 5 años, que evita poder ver datos en tiempo real o relativamente recientes. Además, estas lecturas se realizan en localizaciones muy concretas, quedando exentos de lectura muchos puntos de interés. Por último, hay que tener en cuenta que se trata de un sistema costoso que implica gastos considerables de material y personal.

Por ello, nuestra principal motivación es conseguir crear un sistema barato y móvil que permita la lectura de datos de ruido en tiempo

real, el envío y almacenamiento de dichos datos en un servidor remoto, que además pueda representar los mapas de ruidos elegidos por los clientes del sistema.

1.2 Objetivos

Este proyecto tiene como objetivo principal la creación de una aplicación basada en la estructura Internet de las Cosas (*IoT*) [1] que nos permita realizar un funcionamiento similar que el proyecto estatal a un coste mucho más reducido.

La idea es conseguir que esta aplicación funcione mediante una red de personas voluntarias, en un modelo que se conoce como *People as Sensors* [2], cada una con un Sistema lector y un Dispositivo móvil [Figura 1.1] y se pueda visualizar gráficamente los datos de ruido en tiempo real de distintas zonas de una ubicación, dependiendo de los datos enviados al servidor en la nube por los usuarios.

Figura 1.1 Diagrama representativo de la estructura del sistema final

A continuación, se describen brevemente cada una de las partes que constituyen la aplicación:

- Sistema lector: en nuestro caso se tratará de una placa Arduino, que constará de un sensor de sonido que se encargará de leer los datos. Además, posee un módulo Bluetooth, que nos permitirá el envío de datos a corta distancia, en este caso hacia el dispositivo móvil.

- Dispositivo móvil: se trata de un teléfono móvil, con sistema operativo Android, y con una aplicación, que habremos desarrollado previamente, que permita conectarse al Sistema lector y recibir los datos. Esta misma aplicación será la encargada de enviar esos datos, mediante la red móvil (internet) a nuestro Servidor.

- Base de Datos: se utilizará una base de datos remota en la nube, que nos permitirá almacenar los datos leídos.

- Servidor: su función es recibir las peticiones del Dispositivo móvil y guardarlos en una base de datos, en un formato correcto. Además, enviará esos datos a cualquier Cliente que se lo solicite.

- Cliente: es una aplicación web que se encargará de mostrar gráficamente los datos almacenados en la base de datos, recibidos del Servidor.

Se puede resumir en que el objetivo principal es la representación gráfica de los datos leídos por diversos de sistemas lectores móviles, tanto en tiempo real como los recientes.

1.3 Estructura de la memoria

Esta memoria está dividida en siete capítulos que recopilan toda la información que es necesaria para entender cómo y qué se ha desarrollado en este Trabajo de Fin de Grado. Para comprender la estructura de esta memoria, ofreceremos un resumen de cada uno de los capítulos.

En el Capítulo 1, actual, se describen tanto el problema que hemos observado como la solución propuesta para solventarlo.

En el Capítulo 2, se explican cada una de las tecnologías utilizadas durante el desarrollo del proyecto.

En el Capítulo 3, se especifica la metodología utilizada durante el tiempo en que se ha desarrollado el producto, así como las características principales de la misma y cómo se ha aplicado a nuestro proyecto en concreto.

En el Capítulo 4, se ofrece un análisis más técnico del sistema, donde incluiremos tanto los requisitos capturados en la frase previa al desarrollo como los casos de uso, de secuencia y de clases en que nos hemos basado.

En el Capítulo 5, se describe el diseño del sistema, en apartados como se ha diseñado y estructurado la base de datos y la interfaz.

En el Capítulo 6, se especifica más en detalle cómo se han implementado cada una de las partes de nuestro sistema final.

En el capítulo 7, se presenta una descripción de cómo se ha probado cada uno de los módulos de nuestra aplicación para comprobar su funcionamiento.

En el capítulo 8, se describe la conclusión final de nuestro proyecto y se ofrecen algunas ideas sobre cómo extender el proyecto desarrollado.

2

Tecnologías

2.1 Arduino

Arduino [3] es una plataforma electrónica que se podría resumir en una placa con un microcontrolador. Además, la placa también posee varios pines GPIO que permiten la incorporación de nuevos componentes como LEDs, pulsadores, sensores, etc.
La simplicidad de su sistema permite infinidad de posibles expansiones y sistemas que construir, y es por ello que a día de hoy es utilizado en un sinfín de proyectos. Sobre todo, está cogiendo mucha importancia en proyectos IoT por su fácil acoplamiento con otros sistemas.

2.1.1 Arduino IDE

Arduino utiliza C++ como lenguaje de programación, aunque simplificado. Para ello, se escribe sobre archivos de extensión .ino.
Arduino IDE es un proyecto software *open-source* que simplifica la escritura, compilación y subida de programas a la placa. Es multiplataforma y puede ser usado en Windows, Mac OS X y Linux.
Este IDE permite compilar los archivos .ino y los sube automáticamente a nuestro dispositivo Arduino.

2.1.2 Arduino Mega 2560

La placa Arduino Mega 2560 [Figura 2.1] es una de las más potentes de Arduino, basada en el microcontrolador Atmega2560 y es el modelo superior al Arduino Mega.
La placa contiene 54 pines digitales y 16 pines analógicos, junto a 4 puertos serie. Tal cantidad de pines permiten al usuario muchas variedades de expansión sin necesidad de una protoboard, por lo que hace el sistema mucho más móvil.

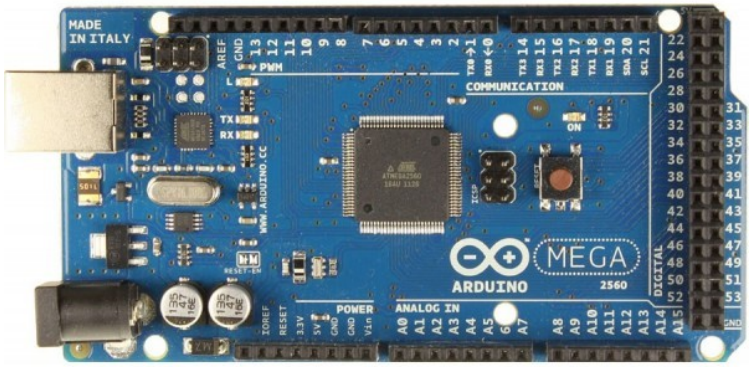

Figura 2.1. Placa Arduino Mega 2560

2.1.3 Módulo Bluetooth HC-06

El HC-06 [Figura 2.2] es un módulo Bluetooth esclavo, es decir, sólo recibe conexiones. Es el estándar utilizado en Arduino para establecer conexiones de baja frecuencia y permite la transmisión de datos entre sistemas.

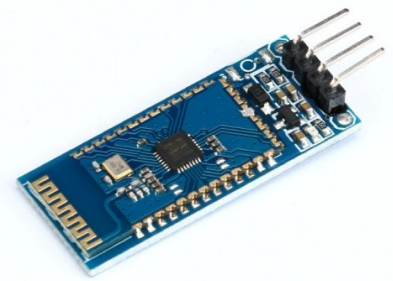

Figura 2.2. Módulo Bluetooth HC-06

2.1.4 Módulo Sensor Acústico

El módulo de sonido utilizado es el *Adafruit Silicon MEMS Microphone Breakout – SPW2430* [Figura 2.3]. Un sensor ecónomico, cuyo precio ronda los 5€, y de tamaño reducido que nos permitirá más

desplazamiento. Su principal característica es que es capaz de detectar sonidos en un rango de 100Hz - 10KHz.

Figura 2.3. Módulo Sensor Acústico Adafruit Silicon MEMS Microphone Breakout - SPW2430

2.2 Node.js

Node.JS [4] es un entorno en tiempo de ejecución multiplataforma de código abierto que se basa en JavaScript. Diseñado para construir aplicaciones de red escalables, como por ejemplo servidores web.

2.2.1 Express.js

Express.js [5] es un framework de aplicaciones web para Node.js, que se autodescribe como mínimo y flexible. Esta tecnología es *open-source* y permite la creación de APIs.

2.3 Open Street Maps

Open Street Maps [6] es un mapa del mundo creado bajo licencia abierto, es decir, que puede ser usado gratuitamente por todo el mundo. Es la principal alternativa libre a Google Maps.

2.3.1 Leaflet

Leaflet [7] es una librería JavaScript *open-source* que permite la creación de mapas interactivos, se usa junto a Open Street Maps y posee multitud de plugins que permite extender su interactibilidad.
Entre sus posibilidades de interacción está la de producir una capa de calor para el mapa.

2.4 React

React.js [8] es una de las librerías JavaScript más importante del mundo (sino la que más). Fue creada por Facebook y actualmente es de código abierto.

Permite la creación de interfaces de usuario basándose en JavaScript, su estructura es *SPA* (Single Page Application), por lo que es muy eficiente tras la carga inicial de la aplicación.
A día de hoy es una de las tecnologías más populares por la comunidad informática para la creación de aplicaciones web.
Para la creación de módulos visuales, se utilizan componentes, que se basan en JSX, una extensión de módulos JavaScript que permite añadir código HTML en nuestro fichero .js (ahora llamado .jsx) y que pasa a ser renderizado posteriormente en el DOM (Document Object Model) de la web.

2.4.1 TypeScript

TypeScript [9] es un superset tipado de JavaScript que nos permite compilarlo en JavaScript plano. Es muy popular para trabajar profesionalmente en proyectos que impliquen el uso de JavaScript ya que le añade uno de los puntos más importantes que le faltan: los tipados, las enumeraciones, los genéricos, etc.
En nuestro caso hemos optado por TypeScript con nuestra aplicación React ya que el código queda mucho más limpio y es mucho más legible a posteriori.

2.4.2 Material-UI

Material-UI [10] es un proyecto *open-source* de Google que ofrece componentes para React siguiendo la implementación de Material Design, también del gigante de la web.
Ofrecen componentes prediseñados y muy configurables para la creación de páginas web, es ideal para realizar una interfaz de usuario bonita y de una manera sencilla.

2.5 Flutter

Flutter [11] es un Kit de Desarrollo de Software (SDK) creado por Google y que permite la creación de aplicaciones móviles, web y escritorio con un mismo código. Es ideal para crear aplicaciones móviles ya que puedes obtener la aplicación tanto para iOS como para Android a partir del mismo código fuente. Es una tecnología muy nueva, ya que la versión 1.0 fue lanzada en Diciembre de 2018. Se basa en el lenguaje Dart, también desarrollado por Google.
Incorpora componentes prehechos siguiendo también el Material Design de Google, aunque también incorpora diseños de Cupertino (Apple) si lo deseamos.
Los componentes son llamados widgets y permiten ser actualizados mediante estados.

2.5.1 Dart

Dart [12] es un lenguaje programación moderno y multiparadigma de Google optimizado para la parte del cliente, y es principalmente utilizado para crear aplicaciones con Flutter.

2.6 Git

Git [13] es un sistema de control de versiones gratuito y *open-source* globalmente utilizado en el mundo de la tecnología para almacenar proyectos y tener flexibilidad en el desarrollo, poder ver y volver a puntos anteriores del proyecto.
Es utilizado prácticamente por todas las empresas más potentes del sector, como Google, Facebook, Microsoft, Twitter, Netflix, etc. y en los proyectos más importantes como Android, Linux, Gnome, etc.

2.6.1 GitHub

Es una compañía que ofrece *hosting* para proyectos de desarrollo de software usando el sistema de control de versiones *git*. Nos permite tener almacenados nuestros repositorios en la nube, además, podremos ver tanto nuestros proyectos como los públicos de otros usuarios mediante su cliente web, desde el que también se permite la subida de nuevos archivos y la descarga de repositorios.

2.7 MongoDB

Es una base de datos orientada a documentos, es decir, *NoSQL*, y que es usada para tratar grandes cantidades datos debido a su gran escalabilidad y flexibilidad.
Mongo se descompone en colecciones y documentos. Los documentos son simplemente archivos que almacenan información siguiendo la estructura de clave-valor, misma estructura que se utiliza en el globalmente utilizado *JSON* (*JavaScript Object Notation*).
Las colecciones son simplemente contenedores de múltiples documentos.

2.7.1 MongoDB Atlas

MongoDB Atlas es un servicio de base de datos en la nube de la propia empresa. Es una herramienta que nos ofrece disponibilidad y

escalabilidad con el sello de una de las empresas muy puntera del sector.
La misma plataforma nos ofrece un plan gratuito para poder *hostear* una base de datos, y en este caso es el que hemos usado para tener una base de datos en la nube y no depender de ninguna instancia local.

2.8 Visual Studio Code

Visual Studio Code es un editor de código creado por Microsoft y que hemos utilizado para desarrollar con JavaScript (Node.JS y React) y con Dart (Flutter). Es muy útil y extensible ya que contiene una gran cantidad de extensiones creadas por la comunidad que permiten desarrollar y *debuggear* en cualquier lenguaje, por lo que hemos cumplido el propósito de desarrollar con multiples *framework* y lenguajes en un mismo programa. Actualmente es una de las grandes alternativas a los entornos de programación.

2.9 Heroku

Es una plataforma como servicio (*PaaS*), es decir, su producto es una plataforma, en este caso para el despliegue de aplicaciones. Soporta múltiples lenguajes, entre ellos Node.js, Java, Python, etc.
La empresa Heroku también permite al usuario utilizar algunos de sus servicios de forma gratuita, y es por ello que la hemos utilizado para desplegar nuestro *back-end* en la nube de forma gratuita. Para su despliegue se realiza utilizando *git* y subiendo nuestro proyecto al repositorio ofrecido por heroku.

2.10 Netlify

Netlify es también una compañía donde uno de sus productos es el ofrecer *PaaS*. Y es por ello que ofrece *hosteo* para aplicaciones web y páginas web estáticas. Actualmente es usada por compañías punteras del sector como lo son Google, Facebook, Verizon, Nike, Samsung, etc.
Y también tiene un plan gratis para subir nuestros propios proyectos, en este caso, lo hemos utilizado para subir nuestro cliente con React.
Para subir los proyectos se utiliza *git*, pero en este caso nos permite enlazar un repositorio almacenado en otros servicios como *GitHub* y *GitLab*, proveyendo, además, un despliegue automático cada vez que se actualiza el repositorio.
Hemos utilizado este servicio enlazado a nuestro repositorio de *GitHub*.

3

Metodología

3.1 Definición del Proceso Unificado de Desarrollo Software

Para el desarrollo de este proyecto se ha decidido seguir la metodología de Proceso Unificado para el desarrollo de software.

3.1.1 Proceso iterativo e incremental

El Proceso Unificado tiene como sus dos principales propiedades que es iterativo e incremental, es decir, se realiza en base a iteraciones que resultan en un incremento del producto.
Las ventajas de este tipo de metodología es que nos permitirá ir viendo el avance del proyecto en el tiempo. Finalmente, con ello tendremos un producto generado tras cada iteración que puede evaluarse e ir comprobando que se están cumpliendo los requisitos.

3.1.2 Dirigido por los casos de uso

El Proceso Unificado, se centra en los casos de uso, que son utilizados para capturar y definir los requisitos del sistema, así como para establecer los objetivos de las distintas iteraciones de desarrollo. Cada una de estas iteraciones se encargarán de implementar uno o varios casos de uso, pasando cada iteración por todas las fases del desarrollo de software: diseño, implementación, pruebas…

3.1.3 Centrado en la arquitectura

Esta metodología asume que existen múltiples bloques que definen el sistema, y que es necesario ir construyendo uno sobre otro para obtener el resultado final esperado. Se puede asimilar al proceso de construcción de un edificio, que tiene varios bloques: arquitectura, albañilería, electricidad...

3.2 Fases del Proceso Unificado

El Proceso Unificado de Desarrollo de Software se divide en cuatro fases: Inicio, Elaboración, Construcción y Transición [Figura 3.1].

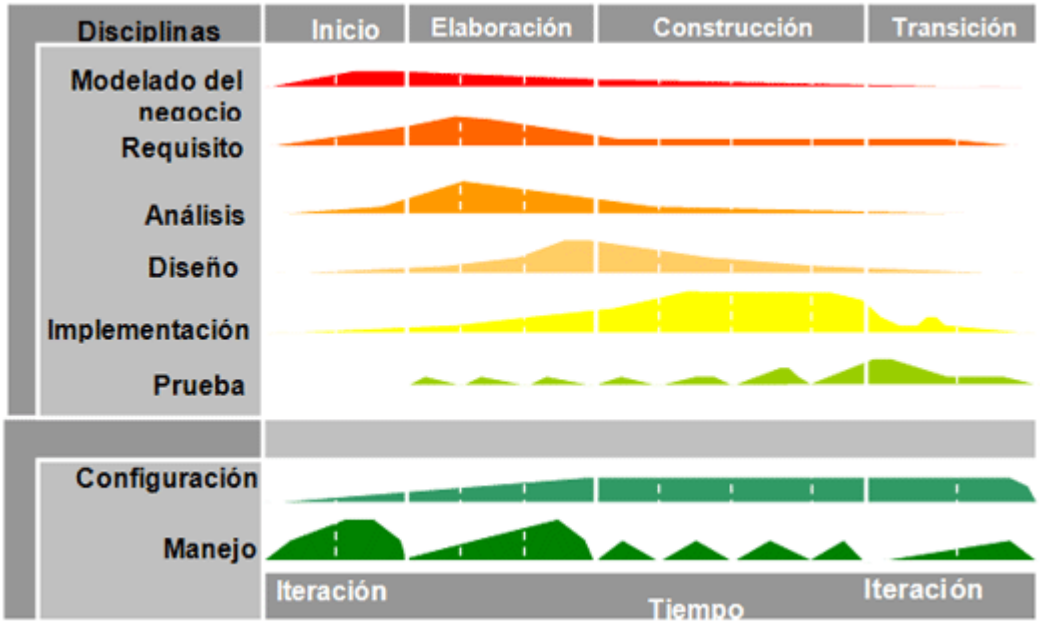

Figura 3.1 Fases del Proceso Unificado de Desarrollo de Software

3.2.1 Fase de Inicio

La primera fase se llama *Inicio*, es la que se encarga principalmente de realizar el análisis del modelo de negocio, es decir, comprobar la viabilidad del proyecto, calcular los costes, establecer las metas, se definen los plazos y se analizan los riesgos, entre otras múltiples acciones.

3.2.2 Fase de Elaboración

Esta etapa tiene como objetivo la acción de completar el análisis de los casos de uso y definir la arquitectura del sistema.

3.2.3 Fase de Construcción

Dicha fase implica la evolución del sistema hasta convertirse en un producto listo que incluya todos los requisitos, es decir, la construcción de una primera versión final completa.

3.2.4 Fase de Transición

En esta última fase, es la que se encarga de transformación del producto hasta la versión final del producto que se entrega a los clientes para su consecuente utilización.

3.3 Planificación de tareas

Una de las partes más importantes a la hora de planificarlos es usar una herramienta que nos permita registrar todas las tareas en realización, realizadas y por realizar. Para ello, hemos utilizado la herramienta Trello [Figura 3.2], que nos permite añadir un tablero con tarjetas para nuestras tareas con columnas para mover el estado de la tarjeta actual.

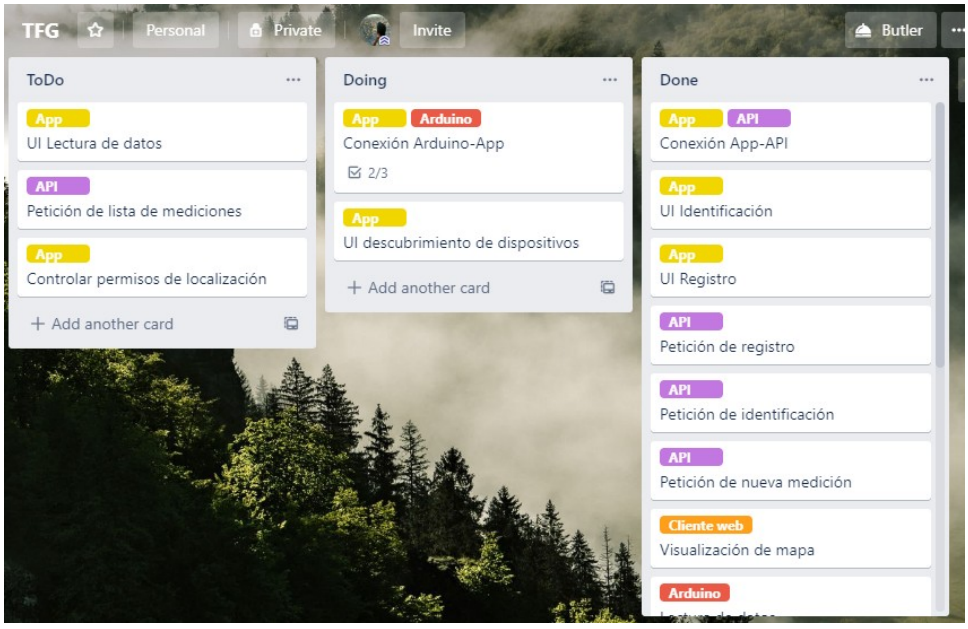

Figura 3.2 Tablero de tarjetas del proyecto

4

Análisis del sistema

4.1 Especificación de los requisitos del sistema

Para el desarrollo del sistema es primordial realizar una lista con las funcionalidades que debe implementar el producto final. Dicha lista estará dividida en dos bloques: los funcionales, que describen las acciones que debe realizar nuestro sistema y los no funcionales, que nos indican otras características y limitaciones que debe tener el sistema.

4.1.1 Requisitos funcionales

1. Gestión de usuarios
    1.1. Registro de usuario. El usuario puede registrarse en el sistema.
    1.2. Identificación de usuario. El usuario puede identificarse con una cuenta previamente creada.
2. Conexión de dispositivos

2.1. Conexión con Arduino. El usuario de un dispositivo móvil debe poder conectarse con la placa mediante Bluetooth.
2.2. Desconexión con Arduino. El usuario de un dispositivo móvil puede finalizar la conexión con la placa.

3. Gestión de mediciones
   3.1. Iniciar envío de datos. El usuario de un dispositivo móvil puede enviar los datos del sonido al servidor web.
   3.2. Finalizar envío de datos. El usuario de un dispositivo móvil debe poder finalizar el envío de mediciones.
   3.3. Visualización de las mediciones. El usuario del cliente web y móvil puede visualizar las mediciones en un mapa de calor.
   3.4. Filtración por zonas. El usuario del cliente web debe poder filtrar las mediciones por zonas.
   3.5. Descarga de las mediciones. El usuario del cliente web debe poder descargar un fichero con todas las lecturas.

4.1.2 Requisitos no funcionales

1. Encriptación de contraseñas. El sistema no puede almacenar datos sensibles sin cifrar.
2. Diseño *responsive*. El sistema debe mantener el diseño para todas las plataformas.
3. Multilenguaje. El sistema debe soportar múltiples lenguajes: español e inglés.

4.2 Casos de Uso

4.2.1 Diagramas de Casos de Uso

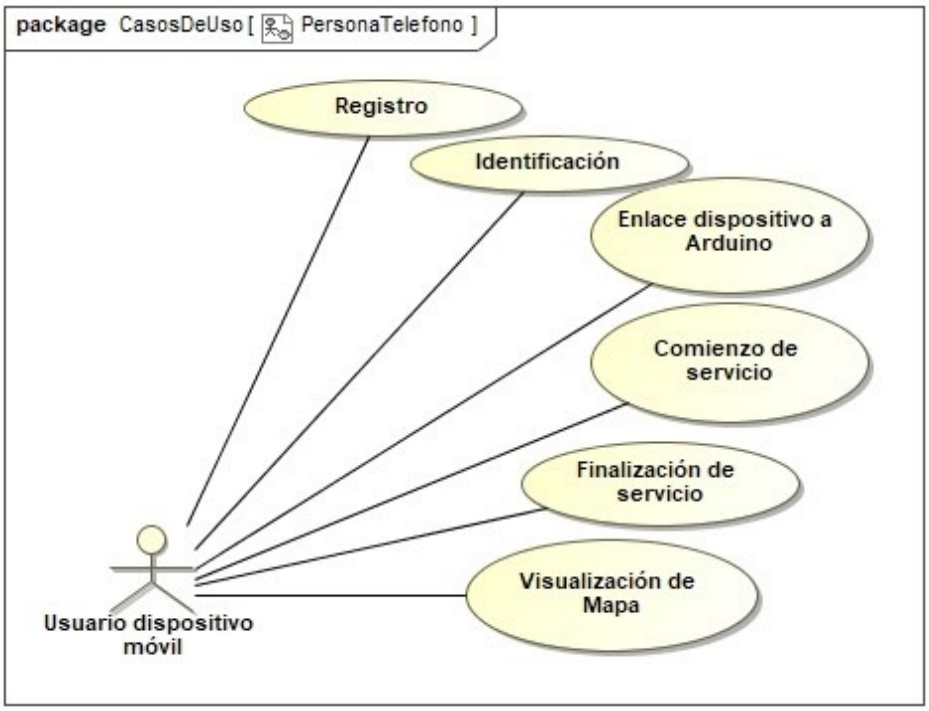

Figura 4.1 Diagrama de Casos de Uso para un Usuario de un dispositivo móvil

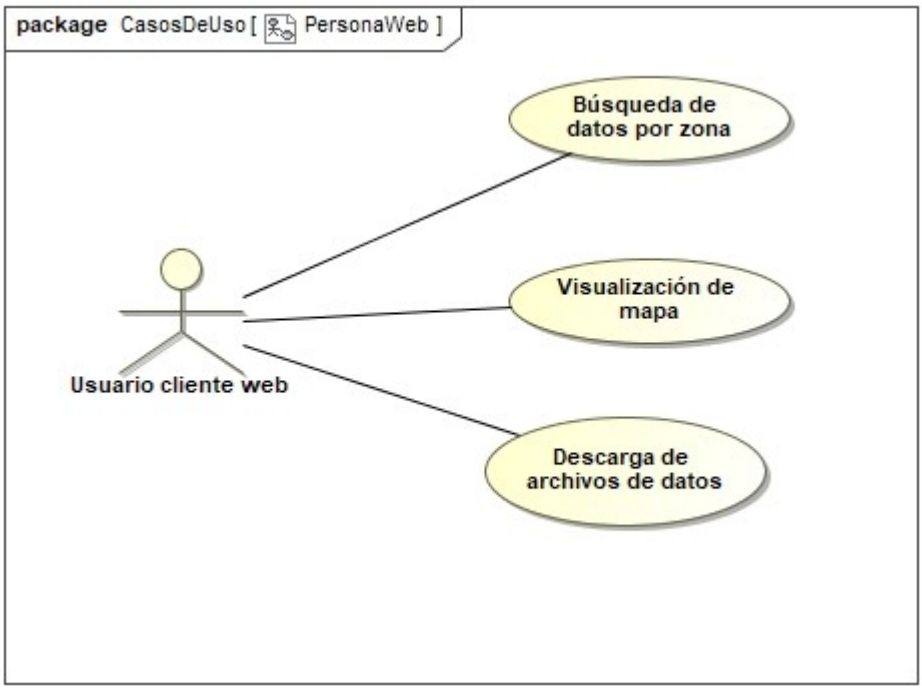

Figura 4.2 Diagrama de Casos de Uso para un Usuario del cliente web

4.2.2 Especificación de Casos de Uso

Caso de Uso: 1. Registro de usuario.
Actor principal: Usuario de dispositivo de móvil.
Precondiciones: El usuario no ha iniciado sesión.
Garantía de éxito: - El usuario se encuentra en la pantalla principal. - El usuario recibe un *toast* de registro con éxito.
Escenario de éxito: 1. El usuario abre la aplicación. 2. El usuario hace click en el botón de “Registro”. 3. El usuario introduce un nombre de usuario. 4. El usuario introduce una contraseña. 5. El usuario introduce de nueva la contraseña para cofirmarla. 6. El usuario hace click en el botón de “Enviar”.
Extensiones: 6a. El nombre de usuario ya está registrado Vuelve a 3. 6b. Las contraseñas no coinciden Vuelve a 4.

Tabla 4.1 Caso de uso. Registro de usuario.

Caso de Uso: 2. Identificación de usuario.
Actor principal: Usuario de dispositivo de móvil.
Precondiciones: El usuario no ha iniciado sesión.
Garantía de éxito: - El usuario se encuentra en la pantalla principal. - El usuario recibe un *toast* de identificación con éxito.
Escenario de éxito: 1. El usuario abre la aplicación. 2. El usuario hace click en el botón de “Identificarse”. 3. El usuario introduce su nombre de usuario. 4. El usuario introduce su contraseña. 5. El usuario hace click en “Enviar”.
Extensiones: 5a. Inicio de sesión erróneo. Vuelve a 3.

Tabla 4.2 Caso de uso. Identificación de usuario.

Caso de Uso: 3. Enlace de dispositivo Arduino
Actor principal: Usuario de dispositivo de móvil.
Precondiciones: - El usuario ha iniciado sesión. - El usuario se encuentra en la pantalla principal.

- El dispositivo móvil tiene el sensor Bluetooth activado. - El dispositivo Arduino está iniciado. - La aplicación móvil tiene permisos para usar Bluetooth.
Garantía de éxito: - El usuario se encuentra en la pantalla de inicio de servicio. - El dispositivo móvil se encuentra enlazado al Arduino mediante Bluetooth. - El usuario recibe un *toast* de enlazado con éxito.
Escenario de éxito: 1. El usuario selecciona el dispositivo Arduino de la lista de dispositivos disponibles mediante Bluetooth.

Tabla 4.3 Caso de uso. Enlace de dispositivo Arduino.

Caso de Uso: 4. Comienzo de servicio
Actor principal: Usuario de dispositivo de móvil.
Precondiciones: - El usuario ha iniciado sesión. - El usuario se encuentra en la pantalla de inicio de servicio. - El dispositivo móvil está enlazado al Arduino. - La aplicación móvil tiene permisos para usar la localización.
Garantía de éxito: - El usuario se encuentra en la pantalla de servicio en funcionamiento. - El usuario observa un mensaje en la pantalla “Leyendo datos...”.
Escenario de éxito: 1. El usuario hace click en el botón de “Comenzar servicio”.

Tabla 4.4 Caso de uso. Comienzo de servicio.

Caso de Uso: 5. Finalización de servicio
Actor principal: Usuario de dispositivo de móvil.
Precondiciones: - El usuario ha iniciado sesión. - El usuario ha iniciado el servicio. - El usuario se encuentra en la pantalla de inicio de servicio.
Garantía de éxito: - El usuario se encuentra en la pantalla de inicio de servicio.
Escenario de éxito: 1. El usuario hace click en el botón de “Finalizar servicio”.

Tabla 4.5 Caso de uso. Finalización de servicio.

Caso de Uso: 6. Visualización de Mapa
Actor principal: Usuario de dispositivo de móvil.
Precondiciones: El usuario ha abierto la aplicación
Garantía de éxito:

- El usuario visualiza el mapa de calor de sonido.
Escenario de éxito: 1. El usuario hace click en el botón del menú inferior con el icono del mapa.

Tabla 4.6 Caso de uso. Visualización de mapa en dispositivo móvil.

Caso de Uso: 7. Visualización del mapa.
Actor principal: Usuario de cliente web.
Precondiciones: -
Garantía de éxito: - El usuario visualiza el mapa de la aplicación.
Escenario de éxito: 1. El usuario abre en un navegador la página web del proyecto.

Tabla 4.7 Caso de uso. Visualización del mapa en cliente web.

Caso de Uso: 8. Búsqueda de datos por fecha.
Actor principal: Usuario de cliente web.
Precondiciones: - El usuario abre en un navegador la página web del proyecto.
Garantía de éxito: - El usuario visualiza el mapa de la aplicación de la fecha deseada.
Escenario de éxito: 1. El usuario hace *click* en el menú de la barra superior de la web. 2. El usuario introduce una fecha de inicio de filtrado. 3. El usuario introduce una fecha de final del fitrado. 4. El usuario hace *click* en el botón “Filtrar”.

Tabla 4.8 Caso de uso. Búsqueda de datos por zona.

Caso de Uso: 9. Generación de archivo de datos
Actor principal: Usuario de cliente web.
Precondiciones: - El usuario abre en un navegador la página web del proyecto.
Garantía de éxito: - Un archivo .JSON con los datos de sonido es descargado.
Escenario de éxito: 1. El usuario hace click en el icono de menú en la barra superior de la web. 2. El usuario hace click en el botón de “Descargar archivo”.
Extensiones: 3a. El navegador falla al descargar el archivo. Fin del caso de uso.

Tabla 4.9 Caso de uso. Generación de archivo de datos.

4.3 Diagramas de secuencia

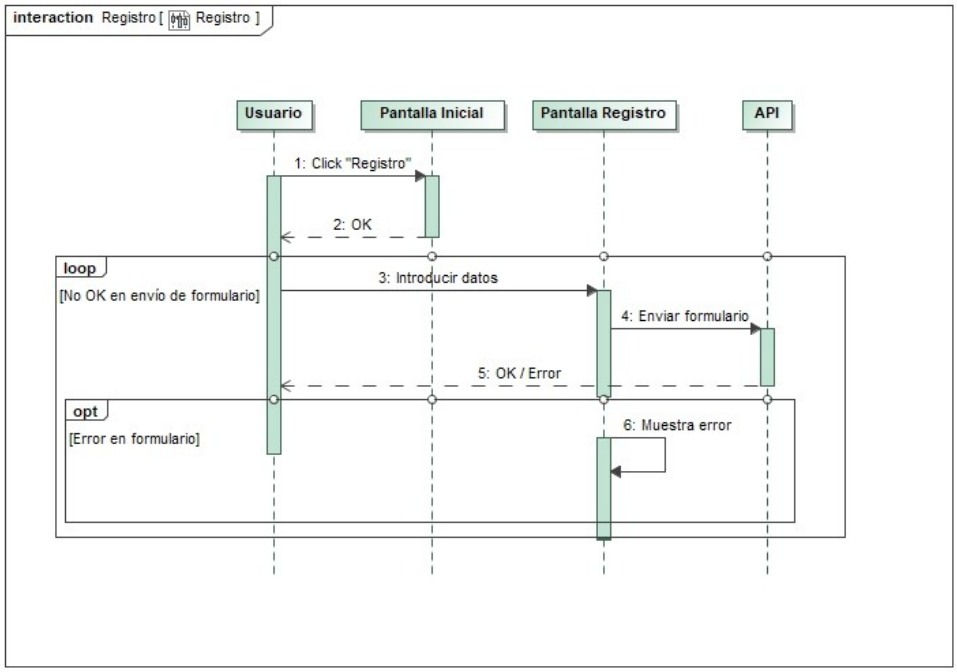

Figura 4.3. Diagrama de secuencia de Registro.

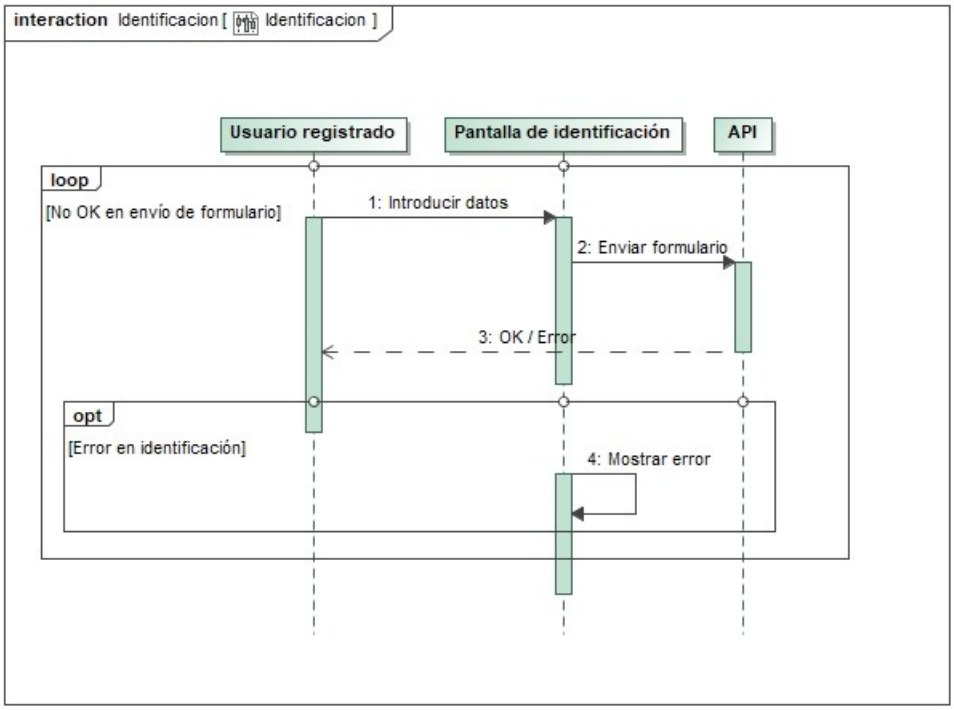

Figura 4.4. Diagrama de secuencia de Identificación.

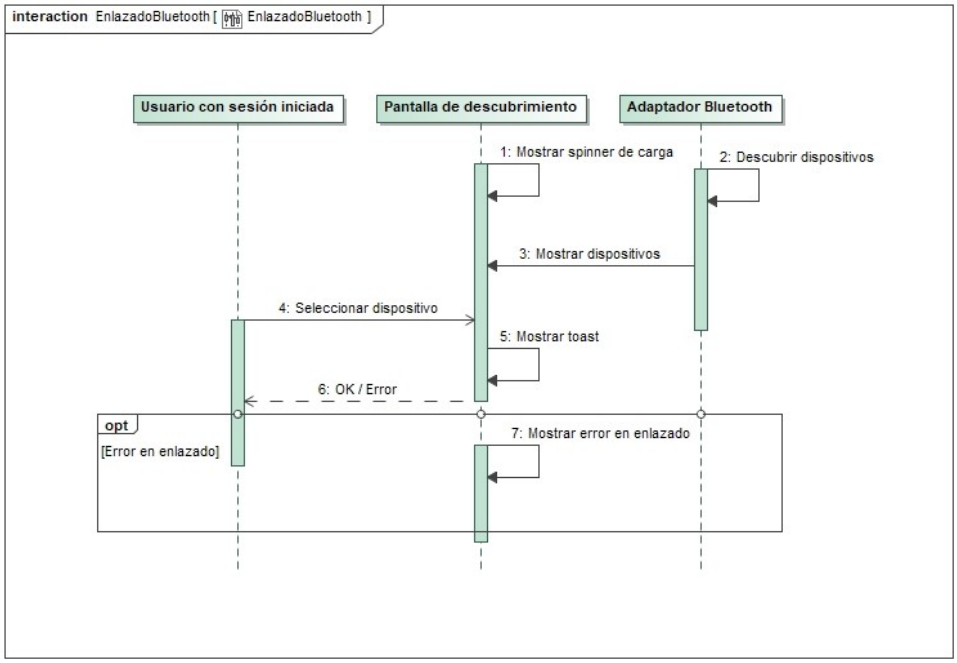

Figura 4.5. Diagrama de secuencia de Enlazado Bluetooth.

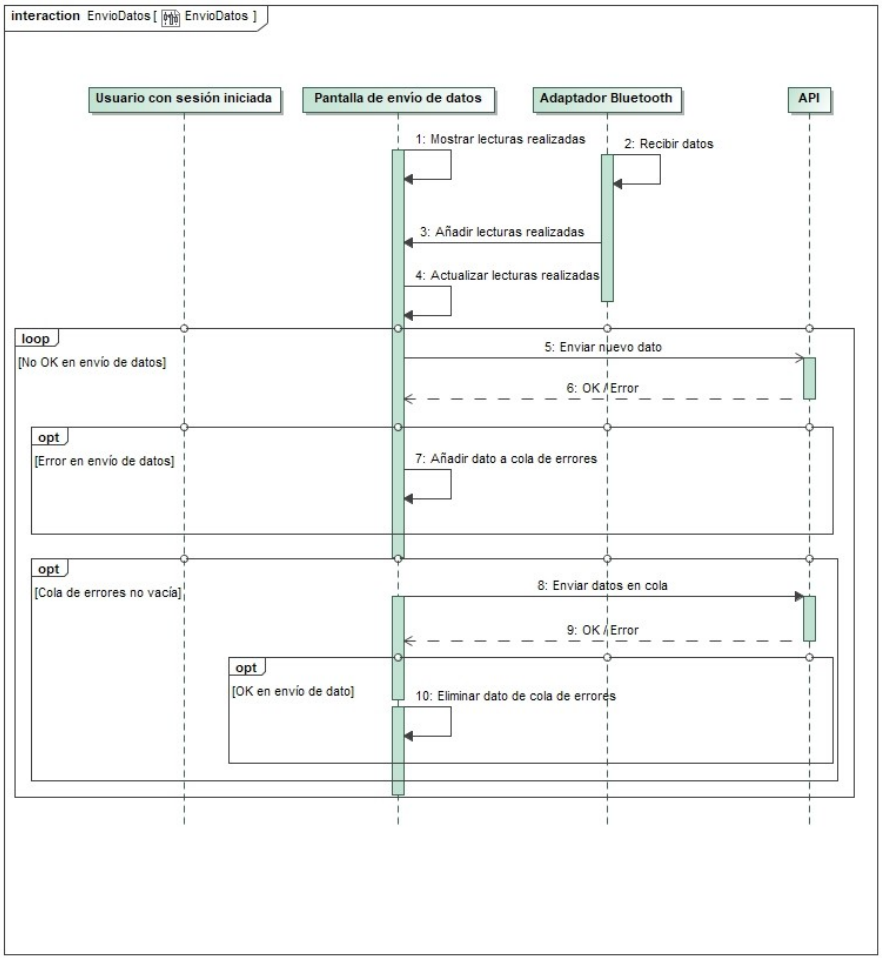

Figura 4.6. Diagrama de secuencia de Envío de datos.

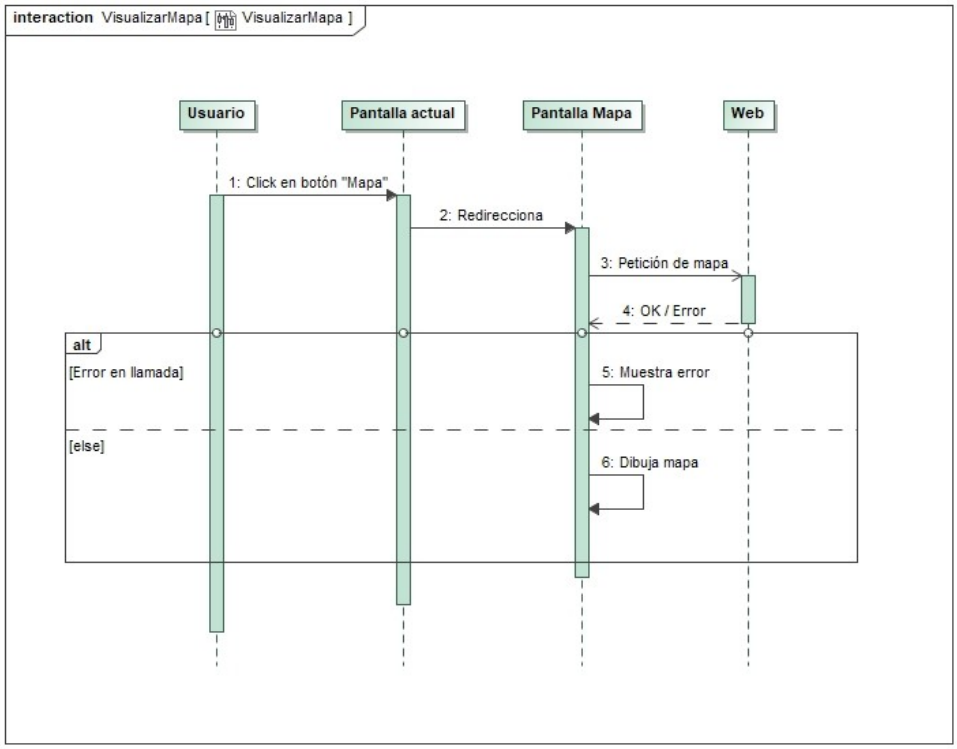

Figura 4.7. Diagrama de secuencia de Visualizar Mapa.

4.4 Diagrama de clases

En este apartado, vamos a mostrar los distintos diagramas de clases que se han seguido para la realización de la estructura de nuestras aplicaciones. Y en ellos se puede denotar la organización dentro de nuestras distintas aplicaciones.

4.4.1 Diagrama de clases de Aplicación móvil

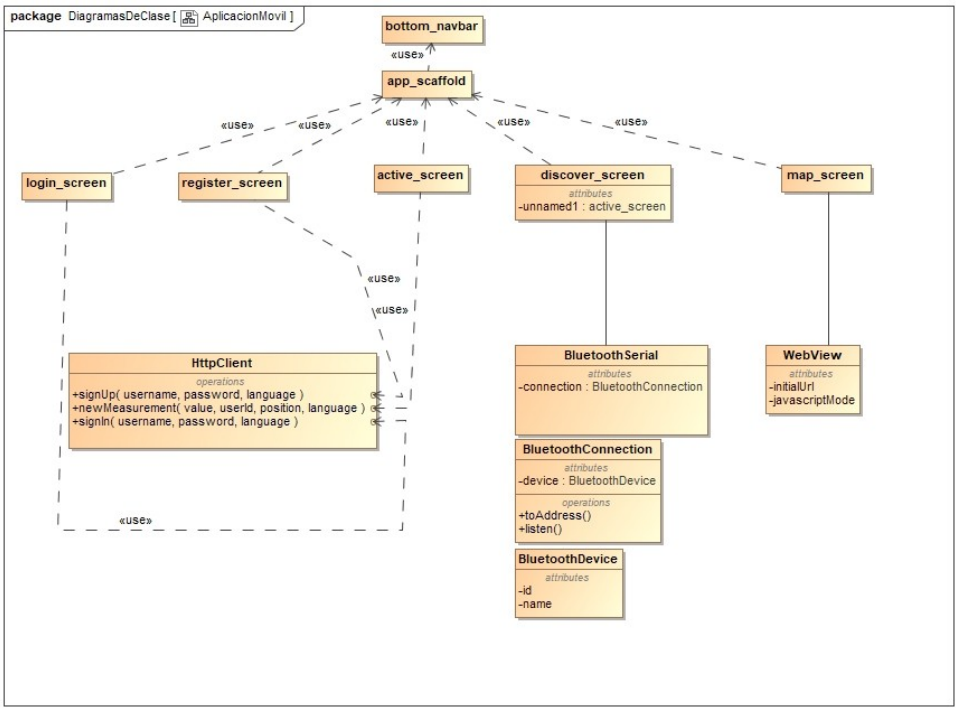

Figura 4.8 Diagrama de clases de aplicación móvil

4.4.2 Diagrama de clases de Cliente web

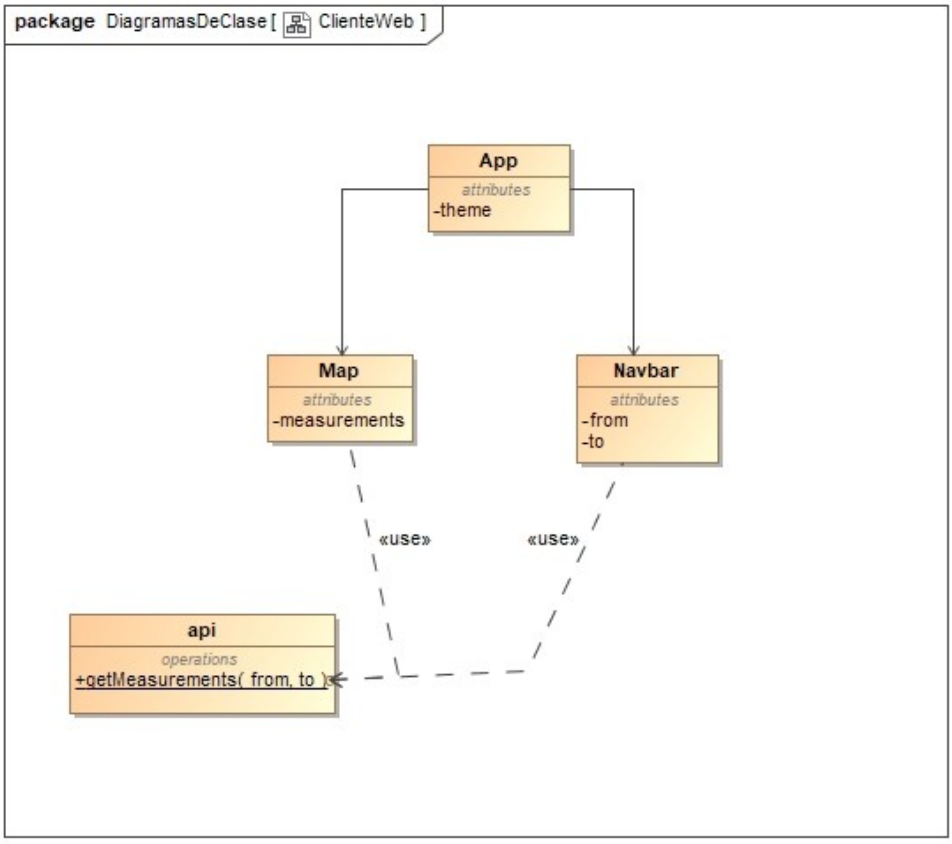

Figura 4.9 Diagrama de clases de cliente web

4.4.3 Diagrama de clases de Servidor web

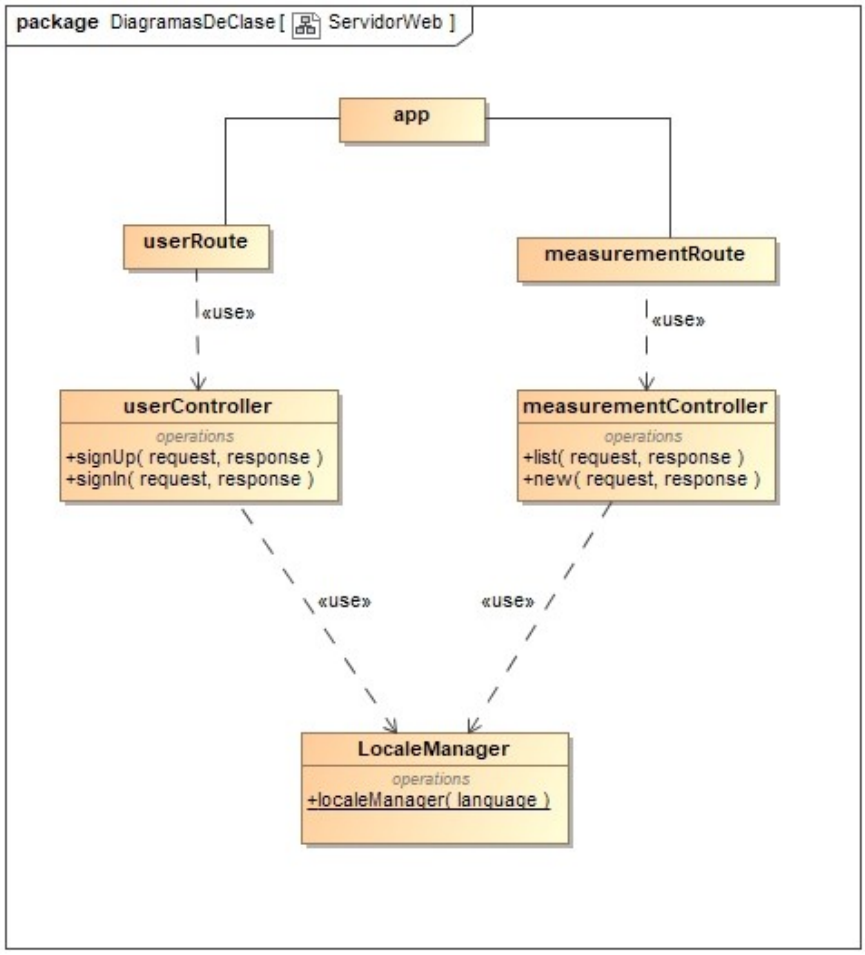

Figura 4.10 Diagrama de clases de servidor web

5

Diseño del sistema

5.1 Diseño de la base de datos

La base de datos del sistema, como ya hemos explicado previamente, es no relacional y por lo tanto lo que tendremos almacenados serán colecciones con documentos.
En nuestro caso, las colecciones que hemos almacenado son por una parte usuarios y por otra mediciones. En la primera de ellas, usuarios, se guardan cada uno de los usuarios que se han registrado dentro de la aplicación, para poder realizar el proceso de inicio de sesión con autenticación. Mientras, en las mediciones se guardan la localización y valor de cada lectura realizada por un usuario.
Aunque realmente una base *NoSQL* de MongoDB tiene un esquema libre y flexible de los datos, en nuestro caso al ser tratados previamente por un servicio externo, nuestra API, se guardan siguiendo una estructura predeterminada.
Cabe destacar que todos los documentos contienen una propiedad autogenerada por el propio Mongo *_id* que identificada y diferencia a cada uno de los documentos, se podría decir que es nuestra clave primaria.

Un usuario está compuesto por:

- *username*, básicamente el nombre identificativo del usuario como cadena de texto.
- *password*, la contraseña encriptada en formato cadena de texto.

Una medición, *measurement* [Figura 5.1], se compone de:

- *userId*, es el identificador *_id* del usuario que envió la lectura, es nuestra manera de tener identificados a los usuarios con las mediciones que se suben y evitar que estén anonimizadas y las acciones fraudulentas queden anónimas, es una cadena de texto.
- *position*, se trata de un objeto compuesto, que a su vez tiene dos propiedades: *latitude* y *longitude* que nos ofrecen la localización de la lectura. Ambos valores se guardan en formato numérico.
- *value*, es el dato que nos aporta el valor del sonido de la medición, es un valor numérico.
- *date*, es otro valor numérico que almacena la marca de fecha y hora del momento en que se realizó la lectura. El valor numérico corresponde al número de milisegundos desde el 01-01-1970 00:00:00 UTC, que es el valor que devuelve y utiliza JavaScript para las fechas.

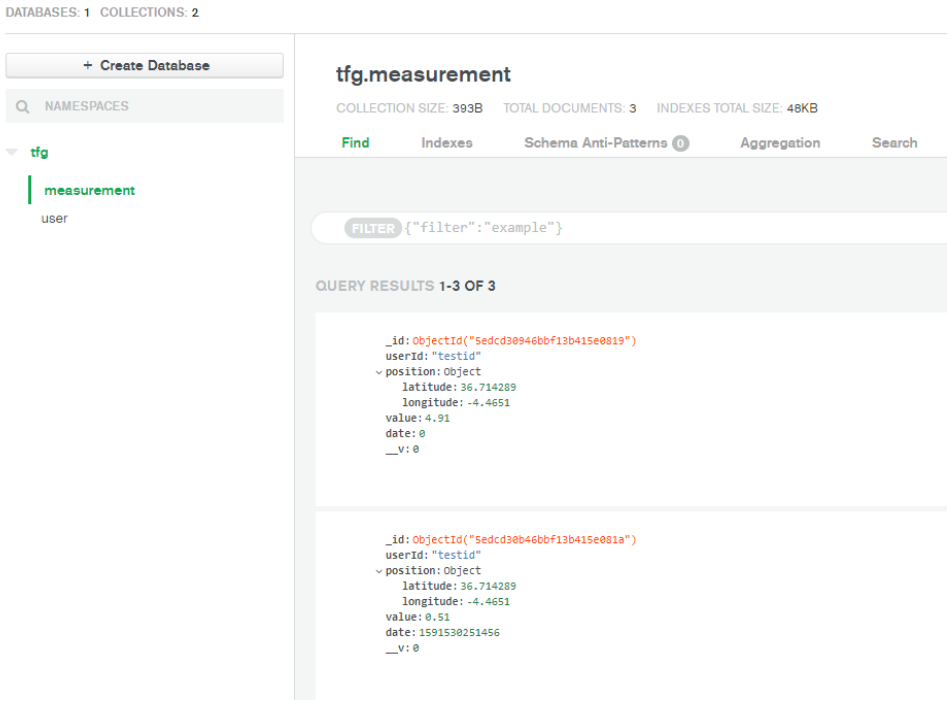

Figura 5.1 Estructura del objeto measurement almacenado

5.2 Diseño de la interfaz de usuario

En este apartado se mostrarán las maquetas de diseño que se han utilizado como base para el diseño de nuestra aplicación y qué funcionalidades deben aparecer en cada una de ellas.

5.2.1 Interfaz de la aplicación móvil

La interfaz de la aplicación móvil contiene siempre una estructura muy definida, para mantener la consistencia del diseño, los botones estar en posiciones muy similares, al igual con los demás elementos como campos de texto y títulos.

Como punto genérico cabe destacar que todas las interfaces, excepto la de lectura de datos, contienen una barra de navigación inferior que contiene dos enlaces, uno hacia el mapa de datos y otro que vuelve a la pantalla actual.

5.2.1.1 Inicio de sesión

El diseño del inicio de sesión [Figura 5.2] contiene por una parte como funcionalidad principal la identificación de un usuario para su posterior uso de la aplicación, usando varios campos de texto y un botón.

Además, como funcionalidad secundaria tenemos el enlazado a la pantalla de registro, mediante el uso de un segundo botón.

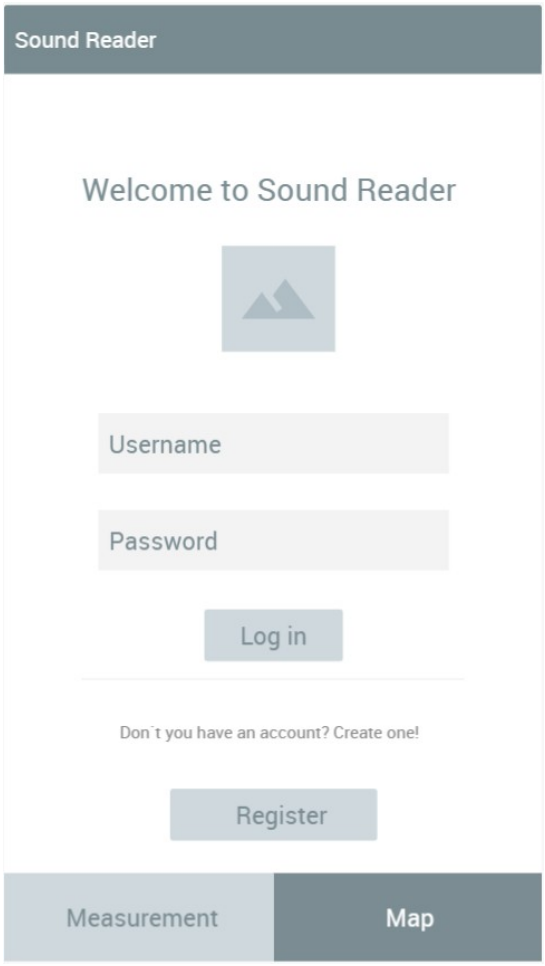

Figura 5.2 Interfaz de inicio de sesión.

5.2.1.2 Registro

La pantalla de registro [Figura 5.3] mantiene la estructura seguida en la pantalla de inicio de sesión. La funcionalidad primaria es el registro de un usuario.

Como principales diferencias tenemos tres campos, dos para la contraseña para verificarla y evitar fallos.
La navegación hacia atrás, pantalla de inicio de sesión, se realiza con un botón con icono de una flecha que se posiciona en la barra de navegación superior, así evitamos mostrar un botón inferior, ya que no es necesario.

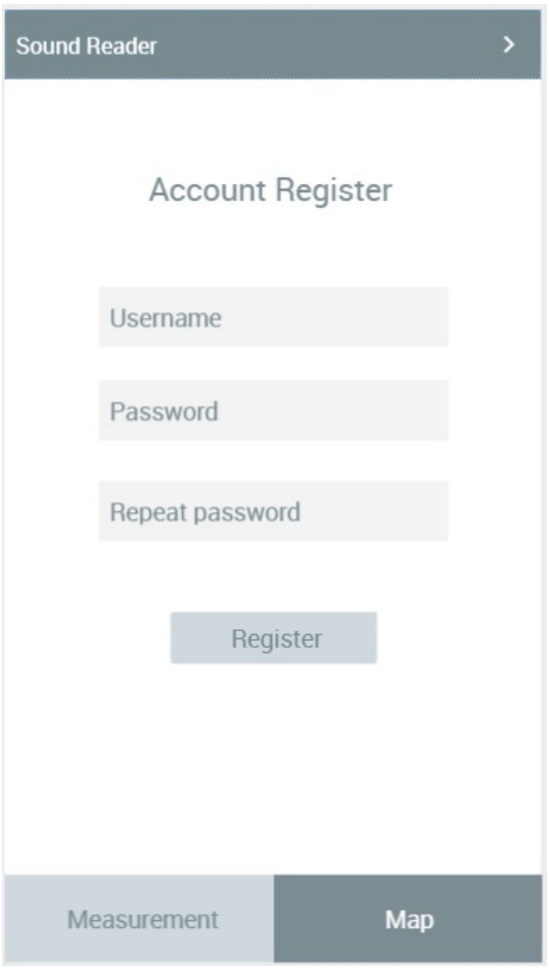

Figura 5.3 Interfaz de registro.

5.2.1.3 Descubrimiento de dispositivos

En esta interfaz de descubrimiento de dispositivos [Figura 5.4] el funcionamiento se basa en el mostrar los dispositivos Bluetooth cercanos y poder enlazarlos.

Para utilizar la funcionalidad explicada, se tiene una lista de elementos, que muestran tanto el nombre del dispositivo como el identificador. Además, se añade un botón de conexión alineado en el extremo derecho para poder realizar el enlazado.

Por otra parte, volvemos a tener un botón de navegación hacia atrás, localizado en la barra de navegación superior, que nos devolverá a la pantalla de inicio de sesión.

Por último, tenemos un elemento de alerta que se muestra en la parte inferior y que sirve de *feedback* para el usuario, ofreciendo información de carga o de fallos.

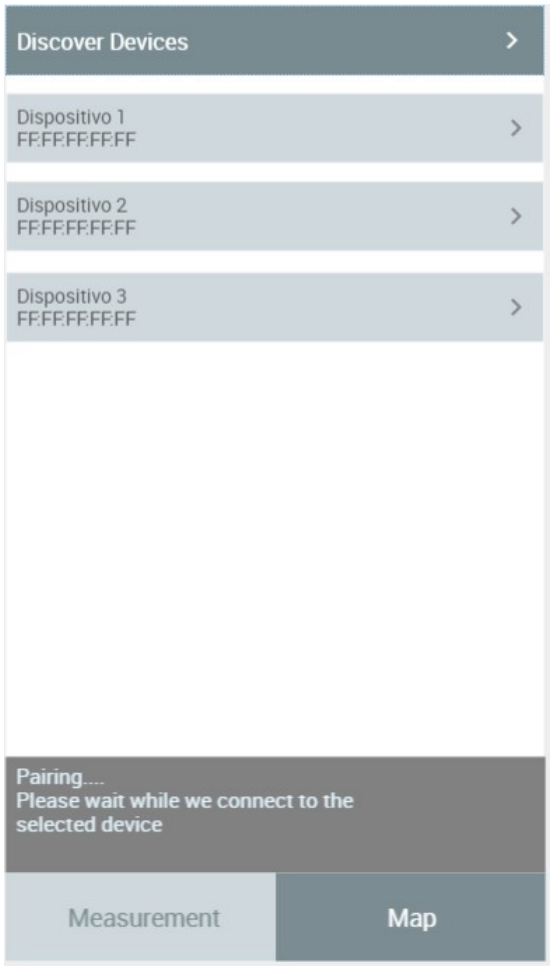

Figura 5.4 Interfaz de descubrimiento de dispositivos.

5.2.1.4 Lectura de datos

Para el diseño de la lectura de datos [Figura 5.5] se ha tenido en cuenta su principal funcionalidad, mostrar la cantidad de datos enviados para que el usuario pueda comprobar que todo funciona correctamente.

Se basa en un elemento que contiene y muestra el número de datos enviados y leídos, en la parte superior de la pantalla.

De nuevo volvemos a tener un botón de navegación, en la barra superior, que hará que el usuario pueda volver hacia la pantalla de descubrimiento de dispositivos.

Figura 5.5 Interfaz de lectura de datos.

5.2.1.5 Visualización de mapa

La pantalla de visualización del mapa [Figura 5.6], la principal y única funcionalidad es ver el mapa en tiempo real.

Esto se conseguirá añadiendo una ventana hacia el cliente web que ocupará todo el espacio disponible.

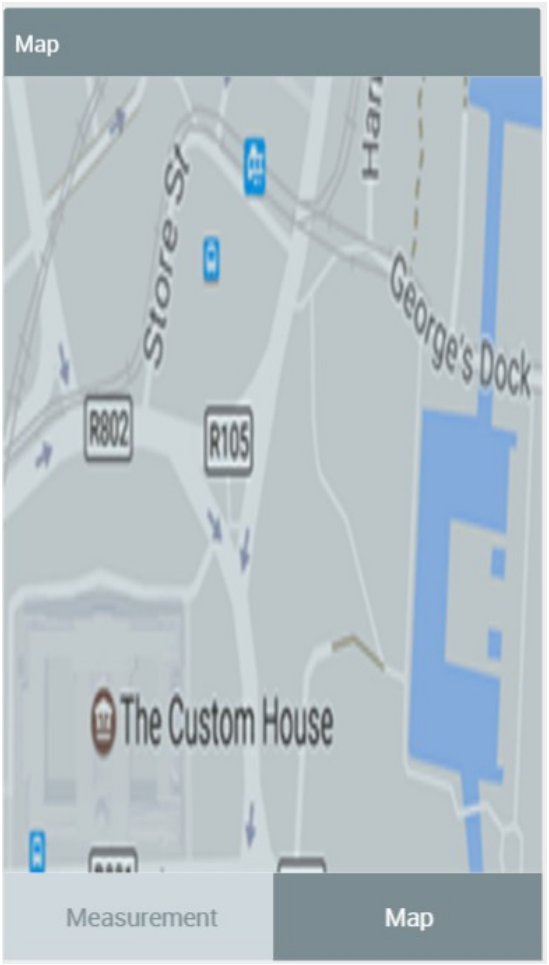

Figura 5.6 Interfaz de visualización de mapa.

5.2.2 Interfaz de la aplicación web

El diseño de la interfaz de usuario del cliente web [Figura 5.7] debe ser capaz de mostrar el mapa en tiempo real, filtrar los datos por fechas y descargar los datos.

Para conseguir los propósitos esperados se muestra una barra de navegación superior con el título de la aplicación a la izquierda y con un botón de menú en la parte derecha.

El menú de la parte derecha tiene como objetivo mostrar el menú de filtrado y el botón de descarga de archivo de datos, que se mostrará en un elemento que aparece sobre el contenido existente.

Y por otra parte, se mostrará un mapa en el espacio restante de la ventana.

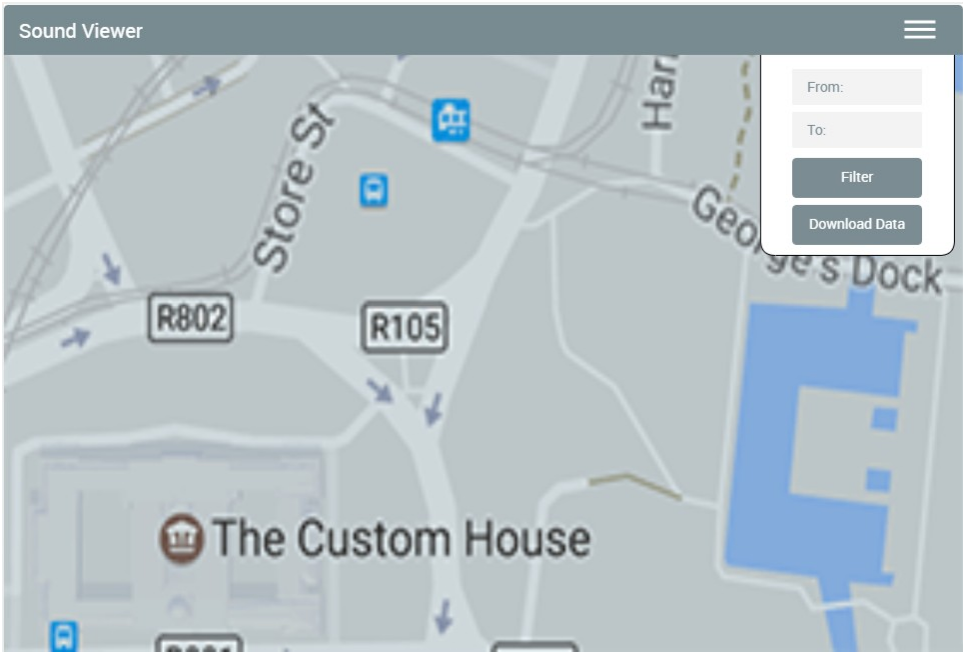

Figura 5.7 Interfaz de aplicación web

6

Implementación

6.1 Implementación de Arduino

6.1.1 Montaje

Para explicar el montaje [Figura 6.1] de nuestro sistema Arduino primero empezaremos explicando los módulos de sonido y de Bluetooth.

El módulo de Bluetooth se compone de 4 pines:

- VCC. Funciona como fuente de alimentación, este pin debe ir conectado al pin de 5V de nuestro Arduino para darle potencia a nuestro sensor.
- GND. Es el pin que va unido a tierra al también pin GND de la placa.
- RX. Es de recepción, es decir, el que se encarga de recibir los datos desde el Arduino, por lo que va enlazado a uno de los GPIO de envío de la placa, uno de los pines TX.
- TX. Es el de envío, realiza la acción de transmisión de datos hacia el dispositivo con el que tenemos un enlace. Se conecta a uno de los pines RX de nuestro Arduino.

El módulo de sonido está constituido por 5 pines:

- AC. Este pin es el que se encarga de transmitir los datos leídos por el sensor, se conecta a uno de los pines analógicos del Arduino.

- VCC. Del mismo modo que en el módulo Bluetooth, se encarga de darle corriente al sensor, en este caso, el módulo sólo necesita el pin de 3.3V para funcionar.
- GND. Igual que en módulo Bluetooth, es el pin de tierra y se conecta con la tierra de la placa.
- DC y Vin, estos dos pines no nos son necesarios para montar nuestro sistema.

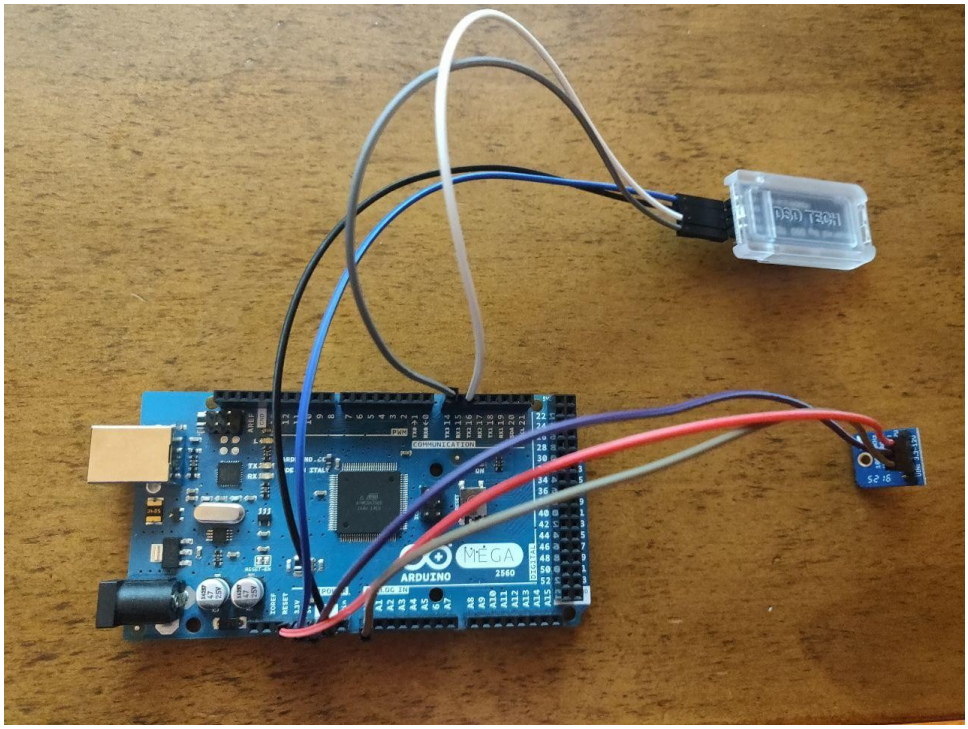

Figura 6.1. Montaje placa Arduino Mega.

6.1.2 Estructura del código

La estructura del código utilizado por una placa Arduino es bastante sencillo.
Se basa en tres partes básicas.

```
// Definiciones de constantes y variables globales
// Importación de librerías

// Método ejecutado al inicio del programa
void setup() {
      // Inicialización de módulos
}

// Método ejecutado infinitamente hasta finalizar la ejecución
void loop() {
      // Lectura de datos
      // Envío de datos
      // Acciones
}
```

6.1.2 Lectura

La lectura de datos haciendo uso de los pines de Arduino es bastante simple.

Primero, debemos de añadir el pin elegido, en nuestro caso el A0 (pin analógico 0), como de entrada. Para ello, lo añadimos en el método *setup()* mediante *pinMode(A0, INPUT);*

Y si queremos leer los datos del sensor, hacemos uso del método *analogRead(A0).* Que nos devolverá un valor entre 0 y 1024 que es el leído por el sensor.
Para tomar una referencia más simple y mostrar los datos en una escala más sencilla, se transforman los datos en voltios, que nos devolverá un valor entre 0 y 3.3, que será la escala en la que se mueva nuestros datos.

6.1.3 Bluetooth

Para la conexión Bluetooth hemos tenido que importar una librería propia de Arduino: *BluetoothSerial.h.*

Además definiremos dos constantes para los pines físicos que reciben la conexión del módulo: *BT_SERIAL_TX* y *BT_SERIAL_RX.*
También es necesaria la creación de una variable global de la clase *BluetoothSerial* que se inicializará utilizando los valores almacenados en las constantes previamente descritas TX y RX.

Esta misma variable global es necesario inicializarla en el método *setup*, mediante uno de sus métodos propios *.begin(9600)*, el valor del argumento es simplemente el número de hertzios a los que leerá nuestro sensor, se suele utilizar 9600 como valor estándar.

Y por último y más importante, para realizar el envío de datos en el método *loop*, se utilizará de nuevo nuestra variable global junto con su método propio *.println(data)* donde *data* es el valor que queremos enviar por Bluetooth.

Resumiendo, nuestro comportamiento para enviar datos mediante una conexión de baja frecuencia Bluetooth sería:

```
#import <BluetoothSerial.h>
#define BT_SERIAL_TX 14
#define BT_SERIAL_RX 15
SoftwareSerial BluetoothSerial(BT_SERIAL_TX, BT_SERIAL_RX);

void setup() {
```

```
  BluetoothSerial.begin(9600);
}

void loop() {
  int value = 0;
  BluetoothSerial.println(value);
}
```

6.2 Implementación de la aplicación móvil

6.2.1 Conexión Bluetooth

Para la conexión con el dispositivo Bluetooth, hemos utilizado una librería externa que nos permitirá interactuar con este tipo de conexión y así, descubrir otros dispositivos, conectarnos a un dispositivo concreto, recibir y enviar datos.

La librería es *Flutter Bluetooth Serial* [14], es una implementación básica del Bluetooth, para enlazarnos con RFCOMM, que es el usado por Arduino.

En este pequeño fragmento de código se puede observar la funcionalidad básica que hemos utilizado en nuestro proyecto.

1. Nos conectamos a un dispositivo Bluetooth dada una dirección (*address*), si no es posible, se lanza una excepción y se mostraría por pantalla un error en el enlazado.
2. Leeremos de la conexión cuando exista algún dato entrante con *connection.input.listen*, que se lee en tipo lista de *bits*, que podremos decodificar para obtenerlo en formato *ascii.*
3. Envío de datos mediante la misma conexión con el método *connection.output.add(data).*
4. La finalización de la conexión, donde simplemente se dará por terminado el enlace y se enviará una señal al dispositivo enlazado, se usa el método *connection.finish()*.

```
try {
    BluetoothConnection connection = await
BluetoothConnection.toAddress(address);
    print('Connected to the device');

    connection.input.listen((Uint8List data) {
        print('Data incoming: ${ascii.decode(data)}');
        connection.output.add(data);

        if (ascii.decode(data).contains('!')) {
            connection.finish(); // Closing connection
            print('Disconnecting by local host');
        }
    }).onDone(() {
        print('Disconnected by remote request');
    });
```

```
}
catch (exception) {
    print('Cannot connect, exception occured');
}
```

6.2.2 Conexión servidor web

Para la conexión con el servidor web, en este caso, hemos utilizado de nuevo una librería externa, una de las más utilizadas y populares de Flutter, *http* [15].

Esta librería es tan útil como simple, nos ofrece un objeto del tipo *HttpClient*, que nos ofrece múltiples métodos para realizar las llamadas a una URL concreta.

```
var response = await http.post(url, body: fooBody});
```

Como se observa en el fragmento de código anterior, tendremos los métodos:
.get(), .post(), .put() y *.delete()*, que enviarán la correspondiente petición a la URL que le pasemos por el primer parámetro y el cuerpo que le pasemos por el segundo, si es que es necesario para ese tipo de petición.

6.2.3 Interfaz

La interfaz de la aplicación móvil está realizada con Flutter, que como se ha explicado previamente es un kit de desarrollo de Google para Android e iOS.

Flutter para crear la interfaces de usuario se basa en *widgets*, que son los componentes que implementaremos para dibujar nuestra aplicación.

Existen dos tipos de con estado (*StatefulWidget*) y sin estado (*StatelessWidget*), que se diferencian principalmente en que, los *widgets* con estado pueden redibujarse cuando cambia alguna de sus propiedades.

En nuestro caso para la creación de la interfaz hemos utilizado el paquete de *material* que contiene infinidad de componentes configurables prehechos, como por ejemplo: *AppBar* (una barra superior para nuestra aplicación donde podemos incluir un menú lateral o el nombre de la aplicación), *Column* (nos permite añadir múltiples *widgets* bajo una misma columna), *Card* (un contenedor con bordes y sombra), etc.

También existe la posibilidad de utilizar *Cupertino* que simula el diseño de la compañía Apple, pero en nuestro caso nos hemos centrado en el diseño *material* que sigue la guía de estilo de *Material Design*.

6.2.3.1 Inicio de sesión

La pantalla de inicial en nuestro caso es la de inicio de sesión [Figura 6.2]. Esta pantalla tendrá como funcionalidad principal permitir al usuario identificarse y acceder al servicio, y como funcionalidad secundaria, enlazar la ventana de registro y el mapa de sonido.

Para la creación de esta pantalla, se ha utilizado un *widget* del tipo *Form* como base, que nos permite crear un formulario con validación (nos devuelve un error si no tiene la longitud necesaria, los caracteres permitidos, etc.). Dentro del formulario, hemos utilizado componentes *TextFormField* que simplemente son nuestros elementos de introducción de datos y que nos servirán para añadir los campos de usuario y contraseña necesarios. Estos campos serán validados tras pulsar el botón de acción, de tipo *RaisedButton* que nos devuelve un botón con elevación y sombra.

Este formulario se envía al servidor que comprueba que son correctos y nos devolverá una respuesto satisfactoria o errónea.
En el caso de que la respuesta sea satisfactoria se navega hacia la pantalla de descubrimiento de dispositivos.
En cualquier otro caso, se mostrará por pantalla un texto con el mensaje de error recibido.

Por otra parte, se añade un botón, del tipo *RaisedButton*, en la parte inferior de la pantalla que al pulsarlo nos redirijirá a la pantalla de registro.

Por último, se ha añadido una barra de navegación inferior, de tipo *BottomNavigationBar*, que nos dibujará un menú inferior con botones e iconos que contiene un enlace hacia la ventana de visualización del mapa.

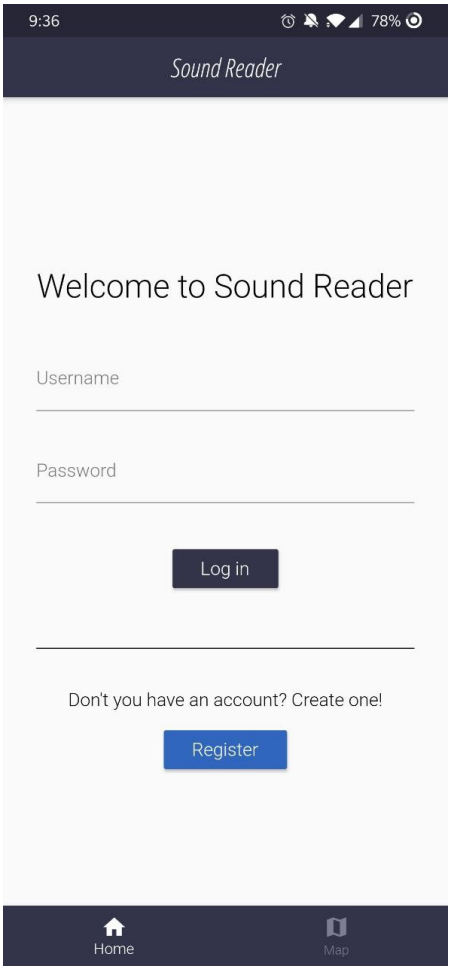

Figura 6.2 Implementación de pantalla de identificación

6.2.3.2 Registro

Esta pantalla [Figura 6.3] tiene una estructura muy similar a la de inicio de sesión. Su principal funcionalidad es registrar a un usuario en el sistema. Para ello, de nuevo, hemos utilizado un *widget* de tipo *Form* con tres campos de introducción de datos *(TextFormField)*, uno para la introducción del nombre de usuario y dos para la contraseña, donde se repetirá su valor para evitar fallos de escritura.

Nuestro formulario comprobará que el nombre de usuario tiene una longitud correcta y que los campos de contraseña coinciden, si no es así, se mostrará un mensaje de error bajo los elementos.

Si todo es correcto, al pulsar el botón de envío de formulario, *(RaisedButton)* se enviará al servidor, cuya respuesta puede ser un error, como por ejemplo que el nombre de usuario ya está en uso, o un mensaje de éxito.
Si se recibe un error, se muestra un texto por pantalla con el mensaje recibido. En caso de éxito, se redirije a la ventana de inicio de sesión.

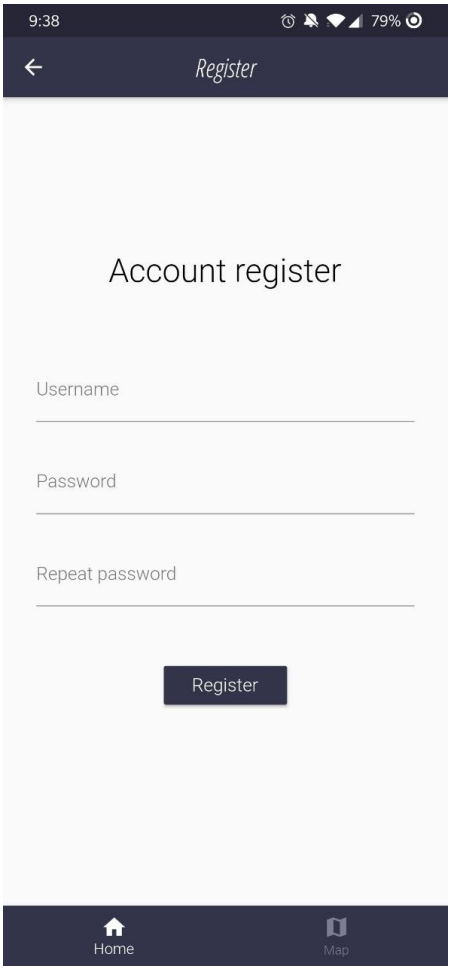

Figura 6.3 Implementación de pantalla de registro

6.2.3.3 Descubrimiento de dispositivos

La vista de descubrimiento de dispositivos [Figuras 6.4, 6.5] hace uso de la conexión Bluetooth del cliente móvil.
Para esta pantalla necesitaremos los permisos de localización, los pediremos al usuario al acceder a la ventana, si el usuario deniega el uso de la localización, se volverá a la ventana de inicio de sesión, si se acepta, se iniciará el servicio de descubrimiento.

El servicio de descubrimiento hace uso de la librería *Flutter Bluetooth Serial* [15], explicada previamente en el punto 6.2.1.

El descubrimiento de dispositivos haciendo uso de la mencionada librería se podría resumir en el siguiente código:

```
_flutterBluetoothSerial.startDiscovery().listen((event) {
      this.setState(() {
        _devices.add(event.device);
      });
}
```

Un objeto del tipo *FlutterBluetoothSerial* empieza el descubrimiento de dispositivos mediante el método *startDiscovery()* que es un objeto del tipo *Stream*, este tipo de objetos es muy usados para servicios asíncronos, es decir, que no se resuelven en un momento concreto, y su funcionalidad es básicamente ir devolviendo elementos conforme sean recibidos, de esta manera se evitan esperas innecesarias.
Este objeto posee un método llamado *listen()*, que toma por parámetro una función, cuyo argumento es el elemento recibido, y que se llama cada vez que se recibe un objeto nuevo.
En nuestro caso, simplemente, añadimos el elemento a una lista llamada *_devices*, donde almacenaremos nuestros dispositivos descubiertos, que en este caso es el elemento recibido del *Stream*.

La lista de dispositivos es mostrada al usuario, para dibujarla hemos utilizado un *widget* del tipo *ListView*, que nos renderizará una lista por pantalla con la lista de *widget* que se le pasa por el argumento *children*.

Cada dispositivo será mapeado a un *widget* del tipo *ListTile*, que mostrará una entrada para la lista. Esa entrada tendrá un hijo, del tipo *Card* con los datos del dispositivo como el título, que se corresponde con el nombre del dispositivo y el subtítulo, que es el identificador único. Además, se añade un botón con el icono de Bluetooth, que nos servirá para iniciar el enlazado al dispositivo seleccionado.

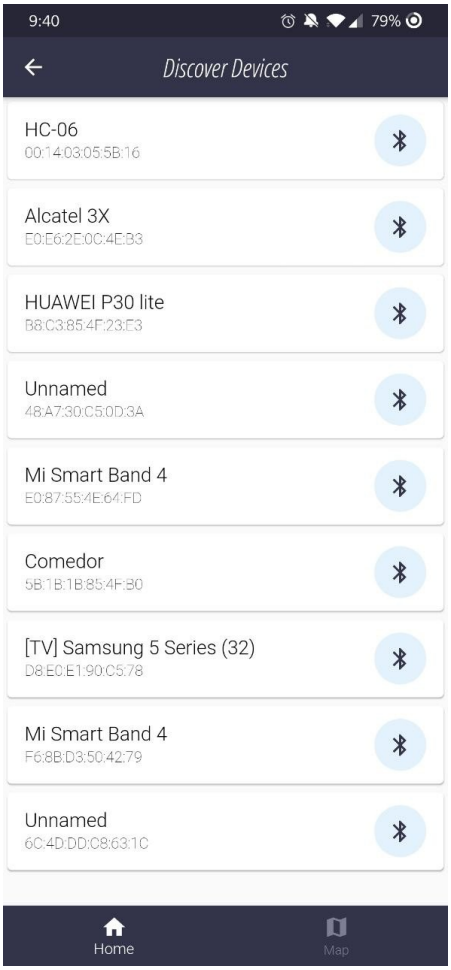

Figura 6.4 Implementación de descubrimiento de dispositivos

Cuando dicho botón es pulsado, se llama al objeto de tipo *BluetoothConnection* mediante su método estático *toAddress()*, donde se le pasará como argumento la dirección (o identificador) del dispositivo para conectarnos a él.
Además, se dibujará en ventana un *toast*, que servirá para decir al usuario que estamos intentando conectarnos al servicio.

```
BluetoothConnection connection = await
BluetoothConnection.toAddress(address);
```

Si la conexión es correcta, navegaremos hacia la ventana de lectura de datos. Si no lo es, se mostrará un mensaje en el *toast* con un mensaje de error en la conexión de emparejamiento.

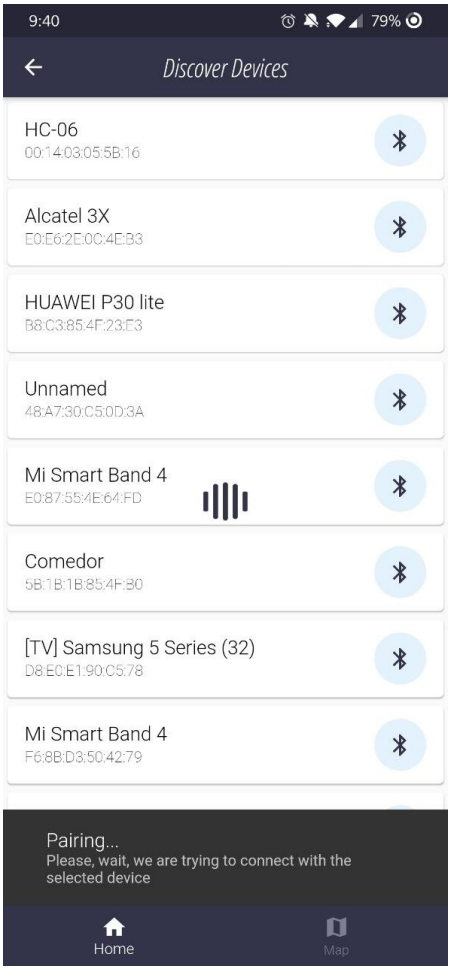

Figura 6.5 Implementación de emparejamiento de dispositivos

6.2.3.4 Lectura de datos

La interfaz de usuario de esta ventana [Figura 6.6] es bastante sencillo, se compondrá de un *Card* que muestra la cantidad de datos enviados y el número de ellos en cola, junto con un botón en el centro que permite finalizar el servicio de lectura y volver a la pantalla de descubrimiento de dispositivos.

```
widget._connection.input.listen((data) {
      readingData += ascii.decode(data);
      String separator = "\r\n";
      if (readingData.contains(separator)) {
        List<String> readings = readingData.split(separator);
        readingData = readings[1];
        Geolocator()
            .getCurrentPosition(desiredAccuracy:
LocationAccuracy.high)
            .then((Position position) {
          String language =
Strings.of(context).valueOf("language");
          String userId = globals.userId;
```

```
          sendRead(userId, position.latitude,
position.longitude, readings[0], language);
        });
      }
    });
```

De la misma manera que se explicó en el punto 6.2.1 se reciben y decodifican los datos.
Una vez tenemos los datos recibidos mediante Bluetooth desde la placa Arduino, se toma la localización del usuario haciendo uso de una librería externa *Geolocator* [16], mediante un objeto del tipo *Geolocator* y haciendo uso de su función *.getCurrentPosition()*.
Haciendo uso de los datos de localización conseguidos, los de lectura recibidos y los de lenguaje e identificador de usuario que teníamos almacenados de etapas anteriores, se realiza una petición al servidor web para añadir una nueva medición.

Dentro de esta llamada, hemos añadido un mecanismo de control de errores, por si el usuario pierde la conexión a internet y evitar perder también los datos leídos. Si una petición al servidor falla, los datos se almacenan en una cola, y cada vez que se reciba un nuevo dato, se enviarán tanto los nuevos como los previos, evitando así la pérdida y haciendo la aplicación mucho más robusta.

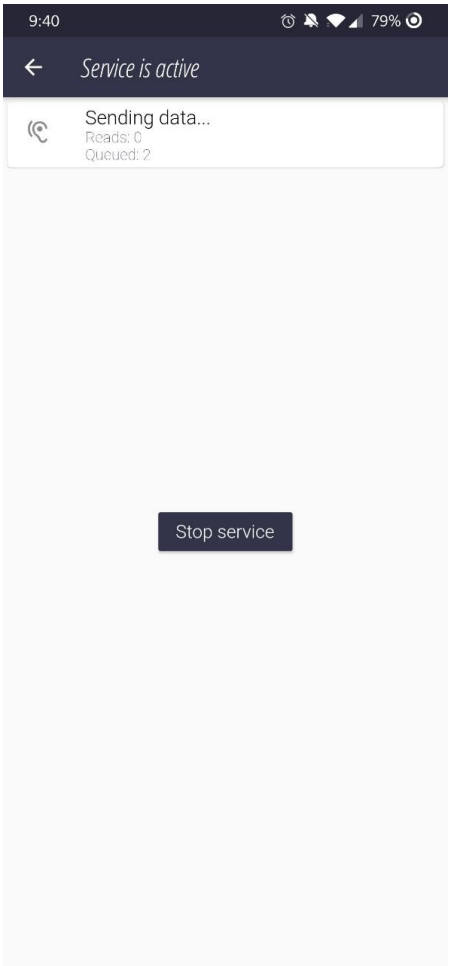

Figura 6.6 Implementación de lectura de datos

6.2.3.5 Visualización de mapa

Para esta pantalla [Figura 6.7], hemos utilizado el cliente web, es decir, lo hemos añadido directamente mediante un ventana a la web con una librería que nos permite mostrar contenido web en nuestra aplicación: *WebView Flutter* [17].

Esta librería nos ofrece un *widget* que tendrá una lógica especial como hemos explicado. Cabe destacar que usa *Chromium*, un proyecto de navegador de código abierto de Google, en su implementación.

```
Widget build (BuildContext context) {
   return WebView(
          initialUrl: 'https://jmartinruiz-tfg-
client.netlify.app/?mode=fullscreen',
          javascriptMode: JavascriptMode.unrestricted,
       );
}
```

El modo de usarlo es bastante sencillo, simplemente se utiliza el *widget* proporcionado, denominado *WebView*, se añade la url que

queremos mostrar y como necesitamos que ejecute JavaScript para mostrar la aplicación React, le añadimos la opcion de modo JavaScript sin restricciones.

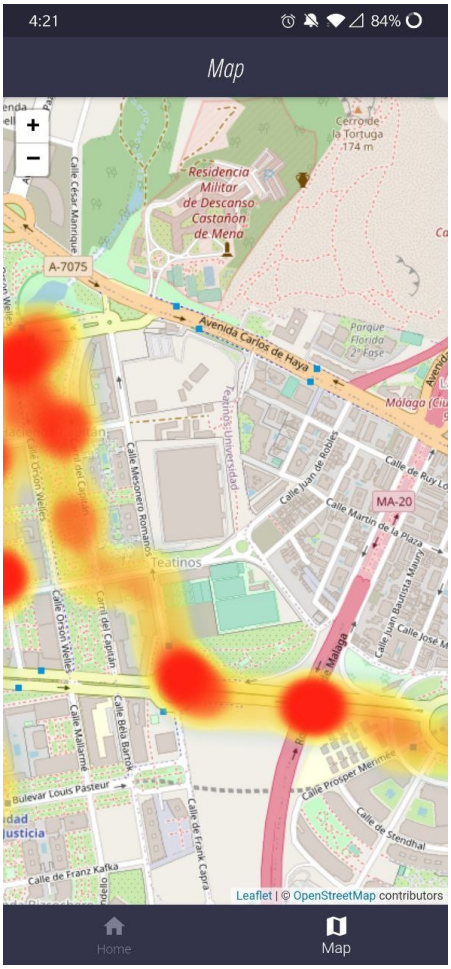

Figura 6.7 Implementación de visualización de mapa

6.2.3.6 Multilenguaje

Para añadir la capa de multilenguaje a nuestra aplicación móvil se ha hecho uso de una librería externa, *internationalization* [18], que nos permite añadir valores para los textos en los idiomas que queramos implementar. Para ello, se hace uso de archivos JSON, que contiene una clave y un valor para cada entrada de traducción que queramos mostrar.

En nuestro caso, simplemente se ha utilizado inglés y español.

Para hacer uso de dicha librería, se utiliza el método *Strings.of(context).valueOf()*, donde *Strings.of(context)* nos devuelve una estructura del tipo *Map* que se corresponde con el archivo JSON de traducciones. Y con el uso del método *.valueOf()*, se recibe el valor de la clave que se le pasa como argumento.

6.3 Implementación del servidor web

6.3.1 Rutas

Para la implementación de nuestro servidor web, la principal parte es dividir nuestro proyecto en rutas, es decir, en las temáticas cuyo datos van a ser tratados. En nuestro caso, y tal como hemos explicado anteriormente, necesitamos dos diferentes rutas, una para gestión de usuarios, dónde trataremos el registro y el inicio de sesión de cada persona y otra en la que gestionaremos las lecturas, es decir, la añadición de una nueva medición y las peticiones de obtener las lecturas hasta ahora, así como la descarga de los ficheros de datos.

6.3.1.1 Usuarios

En esta primera ruta, tendremos diferentes subrutas, es decir, acciones que se podrán realizar dentro de ella, las describiremos mediante una tabla para simplificar su descripción.

Ruta (Método)	**Cuerpo**
/user/signIn (POST)	- *Username* - *Password* - *Language* (Opcional)
/user/signUp (POST)	- *Username* - *Password* - *Language* (Opcional)

Tabla 6.1. Descripción de rutas de usuario

En la primera ruta, se trata el inicio de sesión de un usuario, si el usuario existe y la contraseña introducida coincide con la cifrada en nuestra base de datos, se devolverá un mensaje de éxito, sino, se devolverá un mensaje con el error, por ejemplo, usuario no encontrado o contraseña incorrecta, todo esto en el idioma que se nos haya introducido o en inglés, si no está soportado.

En la segunda ruta, se registra un nuevo usuario dados los datos de usuario y contraseña introducidos, pero, primero, se comprueba que la contraseña es válida, es decir, tiene una longitud mínima establecida. Si todo es correcto, se devuelve un mensaje de éxito.

6.3.1.2 Lecturas

Para la ruta de lecturas, ofreceremos distintas acciones que se podrán realizar sobre las mediciones:

Ruta (Método)	Cuerpo / Parámetros de consulta
/measurement (GET)	- *From* - *To*
/measurement/new (POST)	- *UserId* - *Value* - *Coordenates* - *Language* (Opcional)

Tabla 6.1. Descripción de rutas de lecturas

En la primera ruta, es una llamada *GET*, es decir, se devuelve información, en nuestro caso se devolverá un objeto de lista con todas las lecturas almacenadas en la base de datos.
Además, dependiendo de los *query params* (parámetros de consulta), que se añadan a la petición, los datos serán filtrados por las fecha de inicio (*from)* y final *(to)*, que se definan.

En la segunda ruta, se añade una nueva medición dados un valor, unas coordenadas y un id de usuario, que será el correspondiente al usuario que ha iniciado sesión, si existe algún tipo de problema, se devolverá un mensaje de error.

6.3.2 Multilenguaje

Tal y como hemos mencionado previamente, hemos añadido manualmente un mecanismo de obtención de traducción multilenguaje para nuestra aplicación.

Para ello, primero, en las peticiones podremos obtener una propiedad *language* donde recibiremos el valor del lenguaje que tenemos que devolver, si no recibimos nada utilizamos el valor por defecto, en nuestro caso el inglés.

Por otra parte, tendremos almacenados unos archivos que hemos llamado *locale*, son archivos de tipo JSON que almacenan los datos a devolver en un idioma específico. Es decir, tendremos, por ejemplo, un archivo *es.json* que guardará, entre otros, los mensajes de error en español. En el caso de nuestra aplicación tenemos dos archivos, uno para inglés y otro para español.

Y por último tenemos un método *LocaleManager*, que recibe como argumento un parámetro de tipo cadena de texto, que define nuestro lenguaje seleccionado, si el lenguaje no existe tomaremos el valor por defecto, inglés. Este método devolverá un objeto JSON, tomado del archivo correcto, que hemos explicado previamente, con los datos del idioma requerido.

6.4 Implementación de la base de datos

Para la implementación de la base de datos no ha sido necesario ninguna formalidad extra, simplemente hemos creado una nueva cuenta en el servicio MongoDB Atlas y un nuevo *sandbox* (así es como llaman a las entidades que hostean, básicamente es nuestra base de datos). Luego simplemente el servicio nos ofrece la URL de conexión con los paramétros a rellenar con nuestro usuario y contraseña.

6.5 Implementación del cliente web

6.5.1 Interfaz

En el caso de la aplicación web, hemos utilizado JavaScript y la librería *React* para crear nuestro cliente.

Principalmente, React para la creación de la interfaz se basa en sus llamados componentes, que son las partes visuales junto con su lógica que nosotros crearemos para cada una de las partes de la aplicación.

Cabe destacar que para facilitar el desarrollo de los componentes hemos utilizado una librería de componentes ya diseños y totalmente configurable, se trata de *Material-UI*, creada por Google siguiendo su guía de estilo de *Material Design*, la misma que seguimos en nuestra aplicación móvil, para mantener la consistencia.

Al utilizar *Material-UI*, se nos permite añadir un tema a la aplicación para mantener los colores en todos los componentes a lo largo de nuestro cliente, que hemos inicializado con una temática simple de colores negro y blanco.

6.5.2 Mapa

Para la creación de un mapa en nuestro cliente web [Figura 6.8], hemos utilizado varias librerías externas: *Leaflet*, *React Leaflet* y *React Leaflet Heatmap Layer*.

- *Leaflet*: Como se ha describido en el apartado de tecnologías, es una librería de mapas, que nos permitirá utilizar Open Street Maps en nuestra aplicación de manera gratuita para mostrar nuestros datos.
- *React Leaflet*: Esta librería nos ofrece componentes React que utilizan la lógica de Leaflet y que nos permitirán una fácil adaptación de dicha librería a nuestros componentes visuales.

- *React Leaflet Heatmap Layer*: Es una extensión para la librería anterior, nos añade un nuevo componente para nuestro mapa que nos permitirá mostrar una capa de calor con los datos que nosotros ofrezcamos a dicho componente.

Con esto, simplemente ha sido necesaria la adaptación de los datos con los distintos componentes ofrecidos por la librerías utilizadas, además de estilizarlos para que se dibujen correctamente.

Además, se ha añadido un refresco de los datos del mapa cada 5 segundos. Para ello, se ha hecho uso de los métodos *useEffect* y *setTimeout*, de React y JavaScript respectivamente.

El primer método se encarga de disparar una función que se pasa por parámetro cada vez que los argumentos que se le pasan como segundo paramétro cambian dentro del componente, de esta manera, cada vez que cambiamos las mediciones, es decir, se han recibido nuevos datos, enviamos una nueva petición.

El segundo método, simplemente retrasa una función que se le pasa por primer argumento, la cantidad de milisegundos que se le pasan por segundo parámetro, de esta manera, podemos retrasar la petición de recibimiento de datos la cantidad de tiempo que queramos, en nuestro caso 5000 milisegundos ó 5 segundos.

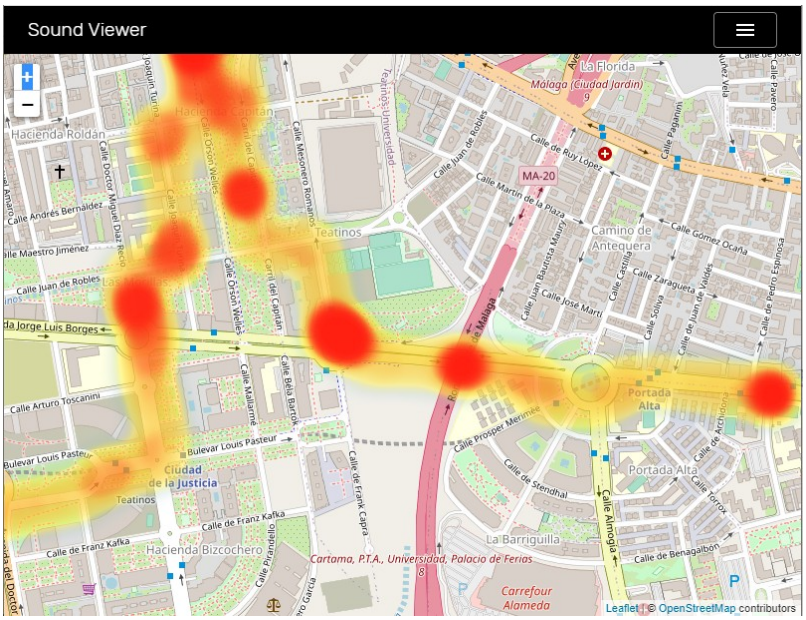

Figura 6.8 Implementación de visualización de mapa en cliente web

6.5.3 Filtrado

Una de las funcionalidades que se definieron como primarias en nuestra aplicación fue la de filtrar datos por fecha, que permitirá poder ver la evolución del sonido a lo largo del tiempo.

Para ello, nuestra aplicación consta de un menú en la parte superior de nuestro cliente, que posee un botón que nos permitirá abrir unas opciones [Figura 6.9]. Dentro de estas opciones, tenemos dos campos de introducción de datos de tipo fecha, dónde el usuario podrá seleccionar el inicio y/o fin del filtrado deseado.
Para que los cambios surjan efecto, también se ha añadido un botón con el texto “Filtrar” para enviar la petición de filtrado.

Una vez pulsado el botón, el mapa volverá a cargarse, pero esta vez con los datos de las fechas seleccionadas.

Para la implementación de estos campos, hemos utilizado el *framework* de componentes Material-UI, donde hemos utilizado:

- *Appbar*: Es el componente que nos permite incorporar la barra superior dónde añadiremos el botón de opciones y el título de la web.
- *Button*: Lo hemos utilizado en varios sitios, por una parte en el botón con el icono de Menú que se ha añadido en la barra *Appbar* y en el botón de filtrado que se ha añadido en el submenú de opciones.
- *Popper*: Este elemento nos permitirá dibujar un diálogo emergente, que funcionará como nuestro submenú que contiene las opciones de filtrado.
- *Textfield*: Básicamente es nuestro elemento de introducción de datos, en nuestro caso hemos utilizado el tipo fecha *(type=”date”)*, para poder ofrecer un campo que sólo permita la introducción de datos del mismo tipo y además mostrar un calendario para facilitar la selección.

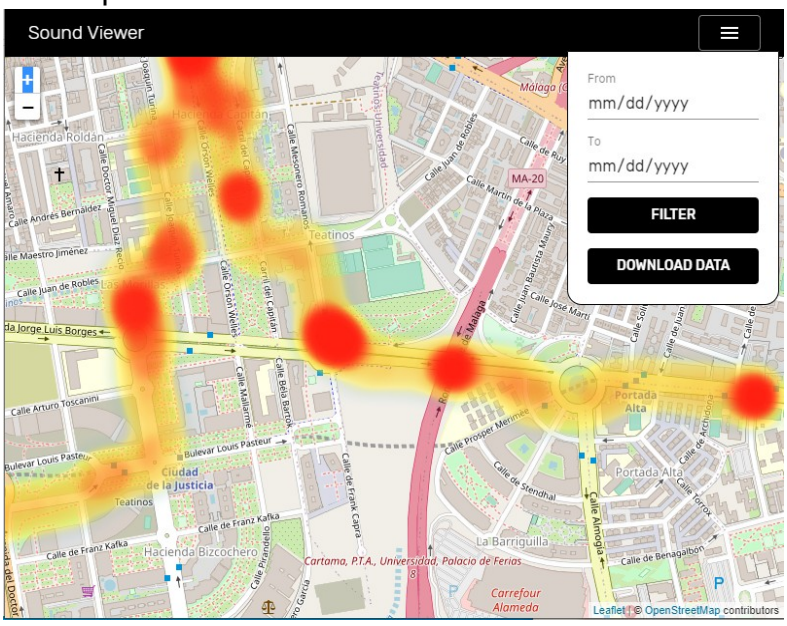

Figura 6.9 Implementación de filtrado en cliente

6.5.4 Descarga archivos de datos

Para la implementación de la funcionalidad de descarga de archivos de datos hemos utilizado el submenú que se utilizó para el filtrado de datos. En este caso, para la interfaz de usuario se ha añadido un botón (del tipo *Button* del *framework* Material-UI), con el texto "Descarga de archivo" que al hacer *click* nos comenzará automáticamente la descarga de un archivo del tipo JSON con los datos de mediciones (filtrados por fechas, si es que tenemos algún filtro).

La lógica interna, se basa en la realización de una petición al servidor web que nos devuelve un objeto, del tipo JSON, con los datos de las mediciones y con JavaScript, haciendo uso de sus métodos nativos, hemos añadido dicha información a un archivo que es descargado.

```
const fileData = JSON.stringify(response.data);
 const blob = new Blob([fileData], {type: "application/json"});
const url = URL.createObjectURL(blob);
const link = document.createElement("a");
link.download = `measurements${from ? "_" + from : ""}${to ? "_"
+ to : ""}.json`;
link.href = url;
link.click();
```

El funcionamiento de esta descarga se puede ver en el código anterior.

1. Usamos el método *JSON.stringify()* para convertir los datos recibidos en un objeto de tipo JSON, que es el que queremos devolver.
2. Creamos un objeto de tipo *Blob* con los datos que acabamos de convertir.
3. Utilizando *URL.createObjectUrl()* creamos un enlace hacia el objeto *Blob* que acabamos de crear.
4. En los siguientes pasos creamos un elemento HTML del tipo *<a>*, que crearemos fuera del DOM (Document Object Model [19]) de nuestra aplicación y que nos servirá para añadir los datos de descarga de nuestro archivo, con el enlace previamente creado.

7

Pruebas

7.1 Pruebas de Arduino

Una de las mayores ventajas de Arduino es que te permite obtener mucha funcionalidad con poco código, esto nos da muchísimas ventajas a la hora de depurar la aplicación y comprobar que funciona correctamente.
En nuestro caso, al ser un código muy sencillo, la prueba que hemos realizado a nuestro Arduino para comprobar el correcto funcionamiento ha sido:

- Leer los datos en dos ambientes distintos:
  - Ambiente tranquilo: Comprobamos que la lectura del sensor es un sonido bajo y normal.
  - Ambiente ruidoso: Confirmar que los datos leídos son altos y se diferencian de los leídos en un ambiente tranquilo.
- Establecer una conexión Bluetooth con un dispositivo y comprobar que los datos recibidos son iguales a los mostrados por la consola del Arduino IDE.

7.2 Pruebas de aplicación móvil

Para comprobar que la funcionalidad de la aplicación móvil es correcta se han hecho uso de los diagramas de secuencia realizados,

se han seguido las acciones del diagrama y se ha comprobado que, tras seguirlas, el resultado es el esperado.

Para abstraernos de nuestra aplicación esta prueba se ha realizado por dos personas distintas: nosotros, como desarrollador de la aplicación y una persona externa al sistema, ambos resultados fueron positivos.

7.3 Pruebas de servidor web

La comprobación del servidor web se ha realizado mediante el programa Postman [20] [Figura 7.1], que nos permite realizar peticiones a una URL con unos datos dados.
Para ello, esta herramienta nos permite crear una biblioteca de peticiones; funcionalidad que hemos utilizado para nuestra prueba.

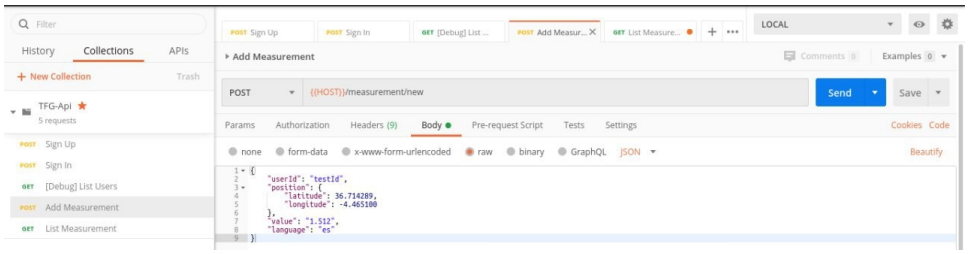

Figura 7.1 Programa Postman con la biblioteca de peticiones del proyecto

Se ha creado una biblioteca con todas las posibles rutas de nuestra aplicación, así como un cuerpo de petición fijo para cada una de ellas. Lo que nos permitirá comprobar tras cada actualización que las peticiones realizadas son siempre iguales y los valores esperados se corresponden con los correctos.

7.4 Pruebas de cliente web

Las pruebas sobre el cliente web se han realizado de distintas maneras dependiendo de la funcionalidad a comprobar.

7.4.1 Mapa con datos correctos

Se ha hecho uso de las herramientas de desarrollador de Chrome [21], concretamente de las herramientas de red que nos permiten visualizar las peticiones *http*, en este caso se ha comprobado que los datos recibidos son los mismos que los recibidos en el servicio de *Postman*, que hemos mencionado previamente, de esta manera comprobamos que los datos sean correctos.

7.4.2 Refresco de mapa

De nuevo haciendo uso de las herramientas de desarrollador se comprueba visualmente que se realiza una llamada al servidor cada 5 segundos de manera ininterrumpida.

7.4.3 Filtro de datos

Esta funcionalidad se ha comprobado tanto de manera visual como interna.

Por una parte, con las herramientas de desarrollador, comprobando que las peticiones actuales y futuras se realizan haciendo uso de los filtros establecidos.

Por otra parte, comprobando visualmente que los datos cambian, que se ha refrescado el mapa con datos distintos.

7.4.4 Descarga de archivo

La comprobación de esta funcionalidad la trataremos descargando un archivo con los datos de mediciones, comprobaremos que el archivo se descarga correctamente y tiene el formato adecuado.

Y por otra parte, utilizando los mismos filtros usados para la descarga del archivo, comprobaremos que el contenido del mismo es igual al realizar la petición mediante el uso de *Postman*.

8

Conclusiones y líneas futuras

8.1 Conclusiones

El objetivo principal de este trabajo de fin de grado era probar lo aprendido durante la duración del grado que he cursado, y se ha cumplido por completo, para el desarrollo de este proyecto se han tocado temáticas y herramientas de todas las índoles.

Hemos conseguido pasar desde el primer paso de nuestra metodología a seguir, la creación de casos de uso y de secuencia hasta la implementación completa y final de nuestro proyecto, pasando por el prototipado de interfaces.

Además, con la ayuda de las documentaciones de todas las distintas tecnologías utilizadas hemos conseguido crear un sistema completo multilenguaje, donde hemos utilizado lenguajes de bajo nivel como Arduino, lenguajes nuevos como Dart y multiusos como JavaScript.
Todo esto teniendo en cuenta la multitud de herramientas extra utilizadas y de muy distinto uso como el kit de desarrollo Flutter, la librería de creación de interfaces de usuario React o el framework de creación de servidores Express.js.

Se ha cumplido el principal objetivo, hemos sido capaces crear un sistema con múltiples entidades todas interconectadas entre si, haciendo uso de distintos tipos de comunicación.

Por lo que podemos confirmar que se ha conseguido cumplir el proyecto descrito en el anteproyecto y aplicado con éxito los conocimientos adquiridos a lo largo del grado.

8.2 Líneas futuras

Algunos de los puntos que no han tenido cabida en nuestro proyecto, pero que sería muy interesante añadirlos en un futuro serían:

- Añadir análisis de datos: mostrar en el cliente una pestaña con gráficas que permitan visualizar los datos de sonido de una manera distinta.

- Añadir filtrado por zonas: que permita al usuario obtener los datos de una zona específica y poder tratar esos datos por separado para el posterior estudio.

- Crear un foro: donde los usuarios activos que aporten las mediciones a la plataforma puedan compartir sus ideas y opiniones.

Referencias

[1] IBM, https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/
[2] Bernd Resch, People as Sensors and Collective Sensing – Contextual Observations Complementing Geo-Sensor Network Measurements
[3] Arduino, https://www.arduino.cc/reference/en
[4] Node.js, https://nodejs.org/en/docs/
[5] Express.js, https://expressjs.com/en/4x/api.html
[6] Open Street Maps, https://wiki.openstreetmap.org/
[7] Leaflet, https://leafletjs.com/reference-1.6.0.html
[8] React, https://reactjs.org/docs/getting-started.html
[9] TypeScript, https://www.typescriptlang.org/docs/home.html
[10] Material-UI, https://material-ui.com/getting-started/installation/
[11] Flutter, https://flutter.dev/docs
[12] Dart, https://dart.dev/guides
[13] Git, https://git-scm.com/docs
[14] React Leaflet Heatmap Layer, https://www.npmjs.com/package/react-leaflet-heatmap-layer
[15] Dart *http* library, https://pub.dev/packages/http
[16] *Geolocator* library, https://pub.dev/packages/geolocator
[17] *WebView Flutter* library, https://pub.dev/packages/webview_flutter
[18] *Internationalization* library, https://pub.dev/packages/internationalization
[19] DOM, https://es.wikipedia.org/wiki/Document_Object_Model
[20] Postman, https://www.postman.com/
[21] Chrome DevTools, https://developers.google.com/web/tools/chrome-devtools

Apéndice A

Manual de Instalación

Requisitos

- Tener Node.JS instalado
- Tener Git instalado
- Tener Arduino IDE
- Tener un dispositivo Android

Instalación de Arduino

1. Abrir la consola de *git*.
2. Usar el comando:
   `git clone https://github.com/JaysusM/TFG-Arduino-Sensor.git`
3. Abrir Arduino IDE.
4. Abrir la carpeta donde hayamos clonado el repositorio en Arduino IDE.
5. Pulsar el botón de compilar y subir (flechita derecha).

Instalación de Aplicación móvil

1. Abrir el siguiente enlace en el dispositivo Android: https://github.com/JaysusM/TFG-Mobile-App/releases/download/1.0.0/SoundReader.apk
2. Permitir la instalación de archivos de origen desconocido en Android.
3. Instalar el archivo *.apk* descargado previamente.

Instalación de Cliente web

1. Abrir la consola de *git*.
2. Usar el comando:
   `git clone https://github.com/JaysusM/TFG-Client.git`
3. Abrir la carpeta donde hayamos clonado el repositorio en la consola.
4. Usar el comando:
   `npm install`
5. Usar el comando:
   `npm start`

El cliente corre, por defecto, en http://localhost:3000

Instalación de Cliente web

6. Abrir la consola de *git*.
7. Usar el comando:
   `git clone https://github.com/JaysusM/TFG-Api.git`
8. Abrir la carpeta donde hayamos clonado el repositorio en la consola.
9. Usar el comando:
   `npm install`
10. Usar el comando:
    `npm start`

El servidor corre, por defecto, en http://localhost:5000

Apéndice B
Manual de Usuario

Manual de Arduino

La placa Arduino tiene como función primaria la lectura de la cantidad de sonido y enviarlos al dispositivo móvil.
Para realizar dicha acción, el dispositivo tiene que estar correctamente configurado y montado.

Para ello, tendremos que montar correctamente los módulos en la placa Arduino, en nuestro caso el de sonido y el de Bluetooth.

Empezaremos por el módulo de sonido, donde tendremos que conectar tres cables con la placa.

- GPIO 3V del módulo con GPIO 3.3V de la placa, este se encarga de darle alimentación al módulo
- GPIO GND del módulo con GPIO GND de la placa, donde conectaremos los pines de tierra.
- GPIO AC del módulo con GPIO A0 de la placa, por donde pasará el valor leído por el sensor.

Por otra parte tendremos que conectar el módulo de bluetooth HC-06:

- GPIO VCC del módulo con GPIO 5V de la placa, para conectar la alimentación del módulo.
- GPIO GND del módulo con GPIO GND de la placa, para conectar el módulo a un valor tierra.

- GPIO TXD del módulo con GPIO RX3 (14) de la placa, donde conectaremos el recibimiento del módulo con el envío de la placa.
- GPIO RXD del módulo con GPIO TX3 (15) de la placa, se conecta el pin de envío de la placa con el de recibimiento del módulo.

Una vez conectado, todo debería quedar como se muestra en la figura B.1, tras esto debemos seguir las instrucciones de la guía de instalación para poner en marcha nuestro dispositivo [Apéndice A - Instalación de Arduino].

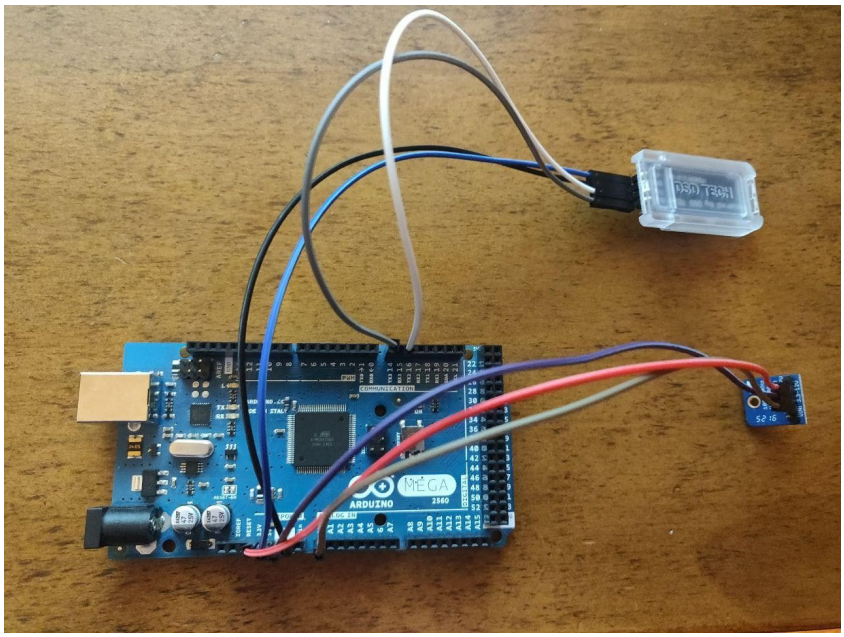

Figura B.1 Montaje de la placa Arduino.

Manual de Aplicación móvil

Para poner en marcha la aplicación móvil lo primero que debemos realizar es seguir los pasos de la guía de instalación [Apéndice A - Instalación de Aplicación móvil] y haber configurado correctamente nuestra placa Arduino (habiendo seguido su manual de uso [Apéndice B – Manual de Arduino]).

Una vez tengamos todo listo e instalado tenemos dos opciones dentro de la aplicación móvil: Visualización de mapa y añadir nuevas mediciones.

Visualización de mapa

Para seguir la primera opción de visualización de mapa, simplemente deberemos abrir nuestra aplicación y pulsar el botón que aparece en la barra de navegación inferior con el texto “Mapa” [Figura B.2], esto abrirá una nueva ventana con el mapa junto a los datos de sonido.

Figura B.2 Botón de visualización de mapa

Nueva medición

Si queremos añadir una nueva medición deberemos seguir los siguientes pasos:

Creación de cuenta

Este paso es opcional si ya tenemos una cuenta creada, y podemos continuar con el siguiente paso: Inicio de sesión.

Para la creación de una cuenta debemos abrir la aplicación y pulsar en el botón de “Registrarse”, este nos llevará a una nueva ventana, la pantalla de registro.

Ahora debemos rellenar los campos de texto con un nombre de usuario y una contraseña, que debemos repetir para evitar fallos inoportunos.
Tras esto, debemos hacer click en el botón de “Registrar” [Figura B.3], si este nos devuelve un error, debemos corregir los datos que hayamos introducido incorrectamente; si por el contrario todo es correcto, se nos redirigirá hacia la pantalla de inicio de sesión.

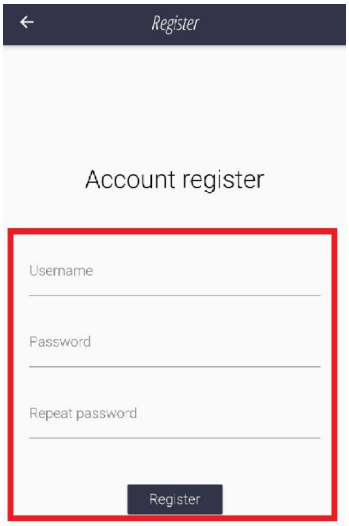

Figura B.3 Formulario de registro

Inicio de sesión

Para esta fase de identificación debemos estar en la pantalla de inicio de sesión, que es la que veremos una vez abramos la aplicación.

Para identificarnos debemos rellenar los campos de usuario y contraseña con nuestros datos de usuario, el que hemos registrado previamente, y tras ello pulsar el botón de iniciar sesión que aparece debajo de ellos [Figura B.4].

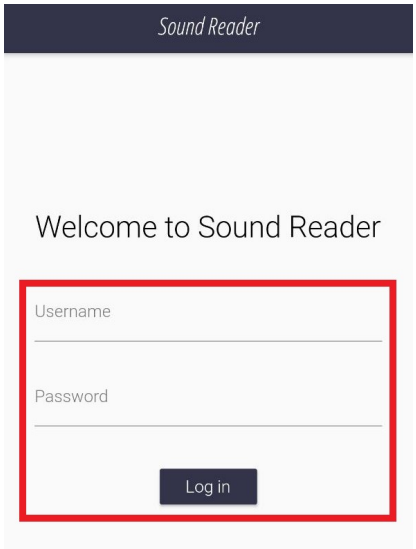

Figura B.4 Formulario de inicio de sesión

Si existe algún error se mostrará por pantalla y deberemos seguir las instrucciones ofrecidas para poder proseguir. Si todo es correcto, el sistema nos redirijirá a la pantalla de descubrimiento de dispostivos.

Descubrimiento de dispositivos

Para esta etapa de descubrimiento y enlazado con el dispositivo Bluetooth, primero debemos tener configurada la placa Arduino correctamente, tras haber seguido los pasos del Apéndice A – Instalación de Arduino y Apéndice B – Manual de Arduino.

En la pantalla veremos los dispositivos descubiertos vía Bluetooth, de entre ellos debemos seleccionar nuestro dispositivo Arduino, normalmente debería aparecer bajo el nombre de “HC-06”, que se corresponde con el nombre del módulo Bluetooth. Una vez identificado nuestra placa Arduino debemos enlazarnos a ella, para ello pulsaremos en el botón que aparece a la derecha del nombre e identificador, un icono con el símbolo de Bluetooth [Figura B.5].

Figura B.5 Botón de enlazado Bluetooth

Si algún error ocurre se mostrará un mensaje por pantalla al que debemos hacer caso para solucionar los problemas.

En el caso en que todo sea correcto, se nos redirijirá a la ventana de lectura de datos.

Lectura de datos

Esta etapa es la última en el proceso de lectura de datos, una vez estemos en la ventana correspondiente a la lectura de datos todo estará en funcionamiento, no es necesario ninguna acción extra para que el sistema envíe datos.

Si queremos finalizar el servicio de lectura de datos simplemente debemos pulsar el botón “Finalizar el servicio” [Figura B.6] que aparece en el centro de la pantalla y se nos redijirá a la ventana de descubrimiento de dispositivos.

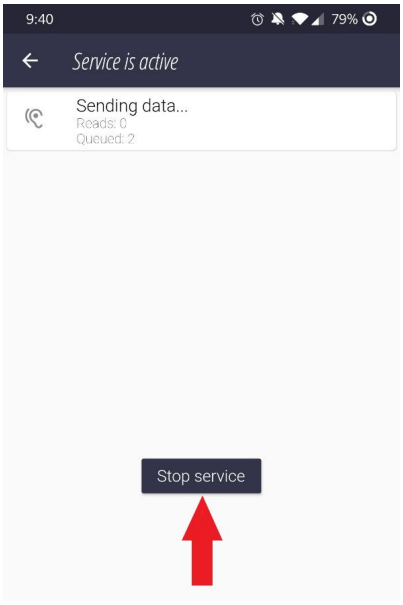

Figura B.6 Botón de finalización de servicio

Manual de Aplicación web

La aplicación web tiene como funcionalidad principal mostrar el mapa con las visualizaciones en directo, este se nos mostrará al abrir el cliente web sin necesidad de realizar ninguna acción.

Además, tenemos varias acciones secundarias como el filtrado de datos y la descarga de archivos de datos.

Para el filtrado de datos debemos abrir el menú de opciones, para ello deberemos pulsar el botón que aparece en la barra de navegación superior [Figura B.7].

Figura B.7 Menú de opciones

Una vez abierto, nos aparecerán dos campos de filtrado para seleccionar un período de tiempo en el que queremos filtrar, debemos rellenarlos con las fechas correspondientes y para finalizar el filtrado,

pulsaremos el botón de “Filtrar” que aparece debajo de ellos [Figura B.8].

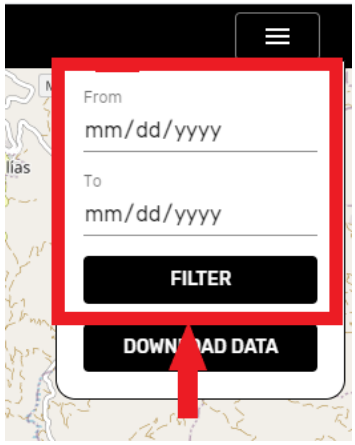

Figura B.8 Opciones de filtrado

La otra acción secundaria posible es la descarga de archivos, para ello podemos utilizar el filtro de fechas opcionalmente para obtener un archivo de datos en un período dado o simplemente descargar todos los datos posibles, simplemente pulsando el botón de “Descarga de datos” [Figura B.9] que aparece en el menú emergente que abrimos anteriormente.

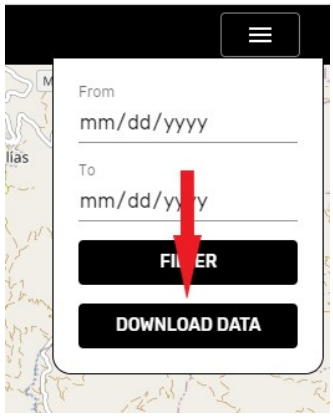

Figura B.9 Botón de descarga de datos

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA