Este Trabajo de Fin de Grado aborda el diseño y desarrollo de un sistema de monitoreo acústico para locales, estructurado mediante grupos de contenedores que optimizan su funcionamiento. El proyecto incluye un simulador de ruido implementado en Arduino, encargado de enviar datos de prueba al servidor, así como mejoras en la plataforma web para facilitar la supervisión y el control de los niveles acústicos. Además, se ha creado documentación completa para apoyar a futuros desarrolladores en el mantenimiento y evolución del sistema.

El proyecto se aborda desde una doble perspectiva: por un lado, la gestión de un sistema de monitoreo acústico en locales, optimizado mediante contenedores que aseguran su eficacia; y, por otro, la creación de un entorno integrado para que los instaladores y autoridades supervisen los niveles de ruido de manera precisa y en tiempo real.

Este enfoque integral requiere la aplicación de metodologías profesionales de ingeniería de software, las cuales minimizan los riesgos y aseguran la implementación y mantenimiento exitosos del sistema.

Pablo Francisco Puga Martínez es estudiante del Grado en Ingeniería Informática. Como autor de este proyecto, inicia una ambiciosa y nueva línea de investigación en el Proyecto GranaSAT, desarrollando su Trabajo de Fin de Grado y aplicando conocimientos avanzados en el ámbito de la informática y las telecomunicaciones.

Andrés María Roldán Aranda es el responsable académico del presente proyecto y tutor del estudiante. Es profesor en el Departamento de Electrónica y Tecnología de Computadores de

TRABAJO FINAL DE GRADO

Plataforma IoT y migración de servicios

Pablo Fransico Puga Martínez

INGENIERÍA INFORMÁTICA

23/24

Universidad de Granada

Grado en Ingeniería Informática

heimdal
SOUND CONTROL

Trabajo de fin de grado

Plataforma IoT y migración de servicios

Pablo Fransico Puga Martínez

2023/2024

Tutor: Andrés María Roldán Aranda

Plataforma IoT y migración de servicios

GranaSat

Autor
Pablo Francisco Puga Martínez

Directores
Andrés Roldan Miranda

Plataforma IoT y migración de servicios

Pablo Francisco Puga Martínez

Palabras clave: monitoreo acústico, gestión de servidores, contenedores Docker, simulación de datos, control de ruido, Arduino, plataforma web, documentación técnica, desarrollo de software, supervisión acústica

Resumen

El proyecto consiste en la gestión de un sistema de monitoreo acústico para locales mediante grupos de contenedores que optimizan su funcionamiento. También se ha desarrollado un simulador de ruido en Arduino que envía datos de prueba al servidor, y se han realizado mejoras en la plataforma web para facilitar la supervisión y el control de los niveles acústicos. Además, se ha generado documentación completa para ayudar a futuros desarrolladores en el mantenimiento y evolución del sistema.

IoT Platform and Service Migration

Pablo Francisco Puga Martínez

Keywords: acoustic monitoring, containers, server management, noise simulator, Arduino, web platform, documentation, future developers, system maintenance, acoustic levels.

Abstract

This project involves the management of an acoustic monitoring system for venues through groups of containers that optimize its operation. A noise simulator using Arduino has also been developed to send test data to the server, and improvements have been made to the web platform to facilitate the supervision and control of acoustic levels. Additionally, comprehensive documentation has been generated to assist future developers in maintaining and evolving the system.

“ Failure is simply the opportunity to begin again, this time more intelligently. “

Henry ford

Acknowledgments

I would like to express my deepest gratitude to all the people and organizations who contributed to the development of this Final Degree Project.

First and foremost, I would like to thank Andrés for his guidance, patience, and unwavering support throughout this process. His expertise and insights have been invaluable to the growth of this project and to my personal and academic development.

I am also grateful to the GranaSat lab team for providing access to technical resources and for their support in several stages of this work.

To my family, thank you for your unconditional support and for encouraging me to keep going during moments of greater effort. Your trust, motivation, and dedication have been essential to reaching this goal.

To my friends, thank you for always being there to support me with whatever I needed.

To my classmates, Jose María and Pedro, who have been with me throughout my studies, even in the toughest exams, and have given me their full support in this project.

Finally, I would like to express my appreciation for all the open-source platforms and tools that have been essential to the completion of this project. The availability of these resources made it possible to implement and optimize the functionalities of this work significantly.

To all of you, my sincerest thanks.

Índice

1. Introducción

1.1 Resumen de definición del proyecto

El presente proyecto se enmarca dentro de mi Trabajo de Fin de Grado, el cual culmina mi formación en el Grado en Ingeniería Informática con especialidad en Ingeniería del Software, cursado en la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada. Se ha llevado a cabo este trabajo con el propósito de demostrar las competencias y habilidades adquiridas a lo largo de mis estudios universitarios.

Para ello, he desarrollado este proyecto, enfocado en la mejora, implementación y migración de un sistema de monitorización de niveles de ruido en locales con equipos de sonido. Este trabajo ha implicado una revisión completa del sistema existente y la incorporación de funcionalidades nuevas, orientadas a optimizar su rendimiento y a facilitar su gestión en entornos reales. Además, este Trabajo de Fin de Grado se realiza en colaboración con el proyecto académico GranaSAT, una iniciativa multidisciplinaria dirigida por el profesor Andrés María Roldán Aranda, que reúne a profesionales y estudiantes de diversas áreas con el fin de fomentar el aprendizaje práctico en Ingeniería Aeroespacial e Ingeniería Electrónica.

Este proyecto se centra en el desarrollo de un sistema de monitoreo acústico para locales, que permite el registro y control de los niveles de ruido, asegurando su conformidad con los límites establecidos. La solución propuesta emplea una aplicación web que recibe y gestiona datos acústicos enviados a través de internet, mostrando de manera accesible y eficiente la información relevante para los usuarios.

Para asegurar la escalabilidad y la facilidad de administración, el sistema se ha implementado en un entorno de contenedores mediante Docker, lo que permite una estructura modular en distintos servicios. Adicionalmente, se ha diseñado un simulador en Arduino que emula el comportamiento de un limitador de ruido, enviando datos de prueba al servidor. Este sistema simulado facilita las pruebas y el ajuste de la aplicación en condiciones reales, garantizando su efectividad en el registro y el análisis de datos acústicos.

Finalmente, se han realizado mejoras en la interfaz y la lógica de la aplicación web para optimizar su uso, proporcionando una experiencia de usuario fluida y centrada en las necesidades de gestión y control de niveles acústicos. Este proyecto constituye así una plataforma completa para la supervisión acústica en entornos de locales comerciales.

1.2 Estructura del documento

Este documento está compuesto por las diferentes secciones puestas respecto al índice anterior:

- **Resumen**: breve resumen del proyecto que simplifica la funcionalidad del proyecto.
- **Introducción:** sección que explica la motivación para iniciar el proyecto, como la necesidad de contar con un entorno de pruebas eficiente y seguro para el sistema, la simulación de datos de ruido y mejoras en la interfaz y backend.
- **Estado del arte:** capítulo que contextualiza el proyecto dentro del ámbito de sistemas IoT, Docker y tecnologías web, destacando el uso de contenedores en infraestructura moderna y la transmisión de datos desde dispositivos físicos a servidores.
- Estado inicial: sección que se enfoca en documentar y explicar el estado del proyecto inicialmente.
- **Objetivos:** en esta sección se detallan los principales objetivos, como la contenerización del servidor mediante Docker, el desarrollo del simulador con Arduino, y las mejoras en la plataforma web en Angular y Node.js.
- **Planificación:** aquí se detalla la planificación del proyecto, incluyendo las herramientas y metodologías utilizadas. Se muestra también el backlog del producto y la organización de los módulos.
- **Diseño y tecnologías:** este capítulo explica las tecnologías empleadas, como Docker para contenerización, Arduino para simulación de datos, y Angular y Node.js para las mejoras en la web, además de los componentes del sistema y cómo interactúan.
- **Despliegue y funcionamiento:** sección dedicada al despliegue del sistema, incluyendo los pasos para montar los contenedores Docker, conectar el simulador de Arduino al servidor y requisitos para la web.
- **Conclusiones y vías futuras:** se resumen los resultados, calidad y logros del proyecto, y se mencionan las posibles mejoras a corto y largo plazo, como optimizar la simulación de datos o expandir funcionalidades en el servidor y la web.

1.3 Marco del proyecto

El presente proyecto se plantea como una solución integral para mejorar un sistema existente de monitoreo acústico para locales, centrándose en la optimización del servidor mediante grupos de contenedores que facilitan su rendimiento y escalabilidad. Este enfoque busca sentar las bases para futuras mejoras y garantizar el correcto funcionamiento del sistema en el largo plazo.

Una de las principales innovaciones es el desarrollo de un simulador de limitador de ruido utilizando Arduino. Este dispositivo permite la simulación de un entorno real al enviar datos falsos al servidor, contribuyendo así a la evaluación y validación del sistema sin necesidad de depender de datos reales en todas las circunstancias.

Además, se han llevado a cabo significativas mejoras en la plataforma web existente. Estas mejoras se han diseñado para aumentar la comodidad del usuario y abordar diversos fallos presentes en el desarrollo anterior, así como corregir secciones que contaban con una implementación deficiente. El objetivo es proporcionar una experiencia de usuario más intuitiva y efectiva, facilitando el acceso a la información y el control de los niveles acústicos en locales.

A lo largo del proyecto, se han tomado decisiones estratégicas respecto a las tecnologías y herramientas a emplear, asegurando que el sistema sea robusto y sostenible para futuras evoluciones. La generación de documentación también ha sido una prioridad, ya que proporciona a los futuros desarrolladores un soporte fundamental para el mantenimiento y la mejora continua del sistema.

1.4 Resumen de objetivo

A la vista de lo descrito en el marco del proyecto, el objetivo principal es mejorar un sistema de monitoreo acústico para locales mediante la optimización del servidor y la implementación de un simulador de limitador de ruido. Este objetivo se puede dividir en varios objetivos más específicos, a partir de los cuales se derivan todos los requisitos del sistema:

1. Establecer un entorno de trabajo basado en grupos de contenedores que optimicen el rendimiento del servidor
2. Desarrollar un simulador de limitador de ruido utilizando Arduino que envíe datos de prueba al servidor.
3. Mejorar la plataforma web existente para facilitar la experiencia del usuario y corregir fallos de implementación previos.
4. Generar documentación que sirva como guía para futuros desarrolladores en el mantenimiento y evolución del sistema.
5. Asegurar la escalabilidad del sistema para permitir futuras mejoras y adaptaciones.

2. Estado del arte

2.1 Introducción al estado del arte

En el mundo actual, la gestión de datos y la monitorización ambiental son fundamentales para garantizar un entorno seguro y saludable, especialmente en espacios comerciales y públicos. Con el creciente interés en la sostenibilidad y el bienestar de la comunidad, los sistemas de monitoreo acústico han cobrado relevancia como herramientas clave para el control de los niveles de ruido en los locales. Sin embargo, ¿qué soluciones existen en la actualidad para abordar estos desafíos? ¿Cómo se diferencia un sistema de monitoreo acústico de otros enfoques en la gestión de datos ambientales? Y más específicamente, ¿qué aspectos caracterizan este proyecto en relación con otros sistemas de monitoreo acústico?

A menudo, se asocia la tecnología de monitoreo con sensores y dispositivos especializados, pero hoy en día se requiere un enfoque más integrado que combine la recopilación de datos, su análisis y la respuesta adecuada. En este contexto, la utilización de plataformas que gestionan datos en tiempo real y permiten la interacción con el usuario es esencial. Actualmente, el sistema se encuentra implementado en un servidor utilizando PM2, lo que ha presentado grandes dificultades para su modificación y mejora. Este enfoque no proporciona un entorno de desarrollo ágil, ya que los desarrolladores no cuentan con un sistema de contenedores que les permita descargar y lanzar la aplicación de forma local, limitando así la capacidad de realizar cambios de manera eficiente.

Este proyecto busca no solo mejorar un sistema existente de monitoreo acústico, sino también crear un entorno de simulación que permita a los desarrolladores y usuarios experimentar con datos acústicos de manera segura y controlada. A través de la dockerización del sistema, se espera facilitar el desarrollo y la implementación de mejoras, contribuyendo así a la evolución de la tecnología de monitoreo ambiental y ofreciendo herramientas más accesibles y efectivas para la gestión del ruido en diversos entornos.

2.1 Sistema del proyecto

A continuación, se presenta un análisis del estado del arte en relación con el proyecto de gestión de un sistema de monitoreo acústico. Se abordarán las tecnologías de monitoreo acústico existentes y su aplicación práctica, así como las ventajas de utilizar contenedores para facilitar la gestión y modificación del sistema. También se explorará el uso de Arduino para simular datos, y se analizarán las mejoras en la interfaz de usuario y la importancia de una buena documentación para el mantenimiento del sistema. Este análisis proporcionará un contexto relevante para entender la necesidad y el enfoque del proyecto en el ámbito actual.

2.1.1 Tecnologías de monitoreo acústico

El monitoreo acústico es fundamental para la gestión del ruido en entornos urbanos e industriales, y existen diversas tecnologías que permiten la captura y análisis de datos acústicos en tiempo real. Entre los dispositivos más utilizados se encuentran los micrófonos, que pueden ser omnidireccionales o direccionales, seleccionados según el tipo de ruido que se desea medir y las condiciones del entorno.

En este contexto, los limitadores de ruido juegan un papel crucial. En particular, los dispositivos CAP21 y Ecudap WatchDog SR 50 son ejemplos destacados de sistemas que no solo miden los niveles acústicos, sino que también actúan para controlar la emisión de ruido, asegurando que se mantengan dentro de límites aceptables. Estos limitadores integran algoritmos avanzados que permiten el procesamiento de los datos acústicos y la toma de decisiones en tiempo real.

Las aplicaciones en tiempo real de estas tecnologías abarcan desde la supervisión de eventos en vivo hasta la regulación del ruido en áreas residenciales. La capacidad de gestionar datos acústicos en tiempo real permite a autoridades y empresas implementar medidas correctivas eficaces para mitigar el ruido y mantener la calidad de vida en sus comunidades.

El presente documento explica el desarrollo de un sistema de prevención de intrusiones (IPS) capaz de analizar el tráfico de un dispositivo para extraer y almacenar información relevante como el origen de un evento, el dispositivo afectado, el puerto de red en el que se ha detectado y la información del evento. Con esta información, se determinará la gravedad de un evento para poder así generar y aplicar las reglas necesarias de cortafuegos sobre un router Mikrotik para bloquear el tráfico entrante al dispositivo.

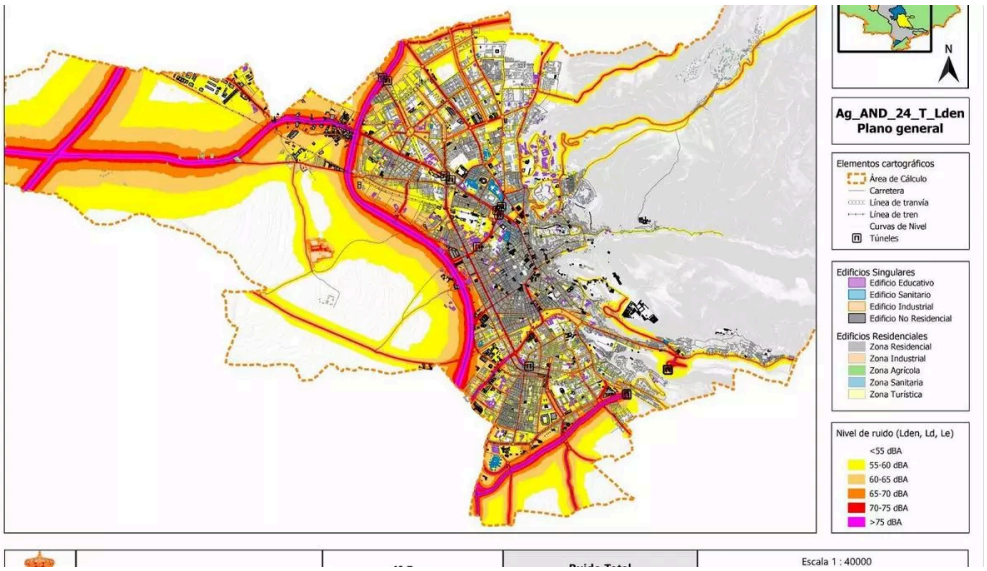


Figura 1: Mapa acústico de Granada 2023. [11].

2.1.2 Contenedores y virtualización

La virtualización y el uso de contenedores, como Docker, han transformado el desarrollo y despliegue de aplicaciones en entornos de producción. Los contenedores permiten empaquetar aplicaciones y sus dependencias en un entorno portable, lo que simplifica la implementación y gestión de sistemas complejos. A diferencia de métodos tradicionales como PM2, que se centran en la gestión de procesos en un solo servidor, Docker ofrece la capacidad de crear entornos de desarrollo consistentes que se pueden replicar fácilmente en diferentes plataformas, desde producción hasta desarrollo local.

Entre los beneficios de utilizar contenedores se encuentran la escalabilidad, el aislamiento de aplicaciones y la facilidad para implementar integración continua y entrega continua (CI/CD). Esta metodología permite realizar actualizaciones y modificaciones de manera más eficiente, facilitando el mantenimiento y la evolución de sistemas en producción. Existen numerosos casos de estudio que demuestran la efectividad de estas tecnologías en contextos similares, donde se ha logrado mejorar la flexibilidad y la eficiencia operativa, favoreciendo la colaboración entre los equipos de desarrollo y operaciones. Todas estas ventajas se reflejan en la siguiente tabla:

Característica	PM2	Docker Compose
Ejecución de Servicios	Ejecuta la aplicación Node.js directamente como proceso.	Maneja múltiples servicios en contenedores (Node.js, MySQL, phpMyAdmin).
Reinicio Automático	Reinicia automáticamente la app si se cae.	Reinicia los contenedores según la política definida (ej. siempre, a la hora de inicio).
Configuración	Configuración manual de cada proceso.	Archivo de configuración (docker-compose.yml) permite gestionar todos los servicios fácilmente.
Escalabilidad	Escala la app usando clústeres, pero limita la capacidad de otros servicios.	Permite escalar todos los servicios y gestionarlos de forma independiente.

Aislamiento	Menor aislamiento entre procesos, todo se ejecuta en el mismo entorno.	Aislamiento completo de cada servicio en su propio contenedor.
Facilidad de Desarrollo	Necesita configuración adicional para replicar el entorno de producción localmente.	Facilita la replicación del entorno de producción en desarrollo, permitiendo iniciar servicios con un solo comando.
Manejo de Dependencias	Las dependencias deben manejarse en el sistema operativo.	Las dependencias se definen en el Dockerfile o en el archivo docker-compose.yml, asegurando un entorno consistente.

Tabla 1: Comparativa entre PM2 y Docker Compose. Creación propia.

3. Estado inicial

Este trabajo tiene como objetivo mejorar y optimizar un sistema de monitoreo acústico utilizado en locales comerciales para supervisar los niveles de ruido. Este sistema permite recoger y analizar datos acústicos en tiempo real, proporcionando un control efectivo sobre el cumplimiento de los límites de ruido permitidos. Sin embargo, antes de abordar las modificaciones y mejoras implementadas, es fundamental comprender la configuración y funcionamiento inicial del sistema, así como los desafíos que presentaba para su mantenimiento y desarrollo.

3.1 Arquitectura de Proxy Inverso con Gestión de Procesos:

La arquitectura del servidor en este proyecto está diseñada para optimizar la gestión de procesos y facilitar la interacción entre distintos servicios. En el corazón de esta arquitectura se encuentra PM2, un gestor de procesos que permite ejecutar y mantener en funcionamiento la aplicación Node.js de manera eficiente. Este servidor se conecta a una base de datos MySQL, desde la cual la aplicación Node.js accede a los datos necesarios a través de una API local.

La aplicación Node.js desempeña un papel fundamental, ya que gestiona múltiples servicios críticos. En primer lugar, lanza una aplicación Angular que presenta datos y estadísticas sobre los limitadores de sonido, ofreciendo una interfaz intuitiva para la gestión de estos dispositivos. Además, este mismo programa está configurado para escuchar en un puerto específico, recibiendo datos de los limitadores de sonido y procesarlos en tiempo real. Por otro lado, la aplicación también es responsable de emitir datos acústicos a ciertos ayuntamientos, facilitando así la comunicación y el cumplimiento de normativas locales.

En este contexto, se pueden identificar tres componentes clave dentro de la arquitectura: **dbaDashboard**, que es la interfaz de usuario para la gestión de limitadores; **dbADataReceiver**, encargado de recibir los datos provenientes de los limitadores; y **dbAEmitter**, que se ocupa de enviar datos específicos a los ayuntamientos. Todos estos servicios son gestionados y supervisados por PM2, lo que garantiza su estabilidad y disponibilidad en todo momento.

Para acceder a la aplicación desde el exterior, se utiliza el dominio **heimdalsoundcontrol.com**, que actúa como punto de entrada. Este dominio es administrado por un proxy inverso configurado con Nginx, que redirige las solicitudes a la aplicación. Esta estructura no solo facilita la gestión de tráfico y la seguridad, sino que también permite una mayor escalabilidad y flexibilidad en la implementación de nuevos servicios o cambios en la arquitectura del servidor.

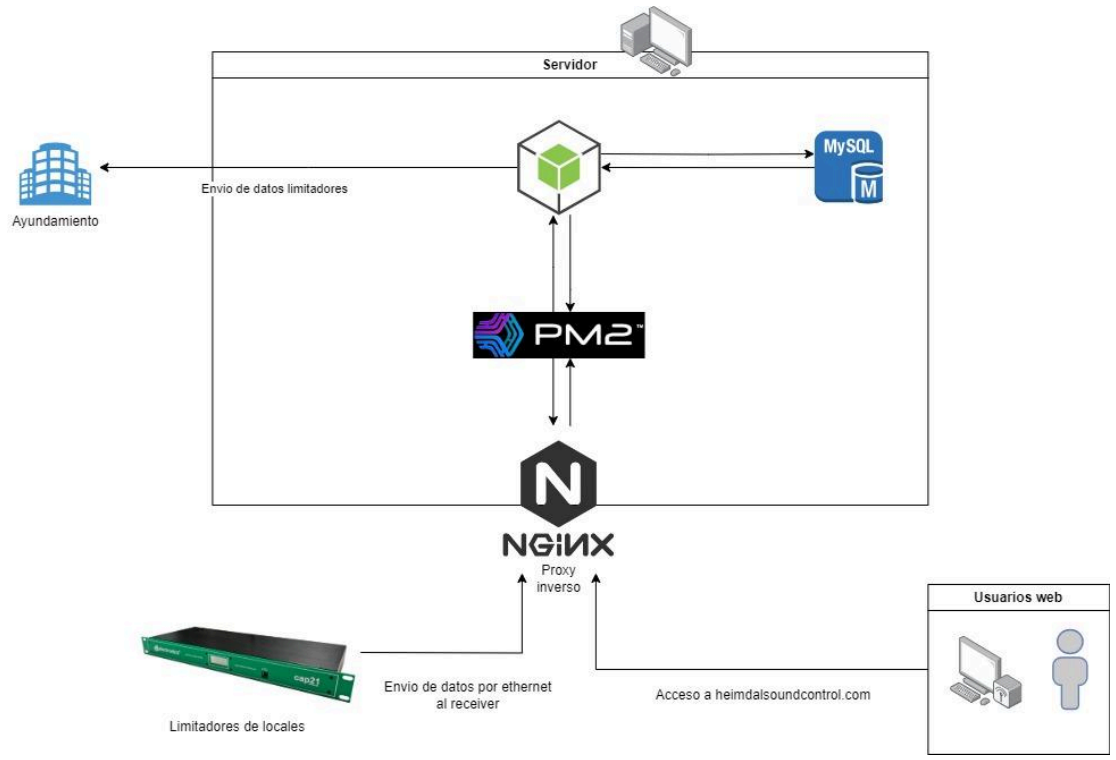


Figura 2. Organigrama del servidor.

Elaboración propia usando draw.io [6].

3.2 Planificación base de datos

La base de datos de este sistema es el núcleo de almacenamiento de datos, diseñada para gestionar eficientemente la información generada y consultada por los dispositivos y usuarios. Está estructurada de manera simple con pocas relaciones entre entidades, permitiendo registrar desde datos de usuario hasta el histórico de mediciones, configuraciones de dispositivos y estadísticas de rendimiento. Cada tabla tiene un propósito específico y, en conjunto, reflejan la complejidad y alcance de las funcionalidades requeridas en el sistema.

A continuación, se presentan los diagramas entidad-relación que muestran las distintas tablas de la base de datos. Estos diagramas permiten visualizar la estructura de cada tabla y las relaciones clave entre ellas, proporcionando una perspectiva clara de cómo se organiza y conecta la información en el sistema. Siendo estas relaciones bastante escasas, con un sistema pobre de casi nula conexión entre tablas requeridas.


users
id int NN
username varchar(80)
password varchar(80)
salt varchar(50)
user_group tinyint
firstname varchar(45)
lastname varchar(45)
phone varchar(45)
email varchar(45)
ccemail varchar(200)
installer int
company int
last_login datetime
enabled int
created_by int
created_date datetime
exist_token tinyint(1)
sessions
id int NN
timestamp datetime
ip varchar(20)
port smallint
raw blob
serialno varchar(45)
initdate datetime
enddate datetime
session smallint
pinknoisespectrum varchar(100)
maxpinknoise tinyint
minpinknoise tinyint
sessionstate tinyint
messageno tinyint
limiter varchar(50)
autocreated tinyint
maxSPL tinyint
minSPL tinyint
companies
id int NN
companyname varchar(80)
manager int
stock_user int
type tinyint
created_by int
created_date datetime
url_stations varchar(45)
appTokens
id int NN
client int
token varchar(50)
created_by int
created_date datetime
safeUsers
ccemail varchar(200)
company int
created_by int
created_date datetime
email varchar(45)
enabled int
firstname varchar(45)
id int
installer int
last_login datetime
lastname varchar(45)
phone varchar(45)
tokenexist tinyint(1)
user_group tinyint
username varchar(80)
user_mediatorServer
id int NN
username varchar(80)
password varchar(80)
firstname varchar(45)
lastname varchar(45)
phone varchar(45)
email varchar(45)
company varchar(80)
enabled int
last_login datetime
cont_logins int
recoveryTokens
id int NN
token varchar(80)
timestamp datetime
email varchar(80)
used tinyint

Figuras 3 y 4. Entidades de la base de datos. Elaboración propia usando dbdiagram.io [13].

3.3 Funcionamiento aplicación web

En este apartado se detalla cómo está configurada actualmente la interacción entre los usuarios y los datos de los dispositivos instalados en los distintos locales. Esta aplicación no permite configuraciones directas en cada dispositivo; sin embargo, el usuario puede acceder a la información de los dispositivos de cada local y recibir datos actualizados sobre su actividad. Asimismo, se incluyen funcionalidades para enviar alertas en caso de que se sobrepase el límite acústico permitido, y en ciertos casos, se envían estos datos a los ayuntamientos correspondientes para asegurar el cumplimiento de las normativas de ruido.

Como ya expliqué anteriormente en otro punto, la aplicación está organizada en tres funciones principales que se ejecutan desde el mismo servicio, cada una encargada de un aspecto clave de su funcionamiento. La primera es **dbADashboard**, que abarca tanto el backend, mediante una API en local, como la aplicación de Angular, desde donde los usuarios pueden visualizar y gestionar datos y estadísticas de los limitadores. Aunque también existe una función denominada dbAEmitter, esta se encuentra actualmente en desuso, ya que su propósito era enviar los datos de los limitadores a los ayuntamientos en aquellos casos donde los dispositivos no podían remitir la información a múltiples destinos a la vez. Por último, **dbADataReceiver** abre un puerto para recibir cada cinco minutos los datos de sonido enviados por los limitadores, incluyendo valores como el nivel de ruido actual, el promedio en intervalos de diez minutos y una hora, así como los registros de ruido mínimo y máximo del día.

3.3.1 DbADataReceiver

A continuación, se presenta un diagrama de flujo que ilustra el proceso de manejo de datos en el servidor de la aplicación. Este flujo describe cómo el sistema se conecta a los dispositivos, recibe información y gestiona diferentes tipos de eventos, como conexiones y desconexiones. Además, se detallan las acciones que se llevan a cabo al recibir datos, incluyendo el almacenamiento de mediciones y sesiones en la base de datos, lo que permite un monitoreo efectivo de la información acústica en tiempo real.

En el archivo de configuración del servidor, se establece la conexión con los dispositivos mediante la definición de la dirección y el puerto a utilizar. Una vez que se establece la conexión, el servidor inicia un proceso para escuchar los datos que llegan desde los dispositivos. Al recibir información, el sistema clasifica los eventos en diferentes categorías, como conexión, desconexión o transmisión de datos. Dependiendo del tipo de evento, se registra la actividad y se actualizan los detalles de la conexión, así como se almacenan las mediciones y sesiones en la base de datos. Este flujo de manejo de datos garantiza que se mantenga un registro adecuado de las interacciones con los dispositivos, facilitando así el monitoreo y la gestión de la información acústica recibida.

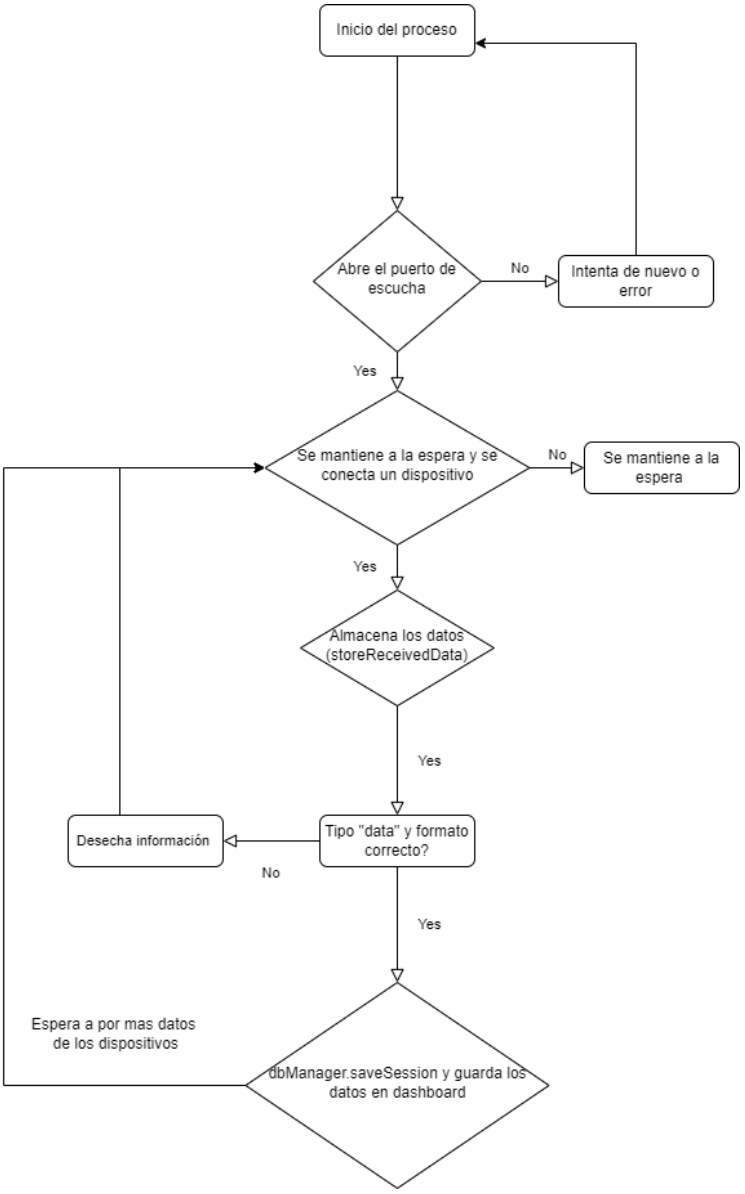


Figura 5. Diagrama de flujo de DataReciever. Elaboración propia usando draw.io [6].

En el archivo de configuración del servidor, se establece la conexión con los dispositivos mediante la definición de la dirección y el puerto a utilizar. Una vez que se establece la conexión, el servidor inicia un proceso para escuchar los datos que llegan desde los dispositivos. Al recibir información, el sistema clasifica los eventos en diferentes categorías, como conexión, desconexión o transmisión de datos. Dependiendo del tipo de evento, se registra la actividad y se actualizan los detalles de la conexión, así como se almacenan las mediciones y sesiones en la base de datos. Este flujo de manejo de datos garantiza que se mantenga un registro adecuado de las interacciones con los dispositivos, facilitando así el monitoreo y la gestión de la información acústica recibida.

3.3.2 DbADashboard

El **dbaDashboard** es la aplicación principal del sistema, diseñada específicamente para la gestión y visualización de datos acústicos en tiempo real. Su interfaz intuitiva permite a los usuarios interactuar de manera eficiente con la información, facilitando la toma de decisiones basada en datos precisos y actualizados. La arquitectura del dashboard está construida sobre un entorno robusto que utiliza **Node.js** para el backend y **Angular** para el frontend, integrando librerías como **node_modules** y **bower.js** para optimizar el desarrollo y la funcionalidad de la aplicación. Esta estructura permite una fácil escalabilidad y mantenimiento, asegurando que la aplicación se adapte a las necesidades cambiantes de los usuarios y del sistema.

La siguiente imagen presenta un mapa de la organización de árbol de los archivos del proyecto. Este diagrama proporciona una visión clara de la estructura de directorios y archivos, lo que facilita la comprensión de la jerarquía y la relación entre los diferentes componentes del sistema.

La estructura de archivos de la aplicación está organizada para facilitar la gestión tanto del frontend como del backend, cada uno con su propia carpeta dedicada, lo que permite un desarrollo claro y modular. Visible también en el organigrama del árbol de la figura [6]

- ***/api***: Esta carpeta contiene el código del backend, que incluye todas las rutas, controladores, modelos y servicios que gestionan la lógica de las peticiones hacia la base de datos principalmente. Aquí se encuentran también los archivos relacionados con la configuración del servidor, como las rutas de la API para interactuar con el frontend y realizar operaciones en la base de datos.

- /***app***: Esta carpeta alberga todo el código del frontend, desarrollado principalmente en Angular. Contiene los componentes, servicios y módulos que gestionan la interfaz de usuario y las interacciones con el backend, como la visualización de datos y el manejo de eventos en tiempo real.

    - /**app/directives**: Dentro de esta carpeta se encuentran los archivos de los menús tanto el superior como el menú lateral que serán reutilizados.:

        - /**header**: Contiene los archivos HTML y JavaScript que definen el menú de navegación superior. Este menú incluye el icono de la aplicación, así como un pequeño menú a la izquierda del email y usuario.

        - /**sidebar**: Es un menú lateral que da acceso a las diferentes secciones de la aplicación, lista de usuarios, dispositivos, ficheros de manuales… .

    - /**app.js**: Es el componente central en la configuración del frontend. Es el punto de entrada donde se define la configuración del módulo principal de la aplicación y las rutas, así como algunos servicios globales y controladores.

    - /**index.html**: Principalmente es la plantilla básica de la aplicación, donde

se cargan los js de todos los controladores.

    - **./views**: En este punto se agrupan las carpetas views y js, ya que sus archivos están estrechamente relacionados, siendo el html y su js de cada sección de la aplicación. En el html está la plantilla visual la cual recibe datos y es lanzada por el controlador del archivo js.

- ***/public***: Aquí se encuentran los archivos estáticos que la aplicación utiliza para renderizar la interfaz de usuario. Esto incluye imágenes, hojas de estilo CSS, y scripts JS que no se gestionan a través de Angular. Además funciones de la web como el **login**, el **recover password** y el **findyourtoken**.

- ***/utils***: Son archivos simples como la conexión a la base de datos, el validador y logger (archivo de impresión console.logs).

- /***node_modules*** y /***bower_components***: Estas carpetas almacenan todas las dependencias de Node.js y los componentes de Bower que se han instalado para el correcto funcionamiento de la aplicación, tanto en el frontend como en el backend. La carpeta /node_modules contiene las librerías y paquetes de Node.js, mientras que /bower_components almacena las bibliotecas gestionadas por Bower. Es importante destacar que estas carpetas no deben modificarse manualmente, ya que su contenido se gestiona automáticamente mediante los respectivos gestores de paquetes.

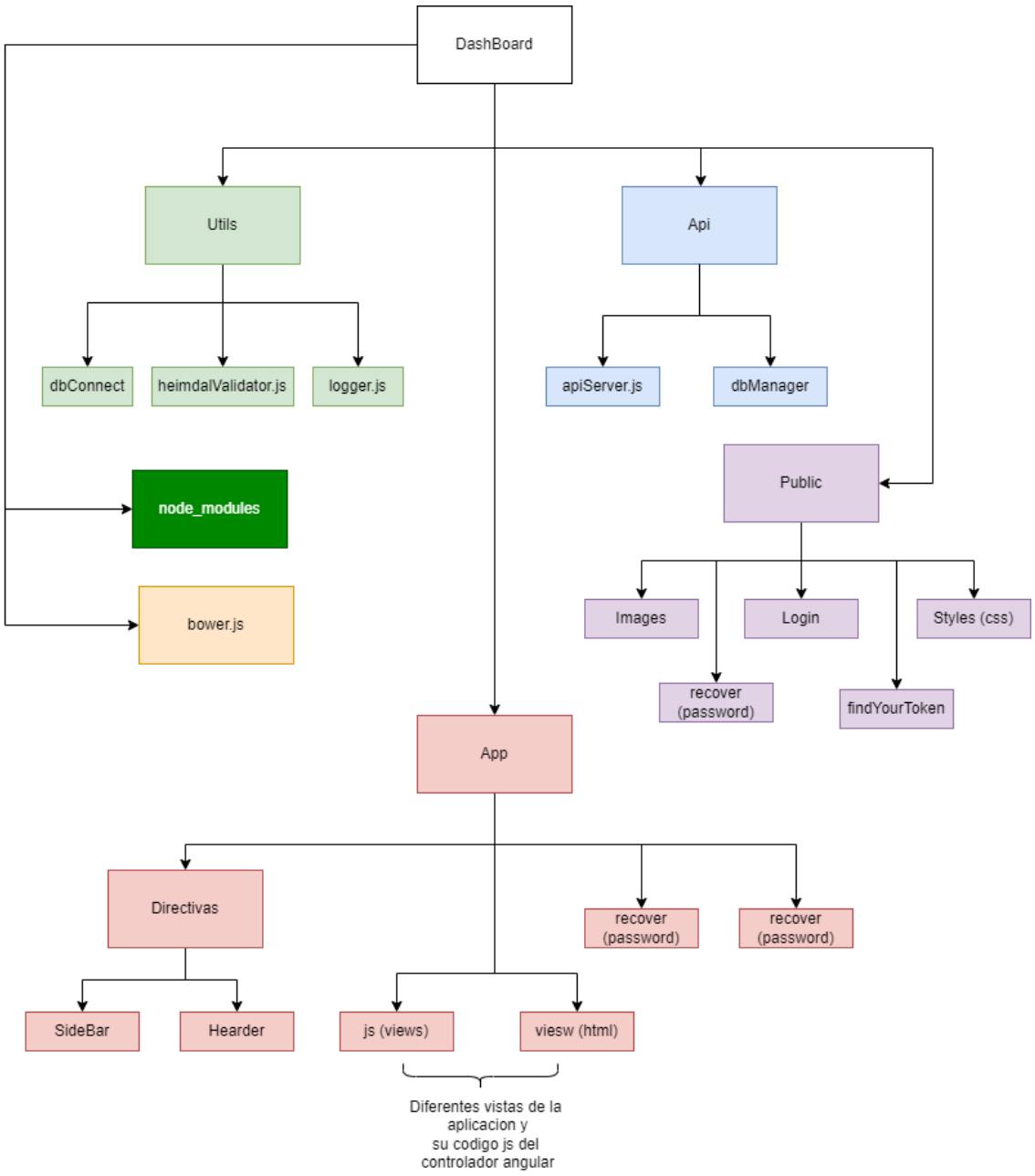


Figura 6. Diagrama de representación árbol de archivos . Elaboración propia usando draw.io [6].

La aplicación web principalmente está diseñada para brindar a cada usuario una visión global de sus dispositivos y de su actividad en tiempo real. La interfaz principal o *dashboard*, como se muestra en la primera captura (Figura 7), despliega un listado de los dispositivos gestionados. En el caso de un usuario estándar, se muestran exclusivamente los dispositivos que tiene asignados en propiedad. Sin embargo, los usuarios con un rango superior tienen acceso a todos los dispositivos, permitiéndoles gestionar y monitorear el estado de los equipos sin limitaciones.

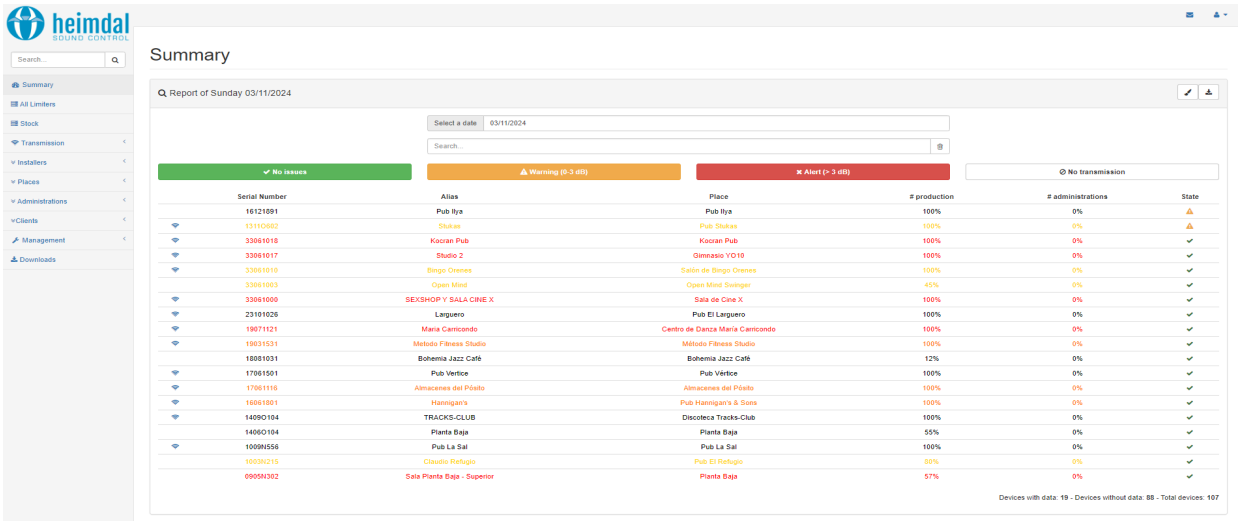


Figura 7. Página principal HeimdalSoundControl.com [10].

Por otro lado, la segunda captura (Figura 7) muestra la vista específica de un dispositivo. Esta pantalla permite a los usuarios visualizar detalles individuales del dispositivo, incluyendo su actividad en términos de nivel de ruido en intervalos de tiempo definidos. En este caso, se observa un gráfico que representa el nivel de ruido medido en el dispositivo en un rango de un día, facilitando así el seguimiento puntual y la gestión de alertas acústicas en tiempo real.

Estas capturas reflejan la estructura inicial del sistema antes de la incorporación de las mejoras y cambios actuales, evidenciando tanto las funcionalidades básicas de monitoreo con gráficos como algunos ajustes en el panel lateral izquierdo..

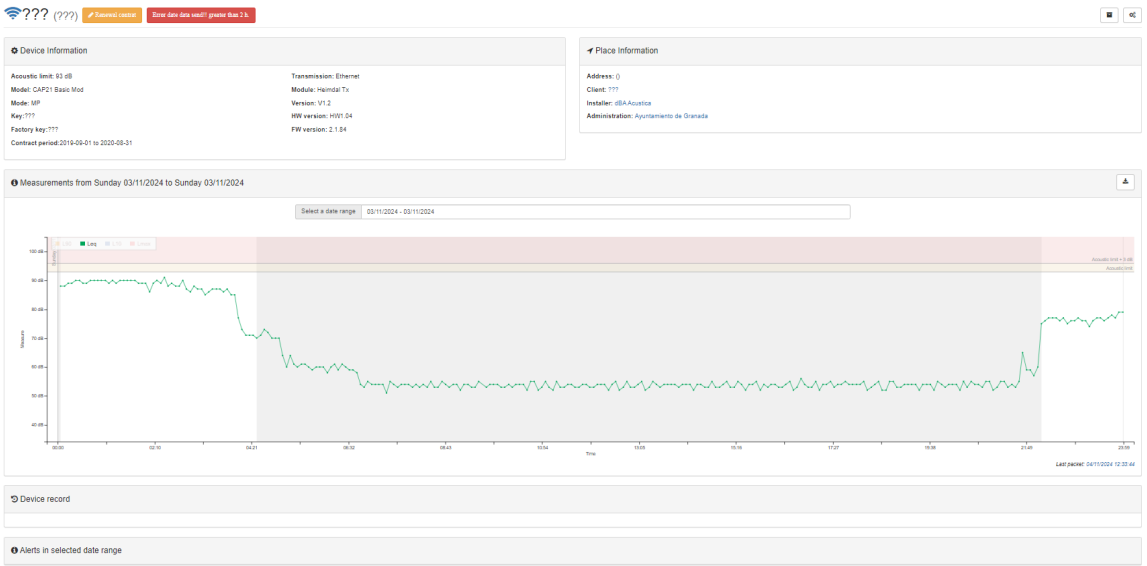


Figura 8. Apartado visualizado diagrama de datos sonido de un limitador en un local en activo. [10].

Por motivos de privacidad y confidencialidad, los nombres y datos específicos del local en Granada han sido sustituidos en este documento. Esta medida se ha tomado para proteger la identidad de las personas y la información sensible del establecimiento,

asegurando que el contenido cumpla con los estándares éticos y legales en el tratamiento de datos.

3.4 Interacción y Limitaciones del Sistema:

El sistema actual tiene sus raíces en un trabajo fin de grado (TFG) previo, que fue posteriormente modificado por otros alumnos sin seguir un flujo de trabajo estructurado ni una práctica adecuada de documentación. Esto ha llevado a una situación en la que abundan los errores y se dificulta enormemente la comprensión y el mantenimiento del proyecto. Uno de los aspectos más problemáticos es la falta de un entorno virtualizado o de contenedores que permita a los desarrolladores trabajar sin interferir en el entorno de producción, lo cual complica la implementación de cambios y el testeo, exponiendo el sistema a posibles fallos.

La configuración de la base de datos es otro punto crítico en el proyecto. No solo es difícil de replicar en un entorno local debido a la falta de virtualización y documentación, sino que también carece de diagramas entidad-relación que permitan entender la estructura de las tablas y sus relaciones. Este escenario obliga a los desarrolladores a investigar la estructura desde cero, ralentizando considerablemente su flujo de trabajo y exponiéndose a errores al desconocer detalles de las interacciones entre los datos. Además, no existe un entorno de pruebas o bases de datos de desarrollo, por lo que cada modificación puede afectar directamente la base de datos de producción si no se tiene extrema cautela.

La falta de documentación afecta otras áreas fundamentales del sistema. Por ejemplo, el uso de PM2 para la gestión de procesos no se encuentra adecuadamente configurado para asegurar la portabilidad entre entornos, y las configuraciones de red, como el proxy inverso mediante Nginx, tuvieron que ser descubiertas mediante prueba y error. Sin documentación que explique el rol de Nginx o su configuración, los desarrolladores no contaban con una guía sobre cómo redirigir las solicitudes a la URL correcta.

Finalmente, no existe documentación técnica sobre el despliegue del sistema, la estructura del proyecto, o manuales que describan cómo deberían llevarse a cabo las configuraciones iniciales y el despliegue en un entorno de producción. Tampoco se cuenta con un esquema del árbol de directorios ni guías para nuevos desarrolladores, lo que hace que cada incorporación al equipo sea un proceso largo y complejo. La combinación de modificaciones no documentadas y el desorden en la estructura técnica y organizativa ha convertido el trabajo con este sistema en una tarea difícil y propensa a errores, que requiere de un esfuerzo considerable para cualquier desarrollador que se adentre en el proyecto.

4. Objetivos

El objetivo principal del proyecto es mejorar un sistema software existente del grupo HEIMDAL, con una aplicación web para gestionar el monitoreo acústico en locales. Este sistema permitirá interactuar con dispositivos de captura de datos y simuladores de limitadores de ruido, procesando la información para asegurar que los niveles acústicos no excedan los límites. Además, se implementará una infraestructura en contenedores que optimice el servidor y facilite la gestión de datos, mejorando la experiencia del usuario.

Para lograr esto, es fundamental establecer una serie de objetivos que se pueden clasificar según el tipo de actividades necesarias para su consecución. Estas clasificaciones son las siguientes:

- **Objetivos de investigación**: se refieren a aquellos propósitos que implican un alto grado de incertidumbre y que requieren un tiempo indefinido para su finalización. Estos objetivos pueden incluir la adquisición de conocimientos sobre herramientas, así como el estudio de tecnologías y dispositivos relevantes.

- **Objetivos de desarrollo**: abarcan las etapas de diseño, prueba, implementación e integración del sistema con otros sistemas existentes. Estos objetivos comprenden el funcionamiento interno de la aplicación, que es lo que se busca alcanzar, incluyendo también el desarrollo de la aplicación web.

4.1 Objetivos de investigación

1. Aprender de la estructura no documentada del proyecto en el servidor.
2. Investigar diferentes herramientas de control de servicios.
3. Desarrollar en profundidad los conocimientos de Docker y Docker-compose para desarrollar .
4. Investigar herramientas y librerías para la creación del imitador del limitador.
5. Investigar el formato binario de como mandan los datos los limitadores al servidor (formato).
6. Investigar diferentes herramientas de administradores de base de datos.
7. Aprender Angular y sus librerías usadas para las mejoras del servidor.
8. Aprender Nginx para modificar el proxy del servidor.
9. Investigar diferentes herramientas y librerías para el arduino uno relacionado con la gestión de fechas y envío de datos por ethernet a la red.

4.2 Objetivos de desarrollo

1. Creación del nuevo diagrama del sistema con los servicios y los contenedores en vez del uso de pm2.
2. Configurar los contenedores que se usarán para adaptar el sistema correctamente.
3. Programar correctamente el fichero docker-compose.yml con los 3 contenedores y el .env.
4. Configurar correctamente el contenedor mysql para adaptarlo a una versión adecuada a las librerías usadas en la aplicación web.
5. Creación de un contenedor de gestión de la base de datos.
6. Creación de un programa arduino para el imitador del limitador de sonido.
7. Configurar el ordenador (local) y el código de arduino para que se comuniquen por TCP compartiendo internet desde local.
8. Sustituir el sistema de servicios con pm2 al uso de docker correctamente.
9. Implementar mejoras en la aplicación web de nodejs y Angularjs así como modificar la base de datos en consecuencia.
10. Documentar las mejoras en la aplicación web y el arduino con Doxygen.
11. Documentar el funcionamiento de la aplicación y su árbol de archivos para futuros desarrolladores.
12. Documentar el método de despliegue de los servicios.

5. Planificación

A diferencia de otros proyectos que suelen enfrentar una gran incertidumbre en sus etapas iniciales, este cuenta desde el principio con ideas claras, como mejorar el sistema de control del servidor mediante contenedores y desarrollar un simulador de limitadores de ruido. Sin embargo, existe una considerable incertidumbre en cuanto a la comprensión de su configuración actual y en aspectos que faciliten el trabajo del desarrollador. Por ello, una planificación cuidadosa se vuelve esencial para optimizar los procesos y alcanzar los objetivos de manera eficiente. En esta sección, se detallarán los elementos utilizados para identificar recursos, establecer plazos y descomponer los objetivos en tareas concretas, componiendo así la planificación general definida para el proyecto.

5.1 Metodología de desarrollo

A lo largo del tiempo, el desarrollo de software ha evolucionado, dejando atrás modelos tradicionales como el enfoque en cascada y adoptando metodologías y marcos de trabajo ágiles, que permiten adaptarse mejor a los cambios frecuentes y rápidos en el entorno. Dado el enfoque práctico y flexible de este proyecto, se ha optado por utilizar el marco de trabajo ágil Kanban, que facilita una adaptación constante a las necesidades que puedan surgir durante el desarrollo.

Kanban se define como un enfoque flexible de gestión de trabajo, centrado en el flujo continuo de tareas sin la necesidad de ciclos o fechas de finalización predefinidas. En lugar de trabajar con Sprints y reuniones estrictas como en otros marcos ágiles, Kanban permite gestionar el flujo de trabajo visualizando las tareas en un tablero y limitando la cantidad de trabajo en progreso (WIP, por sus siglas en inglés). Esto permite al desarrollador ajustar la carga de trabajo de manera eficiente, priorizando las tareas según la capacidad disponible y manteniendo un enfoque ágil y continuo a lo largo del proyecto.

En este proyecto, dado que se cuenta con un único desarrollador, el proceso de trabajo se adapta de manera flexible, priorizando tareas según la demanda y ajustando el flujo de trabajo en función de las necesidades del momento. Aunque Kanban no requiere reuniones formales como en otros marcos ágiles, se realiza un seguimiento constante del progreso mediante la revisión del tablero Kanban, asegurando que las tareas se gestionen de manera eficiente sin sobrecargar al desarrollador.

5.2 Iteraciones

- **Planificación continua**: En lugar de realizar una planificación formal por etapas o sprints, en Kanban las tareas se gestionan de manera continua, añadiendo y priorizándolos en el tablero Kanban según las necesidades que vayan surgiendo. No hay plazos fijos para las tareas, y el flujo de trabajo se ajusta conforme avanza el proyecto.
- **Control del progreso sin reuniones diarias**: A diferencia de otros métodos ágiles, en Kanban no se realizan reuniones diarias de seguimiento. El control del progreso se realiza visualmente a través del tablero Kanban, donde el

desarrollador puede revisar el estado de las tareas de manera continua y tomar decisiones en tiempo real.

- **Revisión continua**: En lugar de una reunión formal de revisión al final de cada etapa, Kanban permite una evaluación constante del flujo de trabajo a medida que las tareas avanzan. Cualquier ajuste necesario se realiza en tiempo real, permitiendo una gestión dinámica del trabajo.

- **Mejora continua**: Aunque no se realiza una retrospectiva formal, Kanban promueve la mejora continua. El flujo de trabajo se revisa constantemente para identificar cuellos de botella o posibles áreas de mejora, adaptando el proceso de trabajo según las necesidades del proyecto

5.3 Método Kanban

Kanban es una herramienta eficaz para analizar y gestionar el flujo de trabajo en diferentes fases. Aunque no utiliza sprints tradicionales, en este proyecto se emplea Kanban como el enfoque principal para gestionar el trabajo de manera continua y visual. En lugar de ciclos de trabajo rígidos, el progreso se evalúa de forma continua a través de un **tablero Kanban**, que permite visualizar el estado de las tareas en tiempo real: aquellas que aún no se han iniciado, las que están en desarrollo, las pendientes de revisión y las que enfrentan bloqueos.

Aunque Kanban ofrece flexibilidad en la gestión continua, debido a que el equipo está compuesto por un solo desarrollador, las tareas se organizan en bloques que funcionan de manera similar a los **sprints**, pero sin la rigidez de los plazos estrictos. Este enfoque asegura que el trabajo se gestione de manera estructurada, manteniendo una visión clara del progreso y facilitando la adaptación a las necesidades del proyecto.

Además, se utilizará un **Product Backlog** para agrupar las **historias de usuario**, lo que permite priorizar las tareas según las necesidades del proyecto. Las tareas en el **Product Backlog** se irán moviendo a través del tablero Kanban, facilitando el seguimiento del flujo de trabajo y proporcionando visibilidad sobre lo que está pendiente, en proceso, revisado o bloqueado.

El uso de Kanban en combinación con los bloques organizados de trabajo mejora la visibilidad del flujo y permite una gestión más ágil, adaptándose rápidamente a los cambios y optimizando la distribución de las tareas. Este enfoque asegura que, aunque el proyecto esté gestionado por un único desarrollador, se mantenga un proceso de desarrollo ágil, con flexibilidad para reaccionar ante cambios y mejoras continuas.

5.3 Historias de usuario

Proyecto de mejora en tres puntos clave:

1. **Dockerización del servidor**: Reestructurar la gestión de dependencias y servicios del servidor en una arquitectura de tres partes mediante Docker.
2. **Creación de un simulador de limitador de sonido**: Diseñar un imitador para futuras pruebas acústicas.
3. **Mejoras en la aplicación Node.js y AngularJS**: Optimizar funcionalidades y resolver problemas de código inestable.

Documentación como prioridad

Dado que la documentación actual es inexistente y el proyecto original fue desarrollado sin consistencia por varios desarrolladores, crear una guía clara es esencial para facilitar futuras modificaciones y ayudar al equipo de GranaSat[14].

Al comenzar este proyecto, partí de una base donde ya existía una implementación previa, y mi objetivo ha sido centrarme principalmente en realizar mejoras tanto en el servidor como en la funcionalidad del sistema. Una de las principales tareas fue crear un imitador de limitador de sonido que sirviera para futuras pruebas, lo que requirió la implementación de diversas funcionalidades específicas.

Base de trabajo
Comenzando desde una implementación previa, el enfoque se centró en:

- Mejorar el servidor y la estructura de la aplicación.
- Implementar el simulador de limitador de sonido.

Historias de usuario para mejoras en el sistema
Las siguientes historias de usuario reflejan ajustes técnicos en el servidor, considerando la evolución y depuración de un código con poca estructura, conocido como "código espagueti". Estas mejoras incluyen:

- **Priorización**: Las historias con menor valor numérico son más urgentes.
- **Estimación de esfuerzo**: Tiempo de programación en horas, ajustado según la complejidad del sistema anterior.
-

Identificador: HU.1	**Título**: Analizar estado del servidor y aplicación
Prioridad: 1	**Tiempo estimado**: 3 PH
Descripción: Como desarrollador, quiero tener una vista limpia del servidor y aplicación para su futuro desarrollo	

Identificador: HU.2	**Título**: Documentar aplicación
Prioridad: 2	**Tiempo estimado**: 2 PH
Descripción: Como desarrollador, quiero tener documentado el funcionamiento de la aplicación para poder modificarla en el futuro	

Identificador: HU.3	**Título**: Crear contenedores docker
Prioridad: 1	**Tiempo estimado**: 15 PH
Descripción: Como desarrollador, quiero mejorar el estado de la aplicación en el servidor en contenedores de nodejs, mysql y un administrador visual a la base de datos.	

Identificador: HU.4	**Título**: Desplegar los contenedores
Prioridad: 3	**Tiempo estimado**: 3 PH
Descripción: Como desarrollador, quiero tener la web y la base de datos desplegada correctamente en contenedores para individualizar los servicios y tener un mayor control.	

Identificador: HU.5	**Título**: Crear imitador de dispositivo de sonido
Prioridad: 1	**Tiempo estimado**: 15 PH
Descripción: Como desarrollador, quiero tener un sistema que me permita enviar datos falsos de número de serie a elección que simula diferentes limitadores para facilitar pruebas futuras.	

Identificador: HU.6	**Título**: Estadísticas de tipos de dispositivos recibidos
Prioridad: 1	**Tiempo estimado**: 2 PH
Descripción: Como desarrollador, quiero tener un control de los tipos de dispositivos recibidos y que datos fallan al entrar para poder mejorarlo en el futuro.	

Identificador: HU.7	**Título**: Organizar y documentar servicios servidor
Prioridad: 3	**Tiempo estimado**: 3 PH
Descripción: Cómo desarrollar, quiero tener una plantilla y explicación base de los servicios relacionados a la aplicación en el servidor, así como el manejo de redes de este (Nginx).	

Identificador: HU.8	**Título**: Ver lista de clientes
Prioridad: 2	**Tiempo estimado**: 3 PH
Descripción: El usuario puede ver una lista de clientes en una página diferente donde accedes al perfil de cada uno.	

Identificador: HU.9	**Título**: Ver lista de lugares
Prioridad: 2	**Tiempo estimado**: 1 PH
Descripción: El usuario puede ver una lista de lugares en una página diferente donde accedes al perfil de cada lugar (places).	

Identificador: HU.10	**Título**: Visibilidad de Gestion of users token
Prioridad: 3	**Tiempo estimado**: 1 PH
Descripción: El usuario puede ver con más claridad los ajustes de gestión de tokens de usuario para la app de móvil.	

Identificador: HU.11	**Título**: Modificación de la gráfica dispositivos
Prioridad: 2	**Tiempo estimado**: 3 PH
Descripción: El usuario puede desplazarse por la gráfica clicando en los laterales para desplazar las fechas mostradas.	

Identificador: HU.12	**Título**: Lista de descargas de archivos
Prioridad: 1	**Tiempo estimado**: 15 PH
Descripción: El usuario puede ver con más claridad los ajustes de gestión de tokens de usuario para la app de móvil.	

Identificador: HU.13	**Título**: Futuras mejoras web
Prioridad: 4	**Tiempo estimado**: – PH
Descripción: Futuras mejoras en la web como el cambio de idioma a español, o el modo oscuro.	

Nota: Se crea una última historia de usuario para encuadrarla en la distribución de tareas más adelante ya que existen varias tareas a futuro implementables con poco valor añadido como desarrollador.

5.4 Product backlog

El **Product Backlog** es una lista priorizada de funcionalidades, mejoras, correcciones y tareas necesarias para el desarrollo del proyecto. En este contexto, refleja las necesidades y objetivos que deben cumplirse para alcanzar los hitos del sistema de gestión de procesos. Este backlog incluye todas las actividades y requisitos que el equipo de desarrollo deberá abordar, organizados según su prioridad y valor para el usuario final. Es importante destacar que, dado que el proyecto está en constante evolución, este backlog no es estático; podrá modificarse, actualizarse o ampliarse conforme avance el desarrollo, ajustándose a nuevos requerimientos o cambios en las prioridades del proyecto.

A continuación, se presenta una tabla que muestra los elementos actuales del Product Backlog, organizados de acuerdo a su importancia y urgencia.

id	Título	Esf.	Prio.	Comentarios
1	Analizar estado del servidor y aplicación	3	1	
2	Documentar aplicación	2	2	
3	Crear contenedores docker	15	1	
4	Desplegar los contenedores	3	3	
5	Crear imitador de dispositivo de sonido	15	1	

6	Estadísticas de tipos de dispositivos recibidos	2	1	
7	Organizar y documentar servicios servidor	3	3	
8	Ver lista de clientes	3	2	
9	Ver lista de lugares	1	2	
10	Visibilidad de Gestion of users token	1	3	
11	Modificación de gráfica dispositivos	3	2	
12	Lista de descargas de archivos	15	1	

Tabla 2. Product Backlog. Creación propia

5.5 Cálculo de velocidad y distribución por etapas

Gestión del proyecto y capacidad del equipo

Este proyecto cuenta con un único programador que dedica 20 horas semanales y trabaja además en otras tareas. Gracias a la flexibilidad de **Kanban**, el trabajo se distribuye en tres **etapas clave**:

1. **Dockerización de la aplicación**
2. **Desarrollo del programa Arduino**
3. **Mejoras en la aplicación web**

Cada etapa se gestiona mediante un tablero Kanban con las columnas: Pendiente, En Proceso, Revisión, y Completado. Este flujo continuo permite que las tareas se ajusten según las prioridades en tiempo real, sin necesidad de sprints fijos.

Distribución del esfuerzo
Las tareas se estiman en horas ideales de programación, pero el tiempo real es mayor debido al aprendizaje adicional y otras obligaciones, lo que multiplica las horas ideales por cuatro.

Adaptabilidad del enfoque Kanban

Al no depender de sprints, el proyecto avanza en función de las historias de usuario de cada etapa. Se utilizarán tres tableros Kanban, uno para cada etapa: Dockerización de la aplicación, Desarrollo del programa Arduino, y Mejoras en la aplicación web. Este enfoque Kanban facilita la priorización y permite modificaciones continuas en cada tablero según surjan cambios, mejorando la respuesta ante imprevistos y permitiendo que cada tarea se complete a su propio ritmo, adaptándose a las necesidades del proyecto y al progreso de las historias de usuario en cada etapa.

5.5.1 Etapas

Como se ha indicado previamente, el desarrollo del proyecto se estructura en tres fases clave. A continuación, se presentará una descripción detallada de cómo se distribuyen las diferentes historias de usuario entre estas fases, lo que permite organizar de manera más eficiente el flujo de trabajo y asegurar que cada una de las etapas reciba el enfoque necesario para su ejecución.

Etapa 1: Dockerización	
HU: 3	Crear contenedores docker
HU: 4	Desplegar los contenedores
HU: 1	Analizar estado del servidor y aplicación

Etapa 2: Arduino	
HU: 5	Crear imitador de dispositivo de sonido
HU: 6	Estadísticas de tipos de dispositivos recibidos
HU: 7	Organizar y documentar servicios servidor

Etapa 3: Aplicación	
HU: 2	Documentar aplicación
HU: 8	Ver lista de clientes
HU: 9	Ver lista de lugares
HU: 10	Visibilidad de Gestion of users token
HU: 11	Modificación gráfica dispositivos
HU: 9	Lista de descargas de archivos
HU: 13	Futuras mejoras

Tabla 3. Etapas de desarrollo. Creación propia.

5.6 Presupuesto

A continuación, se presenta el desglose detallado del presupuesto del proyecto, teniendo en cuenta diversos factores clave. Es fundamental considerar los **componentes de hardware** necesarios para la implementación del sistema, así como el **software** utilizado que requiera licencias para su uso, lo que generará un coste adicional. Además, el presupuesto incluye el coste de los **recursos humanos** involucrados en el desarrollo del proyecto, en función de las **horas dedicadas del trabajador**, ya que recordemos que este proyecto solo cuenta con un único desarrollador. También se deben tener en cuenta los **gastos indirectos**, como los relacionados con el consumo de energía (luz) y los gastos de mantenimiento, que son esenciales para garantizar la operatividad del sistema a lo largo del tiempo. Este desglose permite una visión completa y detallada de los recursos necesarios para la ejecución del proyecto.

- ***Trabajo del desarrollador***:
  - **Horas totales**: calculamos el valor de cada historia en 4 horas reales, por lo que teniendo 66 puntos de historias de usuario.
    - Horas totales: **264.**
  - **Tarifa actual**: 20€/h
  - **Coste total**: 5280€

Componente	Precio	Unid.	Total
Servidor GranaSat	6€/mes	4 meses	24€
Portátil Medion MD62621	679€	1	679€
Arduino Uno	30€	1	30€
Shield ethernet - shield hanrun hr91105a	30€	1	30€
Mantenimiento dispositivos (portátil y arduinos)	100€	Anual	100€
Software: Licencia windows	140€	1	140€
SSL, dominios…	Se mantienen los anteriormente usados		0
electricidad, internet y otros gastos	40€	4 meses	160€
Coste desarrollador	20€	264 horas	5280€
Total:			**6443€**

Tabla 4. Presupuesto del proyecto. Creación propia

5.7 Recurso software a usar

En este proyecto, se emplea la metodología Kanban a través de una de las aplicaciones web más populares para este ámbito, Trello. Primero, añadiendo las historias de usuario conforme se vaya avanzando en el tablero de Trello, organizando las tareas de acuerdo con las tres distintas etapas del proyecto. Esto facilitará una gestión continua y visual del flujo de trabajo, permitiendo una mayor flexibilidad y adaptación a cambios. Además, Trello proporciona una visión clara del progreso y el estado de cada tarea, optimizando así la eficiencia en el desarrollo.

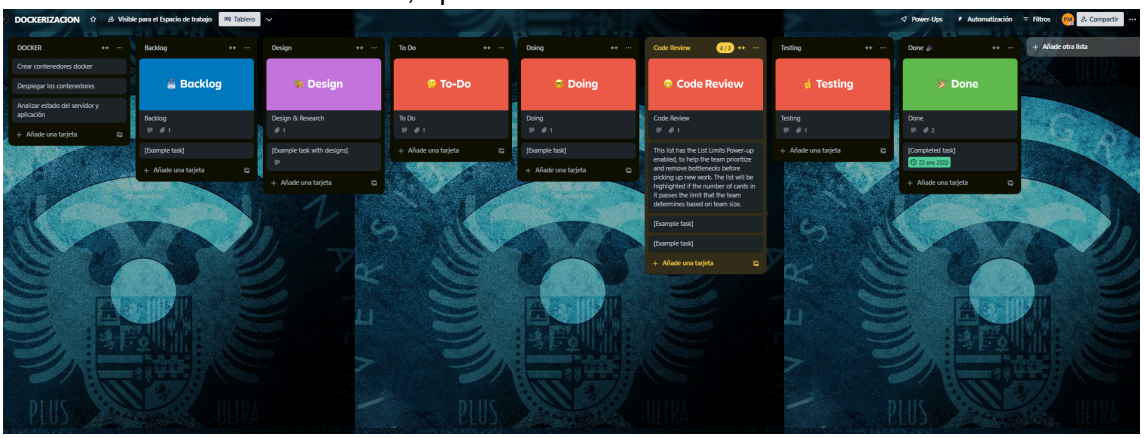


Figura 8. Tablón Kanban en la aplicación trello. [3].

6. Diseño

El diseño y la selección de tecnologías son elementos fundamentales en el desarrollo de este proyecto, ya que impactan directamente en la funcionalidad, eficiencia, escalabilidad y mantenimiento del sistema, así como en su capacidad para adaptarse a futuros requerimientos. Dado que el proyecto parte de una base preexistente, se han tomado decisiones estratégicas para mantener algunas de las tecnologías utilizadas anteriormente, como Node.js para el backend y MySQL para la base de datos, asegurando así la compatibilidad y aprovechando el trabajo previo. Este capítulo aborda las decisiones de diseño y las tecnologías empleadas, proporcionando una visión detallada de los criterios seleccionados y las soluciones implementadas.

La aplicación está compuesta por un programa en Node.js, que actúa como el backend, y una interfaz desarrollada en Angular, que forma el frontend de la aplicación. Angular se comunica con el backend mediante una API REST local, alojada en el mismo proyecto, lo cual permite una integración fluida y facilita el intercambio de datos en tiempo real. Esta arquitectura permite una separación clara entre la lógica de negocio y la presentación, garantizando así un desarrollo modular y facilitando futuras actualizaciones y mejoras en cada una de las partes del sistema sin afectar a la otra.

6.1 Diseño arquitectura docker

La primera etapa del proyecto se centra en mejorar la infraestructura dockerizando la aplicación en el servidor. Para esto, se ha diseñado una arquitectura **docker-compose** que utiliza tres contenedores: el primero para la aplicación principal, ***HeimdalSoundControl***, que alberga tanto el backend en **Node.js** como el frontend en **Angular**; el segundo contenedor para la base de datos, que se migrará de un servicio independiente en el servidor a un contenedor Docker para una mayor portabilidad y facilidad de gestión; y el tercer contenedor para **phpMyAdmin**, una herramienta de administración de bases de datos **MySQL** accesible a través de una interfaz web. phpMyAdmin permite a los desarrolladores gestionar la base de datos con una interfaz gráfica sencilla y eficiente, ideal para operaciones complejas y mantenimiento de datos. Sin embargo, dado que phpMyAdmin es solo para uso de desarrolladores, se ha implementado una restricción de acceso mediante IP para protegerlo y asegurar que solo las direcciones autorizadas puedan acceder a esta herramienta.

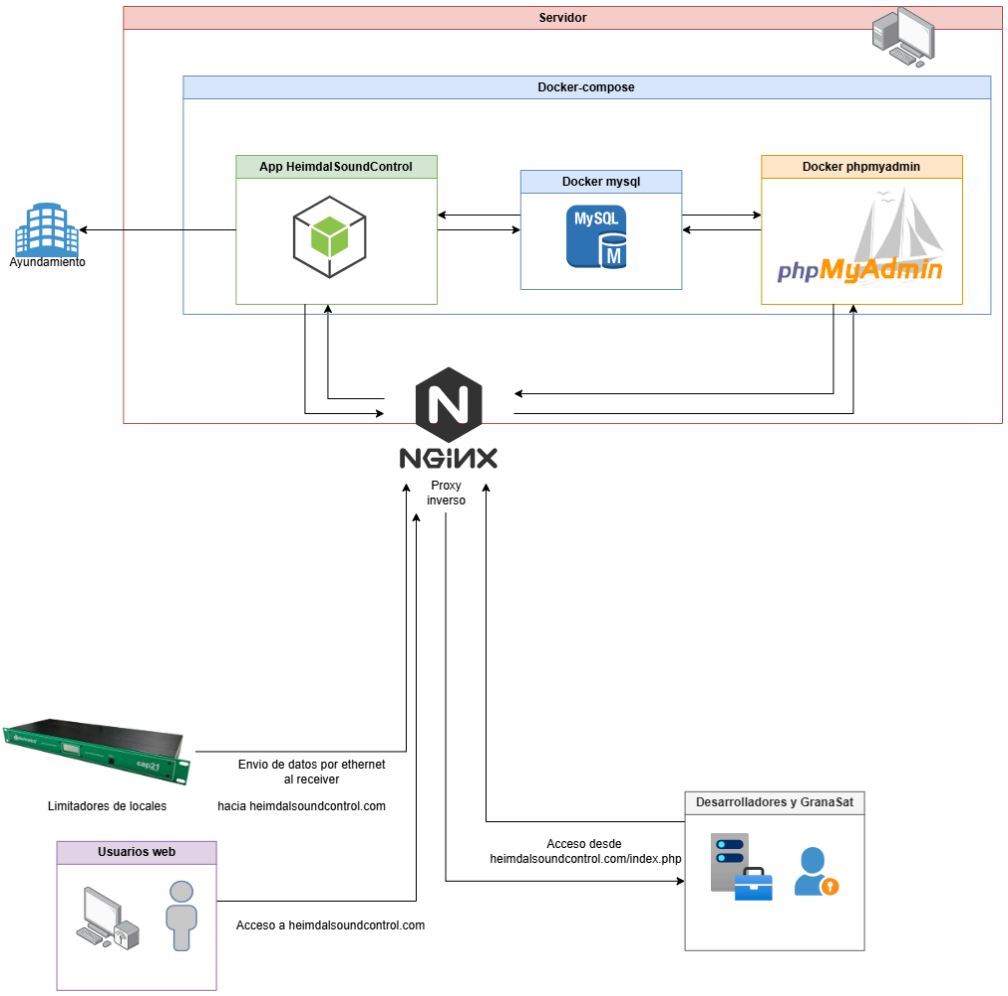


Figura 9. Arquitectura de la aplicación en el servidor. [6].

Razones para el cambio

Anteriormente, el sistema usaba PM2 para lanzar la aplicación Node.js, lo cual permitía una ejecución continua, pero no ofrecía la **portabilidad** ni el **aislamiento** de Docker. Además:

- **Base de datos MySQL**: Configurada como un servicio global en el puerto 3306, exponiendo riesgos de acceso no autorizado y dificultando la configuración de seguridad.
- **Sin interfaz gráfica**: La administración de la base de datos no contaba con una herramienta gráfica, limitando la eficiencia en la gestión y consultas.

Ventajas de Docker
Docker permitió reestructurar el proyecto de forma escalable y segura mediante contenedores, cada uno aislando una parte específica del sistema. Con Docker Compose, se establecieron **tres contenedores**:

1. **Aplicación (heimdalsoundcontrol)**
2. **Base de datos (MySQL)**
3. **phpMyAdmin**: Añadido para gestionar la base de datos de manera gráfica y

más eficiente.

Beneficios principales:

- **Portabilidad**: Facilita la replicación del entorno de producción en desarrollo, reduciendo problemas de configuración y dependencias.
- **Seguridad mejorada**: Docker Networking permite limitar el acceso a MySQL, y solo desarrolladores asignados pueden acceder a phpMyAdmin mediante IPs autorizadas.
- **Control de versiones**: Simplifica la implementación y actualización, permitiendo un control más preciso sobre cada componente de la aplicación.

Este cambio optimiza tanto la **organización** como la **escalabilidad** del sistema, mejorando la productividad del equipo y manteniendo un entorno de ejecución uniforme y seguro.

6.1.1 Puntos claves a implementar

Acceso y configuración de los servicios a través de Nginx y Docker

El acceso a la aplicación y a los servicios internos del sistema está gestionado por un **servidor inverso Nginx**, como se muestra en la **Figura 9**. Nginx redirige las solicitudes de **heimdalsoundcontrol.com** hacia los contenedores Docker internos, garantizando una **comunicación segura**.

1. **Aplicación principal**: El tráfico desde **heimdalsoundcontrol.com** se redirige a la IP interna asignada al contenedor de la aplicación.

2. **phpMyAdmin**: Accesible en **heimdalsoundcontrol.com/index.php**, se redirige a través de Nginx hacia su IP interna específica.

3. **Base de datos MySQL**: Totalmente aislada en una **red interna** de Docker, accesible solo desde otros contenedores, lo que **refuerza la seguridad**.

Configuración de contenedores Docker
Cada componente tiene configuraciones específicas para optimizar el desarrollo y despliegue:

- **Contenedor Node.js**: Se configura con un **volumen** apuntando al directorio del código fuente, lo que permite la **actualización en tiempo real** sin necesidad de reconstruir el contenedor. También tiene acceso directo al contenedor MySQL a través de la red interna Docker.

- **Contenedor MySQL**: Configurado con las credenciales necesarias, y un **volumen** asociado a una carpeta en el servidor para almacenar las **copias de seguridad**. Esto permite restaurar la base de datos rápidamente sin afectar la producción.

- **Contenedor phpMyAdmin**: Accede solo al contenedor MySQL y está restringido por una lista de **IP's permitidas** mediante la configuración de Nginx. Esto asegura que solo los desarrolladores autorizados puedan acceder a la interfaz gráfica de la base de datos.

Esta configuración de contenedores y el uso de Nginx asegura **aislamiento**, **seguridad** y **eficiencia** en la administración y acceso de los servicios del sistema.

6.2 Simulador limitador de sonido

6.2.1 Arquitectura envio limitador a Heimdal

La arquitectura del sistema para el envío de datos acústicos desde el limitador al servidor sigue una lógica de comunicación sencilla y robusta, diseñada para transmitir información de manera fiable y eficiente. En este flujo de trabajo, el Arduino recibe los datos del limitador de sonido a través de un puerto específico. Estos datos representan los niveles de sonido que el dispositivo está monitoreando y se almacenan temporalmente en el Arduino antes de ser enviados al servidor de Heimdal.

El Arduino está configurado para enviar los datos acústicos a través de una conexión Ethernet, utilizando una dirección IP específica del servidor Heimdal y un puerto de red predeterminado. Esta comunicación es gestionada por una serie de comandos programados en el Arduino que se encargan de establecer la conexión, transmitir los datos y verificar que el envío se haya realizado con éxito. Adicionalmente, el código incluye un sistema de respaldo que permite a los datos ser enviados también a un segundo servidor, en este caso de un ayuntamiento, en caso de que sea necesario.

El enfoque de esta arquitectura garantiza que los datos puedan ser enviados a múltiples destinos en paralelo, lo que permite que tanto el servidor de Heimdal como el servidor municipal reciban la información relevante de forma casi simultánea. Esta configuración no solo asegura una redundancia en la transmisión de datos, sino que también facilita el monitoreo en tiempo real y permite que múltiples instancias gestionen los mismos datos. Además, gracias al uso de un sistema basado en Ethernet y DHCP, el dispositivo Arduino puede operar en redes cambiantes, gestionando las conexiones de red de forma automática y adaptándose a los entornos en los que se despliegue.

Descripción:

- **Arduino:** El Arduino actúa como el emisor de datos, conectado al limitador de sonido y a la red Ethernet.
- **Limitador de Sonido:** Este componente envía datos de nivel de ruido al Arduino.
- **Servidor Heimdal:** Es el servidor principal al que el Arduino envía los datos acústicos. Este servidor gestiona y procesa los datos recibidos.
- **Servidor del Ayuntamiento:** Es un servidor adicional para los casos en que también se requiere enviar datos al ayuntamiento.
- **Red Ethernet:** Representa la red Ethernet con acceso a la red a la que el Arduino está conectado y a través de la cual envía datos. El servidor DHCP también se puede representar para indicar que la IP es asignada de forma automática.

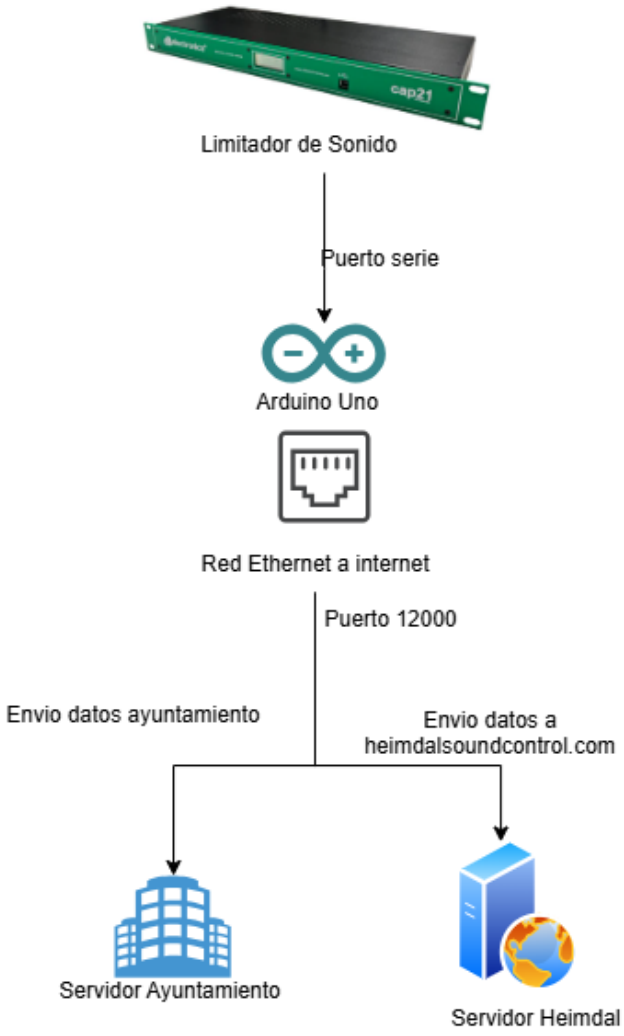


Figura 10. Arquitectura de envio paquetes limitadores de sonido. [6].

6.2.2 Arquitectura del simulador de envio paquetes de sonido

Para la arquitectura del simulador de envío de paquetes de sonido, se emplean un **Arduino Uno** [9] y un **shield Ethernet HR911105A** [9], ambos dispositivos con características esenciales para los requisitos del proyecto. El Arduino Uno ofrece una plataforma de microcontrolador eficiente y económica, ideal para manejar el procesamiento y envío de datos de sonido simulado de manera básica. Además, el uso del shield Ethernet HR911105A permite establecer una conexión de red confiable y directa, esencial para enviar datos hacia el servidor de forma continua.

Ambos dispositivos juntos logran un balance adecuado entre funcionalidad y costo, siendo muy efectivos para esta implementación de simulación de datos de ruido. La conexión por puerto serie al ordenador facilita la supervisión y depuración durante el desarrollo, mientras que el acceso Ethernet proporciona una comunicación rápida y en tiempo real con el servidor de destino.

6.1.3 Objetivo y funcionalidad

En esta parte del proyecto, el objetivo es configurar el Arduino para que actúe como un generador de datos ficticios, replicando los valores que normalmente transmite un limitador de sonido instalado en un local. Este simulador se desarrollará para emular las características y comportamiento de los limitadores **CAP21 tipo X** [7], dispositivos específicos ya en uso en el sistema. Así, se busca que el Arduino envíe datos que sigan el mismo formato exacto, lo cual incluye encapsular la información en una cadena de bytes con la misma estructura que los datos enviados por los limitadores reales.

En el **diagrama de la Figura 11**, se muestra el flujo de trabajo entre el **Arduino** y el **servidor** en un **entorno de pruebas**. Este esquema simula cómo se establecería la comunicación en un entorno real de monitoreo acústico, pero sin involucrar a un destinatario final real como un ayuntamiento.

1. **Conexión Arduino-PC**: El Arduino está conectado al PC, donde genera **paquetes de datos** que simulan los **niveles de ruido** y otros parámetros.

2. **Transmisión de datos**: A través de la conexión **Ethernet** del portátil, los datos son enviados al servidor. Los datos simulados siguen un formato estructurado, que es crucial para que el servidor los **interprete correctamente**, tal como lo haría con dispositivos reales.

3. **Propósito de la simulación**: Este entorno de prueba crea un **escenario controlado** que valida la recepción y procesamiento de datos, utilizando información ficticia pero representativa de un entorno real. Así, se **omite** el envío al ayuntamiento para centrarse en la validación de datos sin requerir hardware adicional.

4. **Simulación de múltiples dispositivos**: El **Arduino** simula envíos de datos de **múltiples limitadores**, como si fuera **varios dispositivos CAP21 tipo X**. Esto permite probar la **capacidad de recepción** del servidor y la **gestión de datos** en tiempo real, asegurando que el sistema pueda manejar múltiples entradas simultáneas sin problemas.

Este enfoque permite **probar la infraestructura completa** de la aplicación y asegurarse de que la **recepción, procesamiento** y **estabilidad de los datos** funcionen correctamente, sin depender de dispositivos físicos reales en la fase de pruebas.

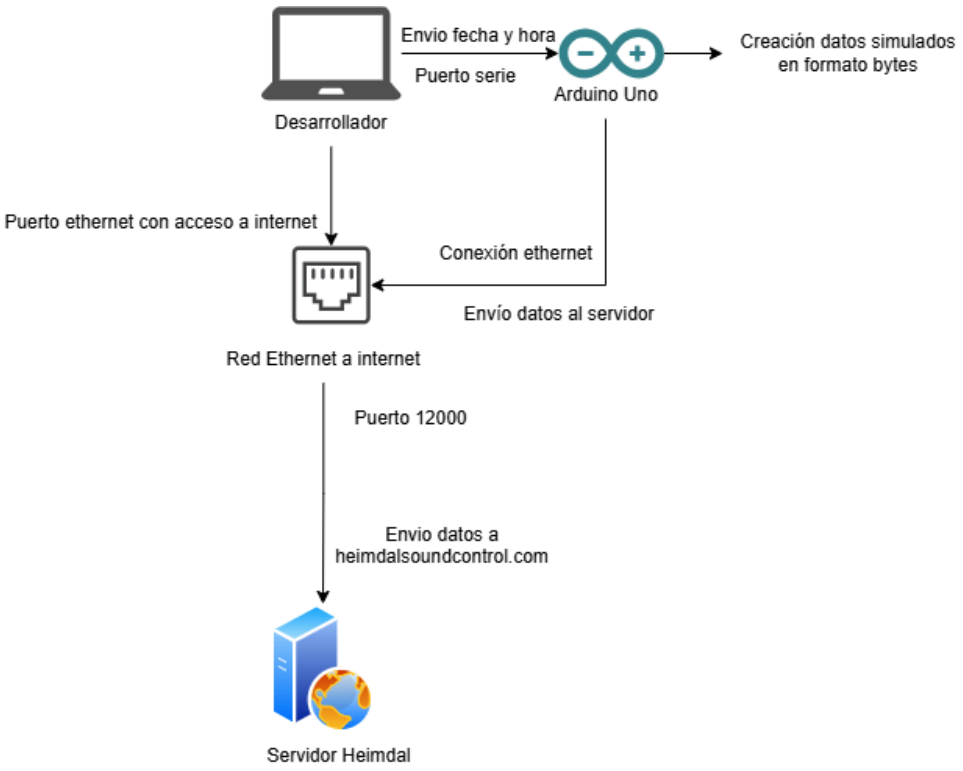


Figura 11. Arquitectura de simulador del limitador de sonido. [6].

6.1.4 Control de Tiempo en Arduino

Para gestionar la sincronización temporal en el sistema de monitoreo acústico, se plantean tres versiones de control de tiempo en Arduino, cada una incrementando la precisión y autonomía en el manejo de la fecha y hora. Estas versiones permitirán, en distintas fases, probar la funcionalidad y estabilidad del dispositivo sin depender siempre de una conexión constante a la red, al tiempo que ajustarán automáticamente la hora en función de las necesidades del proyecto.

La **primera versión** establece un método manual, en el cual se ingresa la fecha y hora inicial en el Arduino. Este valor es almacenado en la memoria EEPROM [9] del Arduino y utiliza el contador de tiempo interno para actualizar progresivamente la hora, incluso después de un reinicio del sistema. Esta versión básica permite un control de tiempo inicial sin la necesidad de componentes adicionales, aunque puede acumular desfases con el tiempo debido a la imprecisión del reloj interno del Arduino.

La **segunda versión** mejora la precisión del control de tiempo mediante el acceso por TCP a un servidor horario europeo, desde el cual se obtiene la fecha y hora actual. El Arduino se conecta a este servidor en intervalos determinados para actualizar la hora y luego ajustarla al horario español, lo que garantiza una mayor exactitud y permite una corrección periódica de la hora sin depender exclusivamente del hardware del dispositivo.

Finalmente, la **tercera versión** utiliza un módulo de reloj en tiempo real (RTC) [9] para mantener la sincronización. Este módulo proporciona una solución autónoma de control

de tiempo, con una alta precisión y sin necesidad de depender de la conexión a un servidor. Además, el RTC mantiene la hora correcta durante los reinicios del sistema, siendo ideal para un funcionamiento continuo en el entorno de pruebas de monitoreo acústico.

6.2 Diseño arquitectura HeimdalSoundControl

La arquitectura de **HeimdalSoundControl** está estructurada sobre un modelo cliente-servidor, con un frontend desarrollado en Angular y un backend en Node.js. El frontend se encarga de la interacción con el usuario, mientras que el backend gestiona la lógica de negocio, la comunicación con la base de datos y los dispositivos de medición acústica. El sistema utiliza contenedores Docker para la gestión de servicios y bases de datos, lo que facilita el despliegue y la escalabilidad del sistema. Esta arquitectura modular permite incorporar nuevas funcionalidades y realizar mejoras sin afectar al rendimiento ni a la estabilidad general de la aplicación.

6.2.1 Arquitectura General

Como se detalló previamente en el punto **3. Estado inicial del proyecto**, la aplicación **HeimdalSoundControl** sigue un modelo **cliente-servidor**. En este esquema, el **frontend** está desarrollado en **Angular** y el **backend** utiliza **Node.js**. La comunicación entre ambos se realiza a través de una **API REST local**, empleando el protocolo **HTTP** para el intercambio de datos en formato **JSON**.

La API **local** se ejecuta en el mismo servidor que el **frontend**, lo que facilita la gestión de la lógica de negocio y las interacciones con la **base de datos**. Dado que tanto la **API** como los accesos a la base de datos están en el mismo entorno de servidor, la eficiencia en la comunicación entre estos componentes está optimizada.

La **base de datos** utilizada es **MySQL**, encargada de almacenar los datos clave del sistema, tales como los dispositivos de medición acústica, las alertas generadas por el sistema y la información de los usuarios. Las **consultas SQL** se gestionan de manera eficiente y están optimizadas según las necesidades específicas de la aplicación.

Mantenimiento de la estructura:

La **arquitectura general** no sufrirá cambios significativos, pero algunas optimizaciones y mejoras se implementarán en áreas clave. En primer lugar, se mantendrá el uso del modelo **cliente-servidor** con el **frontend** en Angular y el **backend** en Node.js, pero se añadirán diferentes vistas en el frontend, así como nuevos accesos a la base de datos y nuevas peticiones api. Además, aunque la base de datos se mantendrá como **MySQL**, manteniendo las mismas tablas evitando en mayor medida su modificación para asegurar que el rendimiento sea cercano al actual, especialmente debido a que se gestionan grandes volúmenes de datos, pero pudiendo añadir tablas en función de las necesidades.

6.2.2 Diseño base de datos

La **estructura y diseño de la base de datos** de **HeimdalSoundControl** se mantendrán mayormente igual, ya que se utilizará la base de datos actual, restaurada desde una copia de seguridad, para garantizar la integridad de los datos existentes. Durante la implementación de la aplicación, se introducirán dos nuevas tablas para ampliar las funcionalidades de la aplicación.

La primera tabla, **deviceTypes**, almacenará información sobre los tipos de dispositivos disponibles en el sistema, permitiendo clasificar y gestionar distintos modelos o categorías de dispositivos de medición acústica. Esta tabla incluye atributos como el nombre del dispositivo, su descripción y otros detalles relevantes.

La segunda tabla, **deviceDownloads**, estará asociada a los dispositivos y almacenará los archivos relacionados con ellos, como manuales, configuraciones o actualizaciones. Estos ficheros estarán disponibles en la pestaña de **Downloads** de la aplicación, donde los usuarios podrán consultar y descargar los archivos correspondientes a cada dispositivo. Ambas tablas serán utilizadas exclusivamente en esa sección de la aplicación, y su implementación contribuirá a una mejor gestión de los recursos y la información relacionada con los dispositivos.

6.2.3 Diseño visual

Como muestra la figura 12, el **diseño visual** de **HeimdalSoundControl** se mantendrá alineado con la estética actual de la aplicación, ya que el objetivo principal es mejorar y optimizar las funcionalidades sin afectar su apariencia visual. No se estipulará un nuevo seguimiento de diseño, por lo que la paleta de colores, formato de letra y estilo general se mantendrán igual, utilizando el esquema actual basado en un fondo blanco con bordes en tonos **azules** o **negros**, según el momento.

A nivel visual, se realizará una intervención mínima para preservar la estética de la aplicación. Los cambios visuales serán puntuales y se mostrarán durante la implementación a medida que se produzcan, sin alterar el diseño general ni comprometer la experiencia del usuario. De esta manera, se garantizará que las mejoras funcionales no interfieran con el diseño visual ya establecido, manteniendo la coherencia y familiaridad de la interfaz.

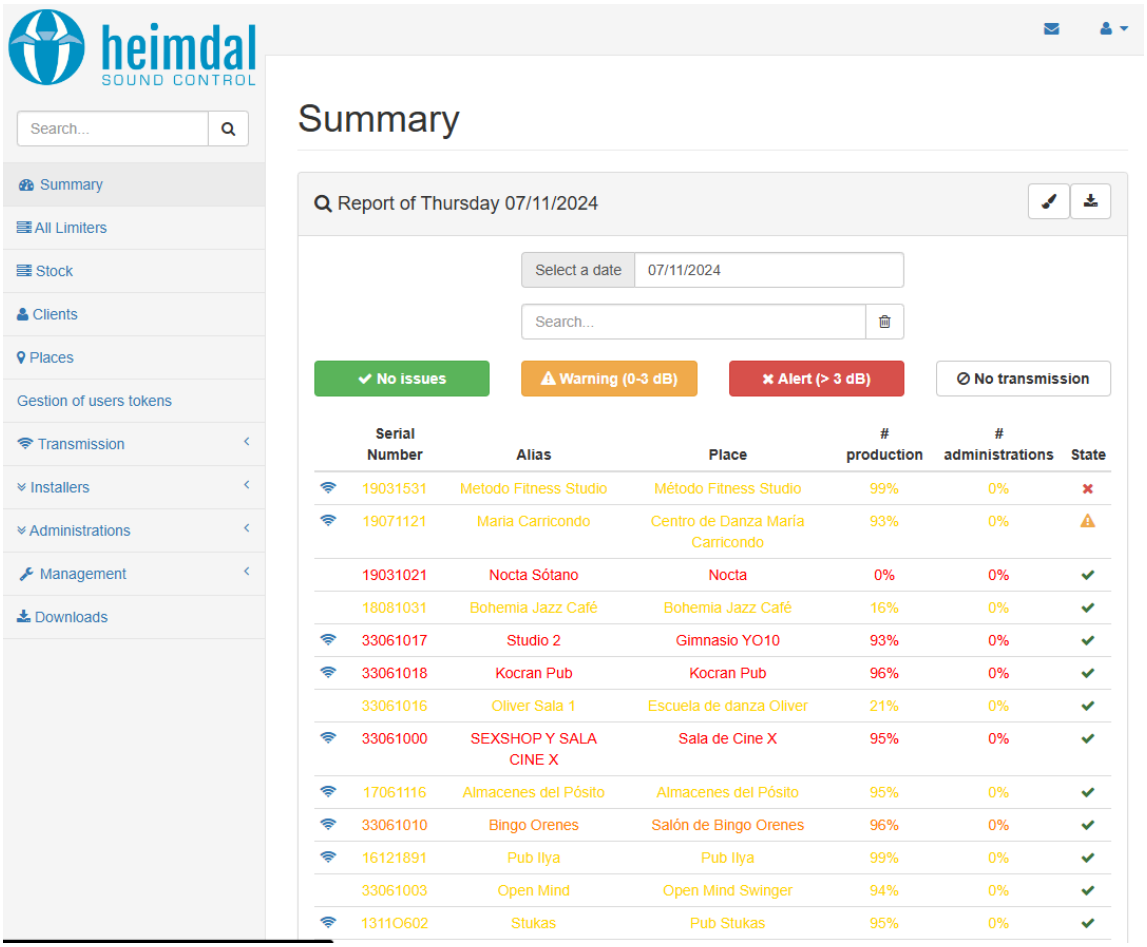


Figura 12. Diseño visual de HeimdalSoundControl con información privada. [10].

6.2.6 Tecnologías Usadas

En **HeimdalSoundControl**, se ha optado por seguir la base tecnológica ya planteada, manteniendo las **mismas tecnologías** que se utilizaban en el sistema original: **Node.js**, **Angular** y **componentes de Bower.js**. Esta decisión responde a la necesidad de evitar problemas y errores derivados de la sustitución de componentes por otros nuevos que aún no han sido probados ni validados en las pruebas anteriores del sistema. Al conservar las tecnologías previas, se asegura la estabilidad del sistema y se minimizan los riesgos asociados a cambios innecesarios en la infraestructura tecnológica.

- **Nodejs:** Es una elección adecuada debido a su naturaleza asíncrona y de alto rendimiento, lo que permite manejar múltiples conexiones simultáneas de manera eficiente, algo crucial en una aplicación que interactúa en tiempo real con dispositivos y bases de datos. Además, la capacidad de **Node.js** para gestionar operaciones de entrada/salida de manera no bloqueante mejora la escalabilidad de la aplicación y permite un procesamiento rápido de datos, lo que es ideal para un sistema de monitoreo acústico.

- Por otro lado, **Angular** se ha mantenido como el framework para el **frontend** debido a su robustez y capacidad para crear aplicaciones web dinámicas y

eficientes. Su enfoque basado en componentes facilita el desarrollo modular y la reutilización de código, lo que simplifica la implementación de nuevas funcionalidades y el mantenimiento a largo plazo. La integración de **Angular** con **Node.js** a través de **APIs REST** permite una comunicación fluida entre el frontend y el backend, optimizando la experiencia del usuario y garantizando un rendimiento excelente en la visualización de datos en tiempo real.

6.2.6 Despliegue y Entorno de Producción

El **despliegue inicial** del proyecto comienza en un entorno local utilizando **contenedores Docker** en mi ordenador. Se parte de una **copia de seguridad** actualizada de la base de datos, que se carga en el entorno Docker para establecer la estructura necesaria para la aplicación. Durante el desarrollo, este entorno local se mantiene con los **datos previos** y permite la **recepción de datos en tiempo real** desde un **simulador de Arduino**, que envía información acústica falsa para pruebas.

En el **entorno de producción**, se replicará la configuración del entorno local en el servidor de **GranaSat**, bajo una copia del proyecto llamada **HEIMDALCOPY**. La gestión de versiones se hará a través de **GitLab**. Esta transición asegurará que la aplicación en producción mantenga la **configuración**, **datos** y **estructura** probados localmente, garantizando **estabilidad** y **consistencia** en el despliegue.

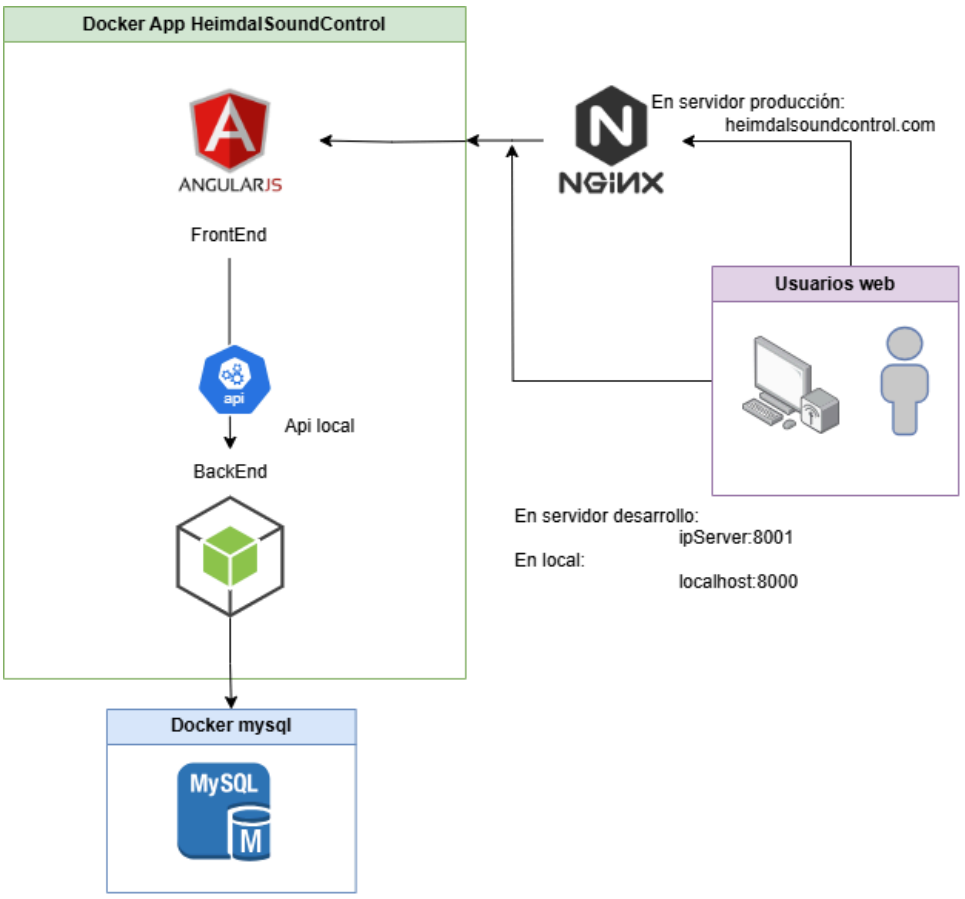


Figura 13. Diferencias básicas local y servidor. Creación propia.

En la **Figura 13** se muestra la configuración de acceso a la aplicación en dos entornos: el **entorno de pruebas** accesible por **IP** y el **entorno de producción** accesible a través de **Nginx** en **heimdalsoundcontrol.com**. Este enfoque es **sencillo** y **económico**, ya que no se utiliza un dominio para el entorno de pruebas y el servidor está poco estructurado, lo que hace difícil su configuración adecuada.

Gestión de Servicios Web

Para permitir que ambos servicios (el original y el de pruebas) operen simultáneamente sin conflictos, se ajustan los puertos en el archivo **.env** del **docker-compose**. Así, los datos de los limitadores se envían solo al servicio original activo.

Configuración de Nginx

Como no se ha configurado Nginx para redirigir la URL **heimdalsoundcontrol.com** hacia la instancia de pruebas, esta URL sigue apuntando exclusivamente al servicio web original, sin interferencias por parte de la copia de prueba.

6.2.6 Pruebas y documentación

En cuanto a **pruebas y documentación** del diseño de la aplicación, las pruebas se plantean como simples y rápidas, ejecutándose inicialmente en local y realizando verificaciones directamente desde la misma web en el entorno de desarrollo. Una vez superadas estas pruebas, el sistema se despliega en el **servidor de copia** para evaluar su estabilidad en un entorno más robusto y así asegurar el soporte adecuado antes de cualquier implementación en producción.

Para la **documentación** del proyecto, se ha optado por utilizar este mismo informe de **Trabajo de Fin de Grado (TFG)** como guía principal, que servirá de referencia para futuros desarrolladores (ver Apartado 3). Además, se ha documentado el código de manera exhaustiva y se han creado otros documentos de apoyo en **notebooks de Jupyter**, proporcionando ejemplos prácticos y explicaciones adicionales sobre el funcionamiento y las configuraciones del sistema.

7. Implementación

Este capítulo describe el proceso de **implementación** de la aplicación, donde los diseños y especificaciones se transforman en un producto funcional. Se cubren todos los aspectos técnicos y prácticos, desde la configuración del entorno de desarrollo hasta las pruebas y la integración final.

La implementación se divide en tres etapas principales: **Docker**, **Arduino** y **Aplicación Web**, cada una gestionada con su propio **tablero Kanban** para un flujo de trabajo continuo y flexible, permitiendo añadir historias de usuario según surjan nuevas necesidades.

Cada etapa incluye:

- **Docker**: Configuración y optimización de contenedores para el backend (Node.js), frontend (Angular) y base de datos, adaptando el **docker-compose** a las necesidades del entorno de desarrollo y pruebas.
- **Arduino**: Configuración del simulador de limitador de ruido, integrando el envío de datos vía Ethernet y asegurando la correcta comunicación con el servidor.
- **Aplicación Web**: Implementación de nuevas funcionalidades en el frontend (Angular), como vistas, formularios y pantallas de monitoreo acústico, junto con las actualizaciones del backend para soportar las nuevas características.

El proceso es continuo y en paralelo, con la evolución dinámica de las etapas según las necesidades de desarrollo.

7.0 Etapa 0: Aprendizaje y planificación

La "Etapa 0" representa una fase de análisis y planificación estratégica inicial. En este paso previo, se abordaron varias actividades fundamentales para preparar la implementación del proyecto:

1. **Investigación del servidor y análisis del proyecto existente**: Se realizó una revisión detallada del servidor, de la estructura de archivos y del código base del proyecto. Este análisis permitió identificar las configuraciones iniciales, las tecnologías ya implementadas, y el estado general del sistema. Este estudio es esencial para entender cómo se gestionan los datos y el flujo de trabajo actual, así como para detectar posibles áreas de mejora o errores en la estructura.
2. **Documentación preliminar**: Con el fin de mantener un registro adecuado para futuras referencias, se comenzó una documentación de los descubrimientos iniciales y del entorno de trabajo del proyecto. Esto incluyó la organización de archivos, la infraestructura del servidor y detalles sobre configuraciones clave. Esta documentación inicial servirá de guía para el desarrollo y para los futuros mantenedores del sistema.
3. **Búsqueda y evaluación de tecnologías**: Como parte de los requisitos de actualización y modernización, se evaluaron diferentes tecnologías que podrían

integrarse al proyecto, comparando opciones y seleccionando aquellas que mejor se adapten a las necesidades del sistema y del entorno.

4. **Diseño de diagramas**: Para obtener una representación clara y visual del sistema, se desarrollaron diagramas preliminares, como el de la arquitectura del sistema, los flujos de datos, y el uso de la interfaz, entre otros. Estos diagramas sirven como base para el desarrollo de las etapas futuras, asegurando una comprensión visual del funcionamiento global del proyecto.

7.1 Etapa 1: Docker

La **Etapa 1** se centra en la transición de la infraestructura a contenedores Docker, buscando mejorar la portabilidad y escalabilidad del sistema. Este cambio implica reemplazar el servicio PM2, previamente utilizado para gestionar procesos de la aplicación, por contenedores Docker, lo que permite una mejor gestión y despliegue. Asimismo, en esta etapa se aísla la base de datos MySQL en su propio contenedor Docker, garantizando un entorno controlado y optimizado para el almacenamiento de datos. Este proceso de containerización supone un primer paso hacia una arquitectura modular que facilita la administración y el desarrollo continuo del sistema.

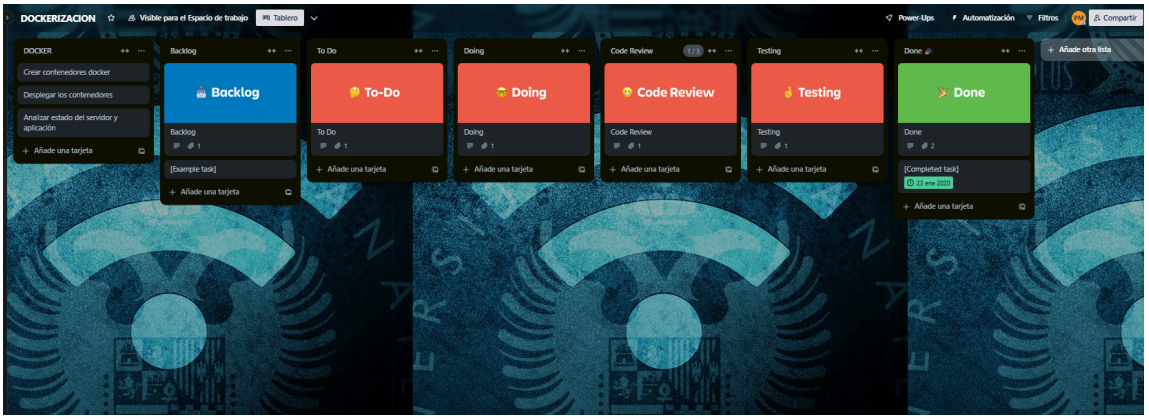


Figura 14. Tablón Kanban inicial etapa 1. Creación propia con trello.

Partiendo de las tres historias de usuario básicas establecidas en la planificación inicial, esta etapa requiere el desarrollo de **historias más específicas**, guiadas por los detalles identificados en la fase de diseño. Estas nuevas historias tendrán como objetivo la creación de tres contenedores Docker, siguiendo el formato previamente diseñado. Dado que cada contenedor cumple una función clave en la arquitectura general, estas historias de usuario no tendrán una prioridad diferenciada; todas deberán completarse para alcanzar el objetivo de esta etapa. Además, las historias estarán orientadas a anticipar y resolver posibles errores que puedan surgir durante el desarrollo, asegurando un proceso de implementación más robusto y controlado.

7.1.1 Tareas a realizar

Historia de usuario	Tareas
Crear archivo docker-compose.yml con tres servicios	1. Definir el contenedor de Node.js para alojar la aplicación. 2. Configurar el contenedor de MySQL con un volumen persistente para datos. 3. Configurar el contenedor de phpMyAdmin para conectarse solo al contenedor de MySQL.
Configurar archivo .env para variables de entorno:	4. Crear un archivo .env con todas las variables necesarias (credenciales de MySQL, puertos de conexión, configuración de phpMyAdmin). 5. Actualizar docker-compose.yml para utilizar variables desde el archivo .env.
Importar la base de datos de MySQL desde el servidor	6. Acceder al servidor para extraer una copia de seguridad de la base de datos. 7. Diseñar un proceso para importar esta copia dentro del contenedor MySQL en Docker.
Automatizar la importación de la base de datos en Docker	8. Automatizar la importación de la base de datos en Docker
Gestionar posibles errores de arranque de contenedores	9. Verificar dependencias de arranque entre contenedores (e.g., MySQL debe estar activo para que phpMyAdmin funcione correctamente).
Revisar la conexión entre los contenedores y pruebas de red.	10. Comprobar la conexión entre Node.js y MySQL, y entre MySQL y phpMyAdmin. 11. Asegurar que la aplicación Node.js puede acceder a MySQL utilizando las variables de entorno del archivo .env.
Pruebas de funcionamiento	12. Ejecutar pruebas básicas de funcionalidad para confirmar que cada contenedor responde como se espera

Tabla 4. Tabla tareas etapa 1. [8].

Voy a trasladar estas tareas detalladas al tablero Kanban, donde se gestionan de manera visual y ordenada. Esto permitirá hacer un seguimiento claro del progreso en cada tarea y facilitará la identificación de bloqueos o prioridades en el proceso de implementación con Docker.

Las tareas se abordarán prácticamente en el orden definido, comenzando con la creación de los contenedores Docker. Este primer paso establecerá la base sobre la cual se desarrollarán el resto de las actividades de la etapa, asegurando que el entorno de ejecución de la aplicación esté correctamente configurado y aislado.

7.1.2 Creación dockers

Las primeras tareas en el tablero Kanban a realizar serán la creación de los contenedores **Docker** para **Node.js** y **MySQL**. En esta fase inicial, se configura un entorno básico de contenedores mediante un archivo docker-compose.yml, que define estos dos contenedores principales: uno para el servicio de aplicación en **Node.js** y otro para MySQL, que almacenará la base de datos. Estos contenedores se configuran de manera que la aplicación y la base de datos estén conectadas a través de una red interna de Docker, permitiendo que interactúen entre sí de manera segura.

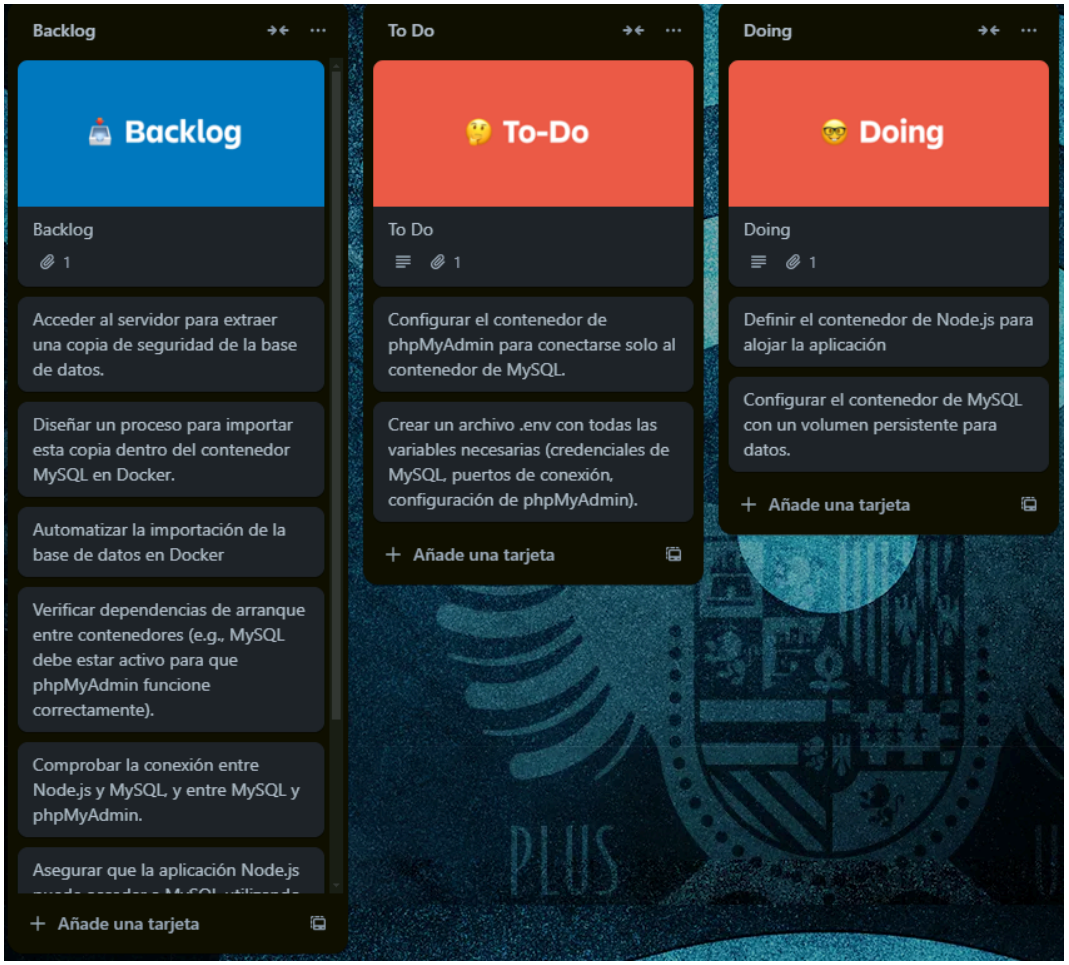


Figura 15. Tablón Kanban etapa 1.1. Creación propia con trello.

Para la aplicación, se utiliza un contenedor llamado **heimdalapp**, el cual construye la aplicación desde el contexto y Dockerfile específicos y expone los puertos necesarios para la comunicación externa e interna. Por otro lado, el contenedor **sqldocker** de **MySQL** se configura con los parámetros de autenticación y las rutas de volúmenes requeridos para almacenar y persistir los datos de manera local. Así, este docker-compose.yml proporciona la base del entorno para ejecutar y conectar los servicios esenciales del sistema en contenedores.

```yaml
#version: "3.7"

services:
  app:
  #  image: node:latest
    container_name: heimdalapp
    build:
      context: .
      dockerfile: ./Dockerfile
    links:
      - sqldocker
    restart: always
    ports:
      - 8000:8000
      - 12000:12000
    expose:
      - 8000
    volumes:
      - ./HeimdalSoundControl:/app
    networks:
        - sql_network
        - node_network


networks:
  sql_network:
    name: sql_network
    driver: bridge
```

Figura 16. Docker Nodejs. [8].

```yaml
# Acceso desde localhost:12000
  sqldocker:
    container_name: mysql
    image: mysql:latest
    volumes:
      - ./MYSQL:/var/lib/mysql
    ports:
      - 3306:3306
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
      MYSQL_ROOT_HOST: '%'
      MYSQL_DATABASE: '${DB_DATABASE}'
      MYSQL_USER: '${DB_USERNAME}'
      MYSQL_PASSWORD: '${DB_PASSWORD_USER}'

    networks:
      - sql_network

networks:
  node_network:
  #  external: true
    name: node_network
    ipam:
      config:
        - subnet: 192.168.184.0/24
  sql_network:
    name: sql_network
    driver: bridge

```

Figura 17. Docker MYSQL. [8].

```dockerfile
FROM node:latest
WORKDIR /app/dBADashboard
CMD ["node", "./appServer.js"]

```

Figura 17. Dockerfile Nodejs. [8].

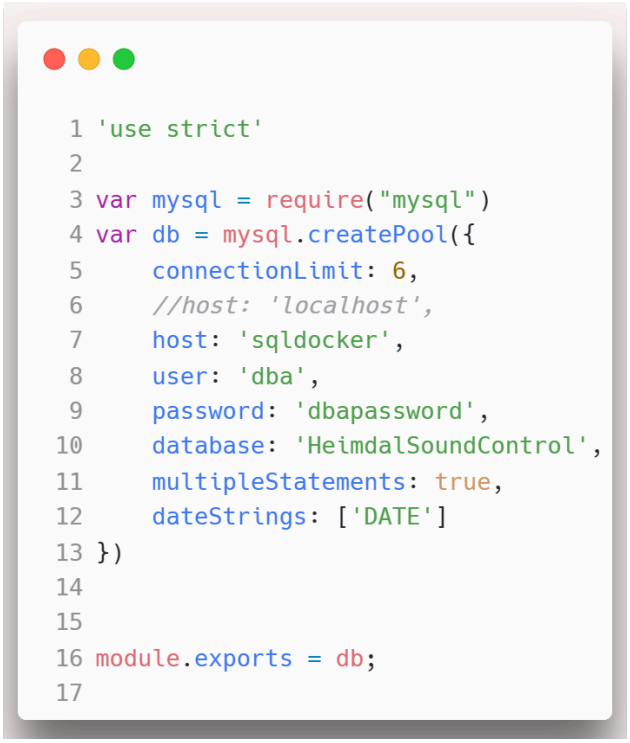
Resumen de configuración:

- **heimdalapp (Node.js)**:
  - **Construcción**: Usa un Dockerfile en el genera una imagen de Node.js que utiliza una base de node:latest, define el directorio de trabajo para la aplicación, y lanza la aplicación ejecutando el archivo appServer.js.
  - **Red**: Enlazado a sqldocker para acceder a MySQL y conectado a node_network y sql_network.
  - **Puertos**: Mapea el puerto de aplicación (8000) y de datos (12000).
  - **Volumen**: Monta el directorio de la app (./HeimdalSoundControl) para

persistencia de código.

- **sqldocker (MySQL)**:
  - **Imagen base**: Usa mysql:latest.
  - **Volumen**: Monta ./MYSQL para almacenamiento de la base de datos.
  - **Puertos**: Expuesto en el 3306 para conexiones locales.
  - **Credenciales**: Definidas mediante variables de entorno en el .env.
  - **Red**: Conectado a sql_network.
- **Redes (node_network y sql_network)**:
  - Se aíslan los servicios y facilitan la organización de la comunicación entre contenedores.

Para permitir que el servicio web se conecte al contenedor Docker de la base de datos, es necesario ajustar la dirección IP de la conexión en el archivo **dbConnect.js**. Esto implica modificar el host de la base de datos para que apunte al nombre del contenedor MySQL (en este caso, **sqldocker**), permitiendo que la aplicación web en Node.js se comunique directamente con la base de datos en Docker dentro de la misma red definida en el docker-compose. Esta modificación asegura que la conexión a MySQL funcione correctamente dentro del entorno de contenedores.



```
1  'use strict'
2
3  var mysql = require("mysql")
4  var db = mysql.createPool({
5      connectionLimit: 6,
6      //host: 'localhost',
7      host: 'sqldocker',
8      user: 'dba',
9      password: 'dbapassword',
10     database: 'HeimdalSoundControl',
11     multipleStatements: true,
12     dateStrings: ['DATE']
13 })
14
15
16 module.exports = db;
17
```

Figura 18. Archivo /HeimdalSoundControl\dBADashboard\utils\dbConnect.js. [1].

Errores producidos:

Al arrancar los contenedores e intentar acceder a la web se produce el siguiente error:

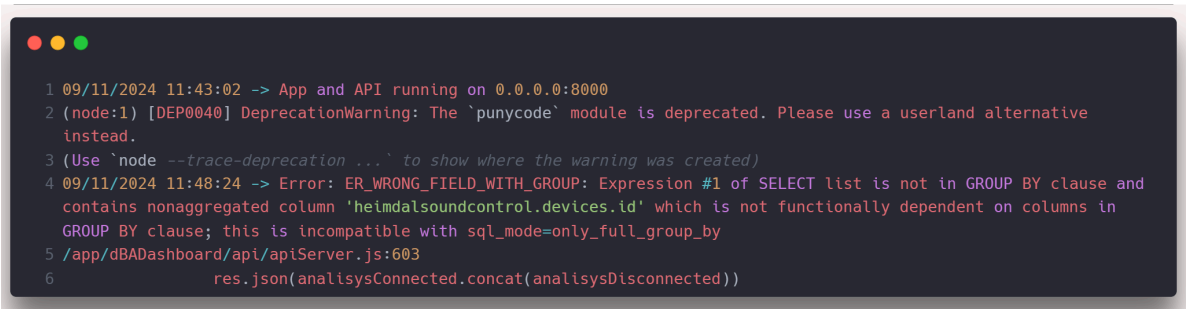


Figura 19. Fallo only_full_group_by.

Este error se debe a que el modo SQL **only_full_group_by** está activado en el contenedor MySQL, lo que requiere que todas las columnas en la cláusula **SELECT** estén definidas en la cláusula GROUP BY o sean agregadas. Esto puede evitarse deshabilitando el modo **only_full_group_by** en la configuración de **MySQL**, o adaptando las consultas SQL para cumplir con esta restricción.

Este ajuste es necesario para que la aplicación funcione sin conflictos con el estándar de agrupación en SQL.

Otro error producido es también en el acceso a la web:

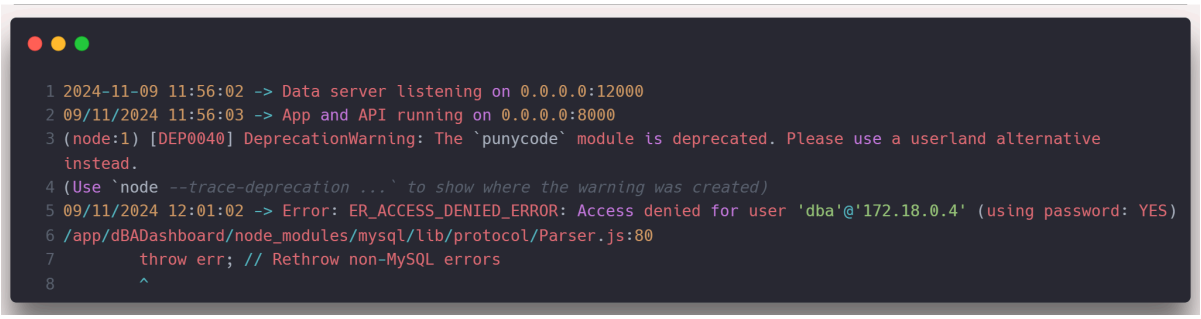


Figura 20. Fallo Native password.

El error **ER_ACCESS_DENIED_ERROR** que aparece en los logs de la aplicación Node.js indica que la aplicación no puede conectarse a la base de datos MySQL. Este problema suele estar relacionado con la configuración de los usuarios y la autenticación en MySQL, especialmente cuando se utiliza el plugin de autenticación **caching_sha2_password** en versiones recientes de MySQL. Este plugin causa incompatibilidad con el método de acceso actual de la aplicación.

La solución adoptada para corregir ambos errores fue agregar dos líneas de comando en la configuración del contenedor MySQL en el archivo **docker-compose.yml**. Estas líneas ajustan el modo de autenticación y el modo SQL para evitar los errores. Las líneas añadidas son:

```
sqldocker:
  container_name: mysql
  image: mysql:latest

  command:
    - --default-authentication-plugin=mysql_native_password
    - --sql-mode=STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION

```

Figura 21. Solución comando docker mysql. Creación propia

Con estas modificaciones, se resolvieron los problemas de conexión y de incompatibilidad con las consultas SQL.

7.1.3 Base de datos docker y phpmyadmin

En esta segunda parte de la Etapa 1, después de haber creado los contenedores de MySQL y Node.js, el siguiente paso es transferir la copia de seguridad de la base de datos desde el servidor al contenedor MySQL de Docker. Para lograr esto, primero debes exportar la base de datos del servidor en un archivo **.sql** usando herramientas como **mysqldump**. Este archivo contendrá todos los datos y estructuras de la base de datos que luego serán importados al contenedor.

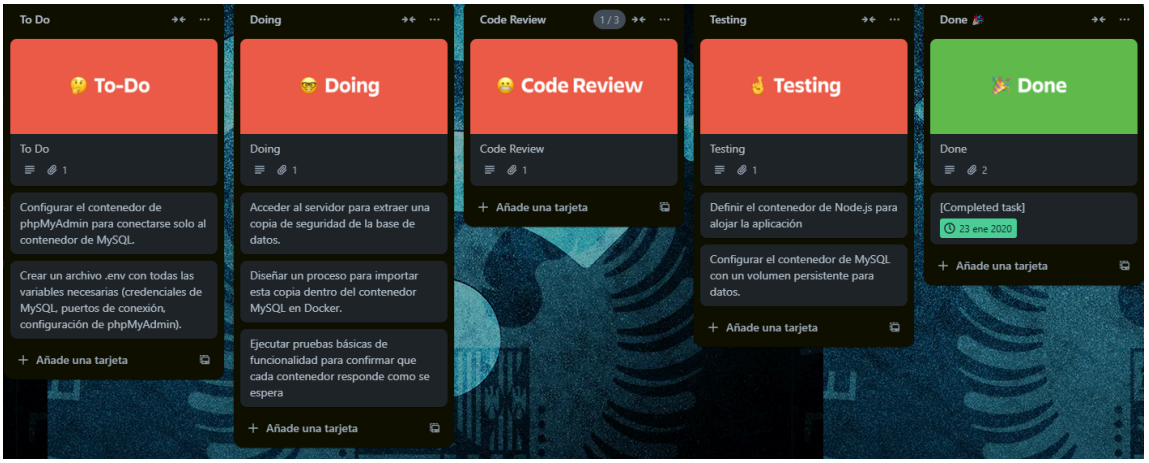


Figura 21. Tablón Kanban Etapa 1 Parte 2. Creación propia con trello.

Extracción copia de seguridad base de datos

Para crear un backup de la base de datos de Heimdal, simplemente utilizaremos un comando sencillo con **mysqldump**. Este archivo contendrá toda la estructura de la base de datos y sus datos, lo que facilita la restauración o transferencia de la base de datos.

```
## Backup mysql del servidor

mysqldump -u dba -p HeimdalSoundControl > backup/mydatabase_backup.sql --all-tablespaces

cd backup

tar cvf   mydatabase_backup.tar mydatabase_backup.sql

```

Figura 22. BackUp Base de datos servidor. Creación propia.

Restaurar la copia en contenedor

Para llevar a cabo la restauración de la base de datos, se añadirá una línea al archivo **docker-compose** que copie el archivo de copia de seguridad al directorio **/docker-entrypoint-initdb.d/** del contenedor MySQL. Esta línea se mantendrá comentada mientras no sea necesario, ya que si la base de datos ya existe en el contenedor, ejecutar la restauración podría sobrescribir los datos y causar pérdida de información. De este modo, la opción queda disponible solo cuando sea necesario restaurar la base de datos.

```
  mysql
    volumes:
      - ./${VolumeSQL}:/var/lib/mysql
    #   Descomentar - ./HeimdalSoundControlDB_backup_*.sql /docker-entrypoint-initdb.d
    #   para usar copia de seguridad y una vez arrancado esperar a que insale todo el sql en el volumen
      - ./HeimdalSoundControlDB_backup_*.sql /docker-entrypoint-initdb.d
```

Figura 23. Volúmenes docker mysql. Creación propia.

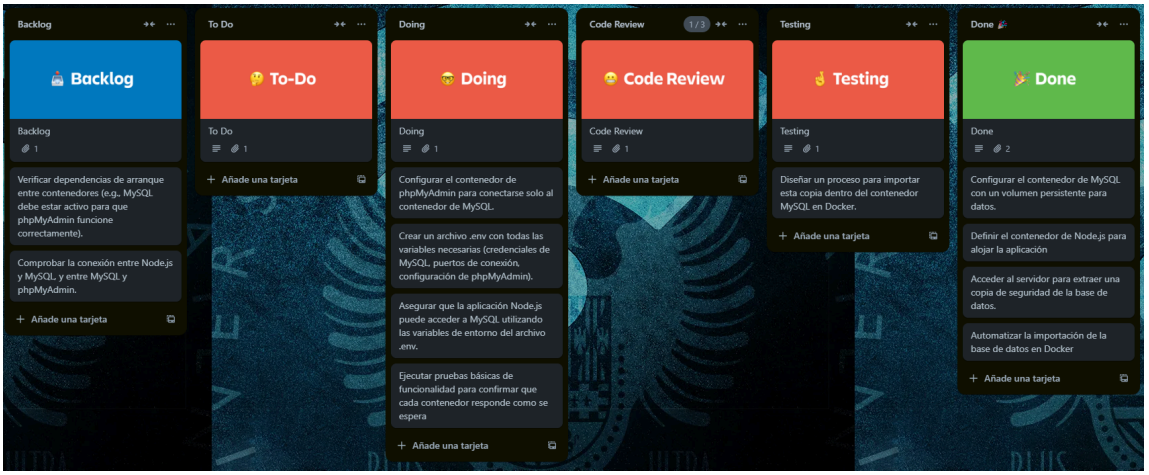


Figura 24. Tablón Kanban Etapa 1 Parte 3. Creación propia con trello.

Antes de configurar el contenedor de phpMyAdmin, se creará un archivo .env con las variables necesarias para configurar el entorno, como puertos, acceso a la base de datos y la ubicación del archivo de backup SQL. Este archivo proporcionará flexibilidad para ajustar parámetros sin modificar directamente el docker-compose, y las variables definidas serán utilizadas por los contenedores de la aplicación y la base de datos.

```
APP_PORT = 8000
SQL_PORT = 127.0.0.1:3306
PHP_PORT = 127.0.0.1:8090


DB_DATABASE = HeimdalSoundControl
DB_ROOT_PASSWORD = dbapassword
DB_USERNAME = dba
DB_PASSWORD_USER = dbapassword

VolumeSQL = MYSQL
SQL_BACKUP = mydatabase_backup.sql

APP_DATA_PORT = 12001

```

Figura 25. Archivo .env. Creación propia

A continuación, se procederá a crear un contenedor de phpMyAdmin en el archivo **docker-compose**, configurado para que solo tenga acceso al contenedor de MySQL a

través del puerto 8090. Este contenedor proporcionará una interfaz web para gestionar la base de datos de manera visual, pero estará restringido para que sólo pueda conectarse al servicio de MySQL dentro de la red del contenedor, asegurando un acceso controlado y seguro.

```yaml
# Acceso desde localhost:8090
  phpmyadmin:
    container_name: phpmyadmin
    image: phpmyadmin/phpmyadmin
    restart: always
    ports:
      - ${PHP_PORT}:80
    # Para conexion a base de datos del docker sql
    #   PMA_HOST: sqldocker
    # Para conexion a cualquier dba
    #   PMA_ARBITRARY: 1
    environment:
      PMA_HOST: sqldocker
      MYSQL_ROOT_PASSWORD: '${DB_ROOT_PASSWORD}'
    depends_on:
      - sqldocker
    networks:
      - sql_network
```

Figura 26. Configuración docker phpmyadmin. Creación propia

7.1.4 Finalización etapa 1

La Etapa 1, centrada en la implementación con Docker, está llegando a su fin. La Figura 27 muestra el tablero Kanban finalizado, reflejando el progreso de las tareas realizadas. Solo quedaría revisar el funcionamiento pleno del sistema. Además, se han añadido funcionalidades como *restart: always*, que garantiza que los servicios se reinicien automáticamente en caso de fallos, y se han establecido dependencias entre los contenedores, evitando que la web o phpMyAdmin estén activos sin que la base de datos esté en funcionamiento. Esto mejora la estabilidad y confiabilidad del sistema.

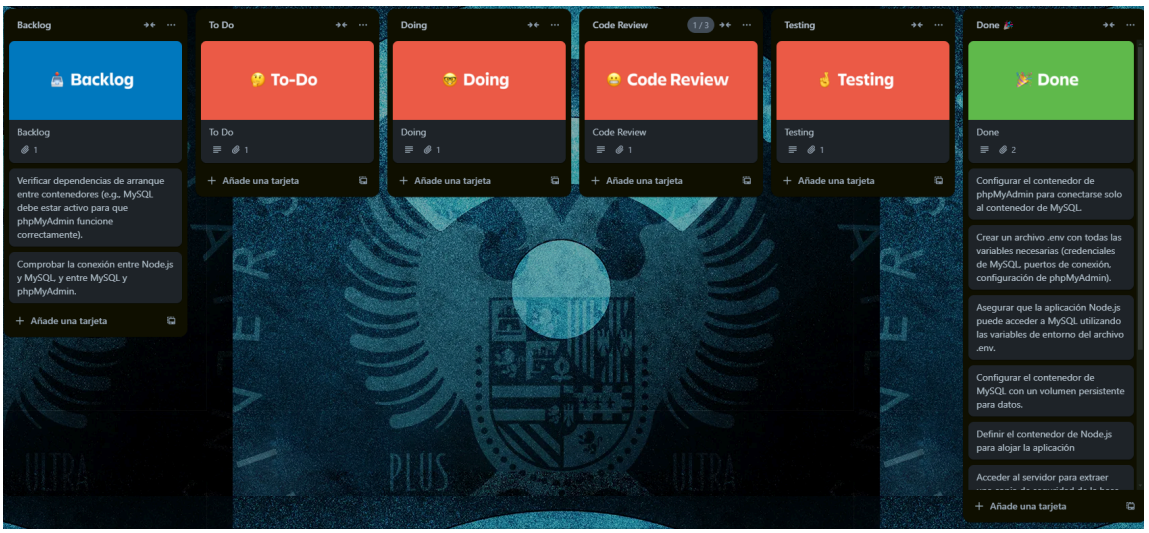


Figura 27. Tablero Kanban finalización etapa 1. Creación propia con trello.

La figura 28 muestra el pleno funcionamiento tanto de la web como de phpmyadmin en mi local, con todas las configuraciones anteriores.

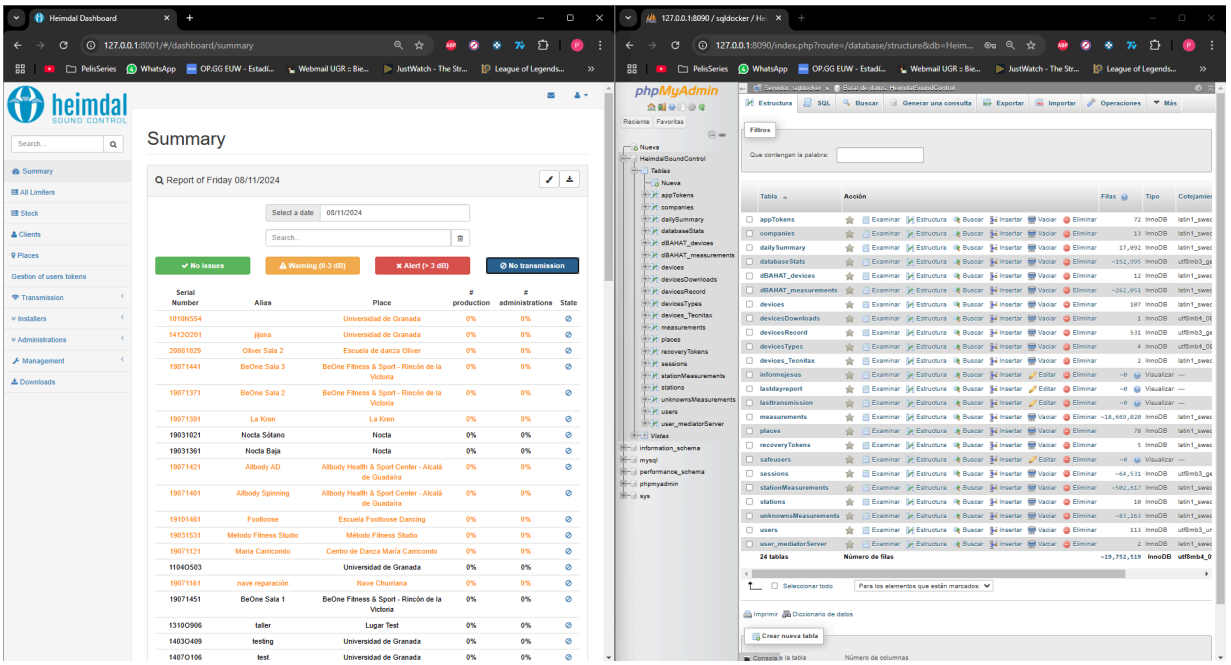


Figura 28. Servicios en localhost.

La figura 29 es el código completo del docker-compose, siendo el dockerfile del node el mostrado anteriormente. Además dicho archivo ya se encuentra comentado al completo tal y como se proyectará en el servidor en implementaciones posteriores:

```yaml
# Docker Compose para la aplicación Heimdal Sound Control - Define y configura servicios en contenedores
#Las variables usadas son creadas en un archivo .env en el directorio actual
# Especificación de versión de Docker Compose
version: "3.7"
services:
  # Servicio principal de la aplicación Node.js

  app:
    container_name: heimdalapp
    build: # La imagen se crea a partir de un Dockerfile en el directorio actual
      context: .
      dockerfile: ./Dockerfile

    links:
      - sqldocker # Conecta este contenedor con el servicio de base de datos MySQL

    restart: always

    ports:
      - ${APP_PORT}:8000 # Puerto para acceso a la aplicación principal
      - ${APP_DATA_PORT}:12000 # Puerto para recepción de datos de dispositivos
    expose:
      - ${APP_PORT}
    volumes:
      - ./:/app # Monta el directorio de trabajo local en el contenedor para desarrollo
    environment:
      MYSQL_DATABASE: '${DB_DATABASE}' # Variables de entorno de configuración de MySQL
      MYSQL_USER: '${DB_USERNAME}'
      MYSQL_PASSWORD: '${DB_PASSWORD_USER}'
      MYSQL_ROOT_PASSWORD: '${DB_ROOT_PASSWORD}'
    networks:
        - sql_network
        - node_network



  # Servicio de base de datos MySQL
  sqldocker:
    container_name: mysql
    image: mysql:8.3.0
    command:
      #Debido a la version antigua de algunas herramientas usadas en la aplicacion heimdal se debe usar el inicio de sesion mysql_native_password y configurar la variable sql_mode pquitando el ONLY_GROUP

      - --default-authentication-plugin=mysql_native_password # Configuración del plugin de autenticación
      - --sql-mode=STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION
    volumes:
      - ./${VolumeSQL}:/var/lib/mysql # Montaje para persistencia de datos en MySQL
      # Descomentar para usar copia de seguridad inicial:
      #   para usar copia de seguridad y una vez arrancado esperar a que insale todo el sql en el volumen
      # - ./${SQL_BACKUP}:/docker-entrypoint-initdb.d/init.sql
    ports:
      - ${SQL_PORT}:3306 # Puerto para acceso a la base de datos MySQL
    expose:
      - ${APP_PORT}
    restart: always

    environment:
      MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD} # Contraseña de root de MySQL
      MYSQL_ROOT_HOST: '%'
      MYSQL_DATABASE: '${DB_DATABASE}'
      MYSQL_USER: '${DB_USERNAME}'
      MYSQL_PASSWORD: '${DB_PASSWORD_USER}'
    networks:
      - sql_network



  # Servicio de phpMyAdmin para administración de MySQL
  phpmyadmin:
    container_name: phpmyadmin
    image: phpmyadmin/phpmyadmin
    restart: always
    ports:
      - ${PHP_PORT}:80 # Puerto de acceso para phpMyAdmin
    environment:
      PMA_HOST: sqldocker # Nombre del servicio MySQL al que se conectará phpMyAdmin
      MYSQL_ROOT_PASSWORD: '${DB_ROOT_PASSWORD}' # Contraseña para autenticación en MySQL
    depends_on:
      - sqldocker
    networks:
      - sql_network



networks:
  node_network:
    name: node_network
    ipam:
      config:
        - subnet: 192.168.184.0/24
  sql_network:
    name: sql_network
    driver: bridge

```

Figura 29. Docker-compose completo Creación propia.

7.1.5 Retrospectiva

La Etapa 1, aunque corta, estuvo marcada por dificultades. La falta de documentación del proyecto y su integración con Docker presentó varios problemas. Además, la restauración de la base de datos, con un gran volumen de datos, llevó hasta una hora, ralentizando el proceso. Algunos errores fueron difíciles de identificar, aumentando la complejidad. Sin embargo, esta etapa fue esencial para abandonar el uso de PM2, logrando mayor escalabilidad y mejor integración en el sistema, lo que proporcionará un entorno más robusto a largo plazo.

La etapa se considera completada, a la espera de su implementación en el servidor. Además, se ha creado un archivo Jupyter con un resumen del proceso, para facilitar futuras revisiones y permitir a otros desarrolladores comprender rápidamente los pasos clave de esta fase.

7.2 Etapa 2: Simulador dispositivos

La Etapa 2 se enfoca en el desarrollo de un programa en Arduino para simular el envío de datos al backend de la aplicación, representando información de dispositivos registrados en la web. El programa generará datos falsos de varios dispositivos siguiendo un patrón realista, permitiendo verificar el correcto procesamiento y almacenamiento de la información en el sistema. Esta etapa es clave para crear un entorno de prueba que refleje condiciones reales, facilitando ajustes y mejoras en el procesamiento de datos en la aplicación.

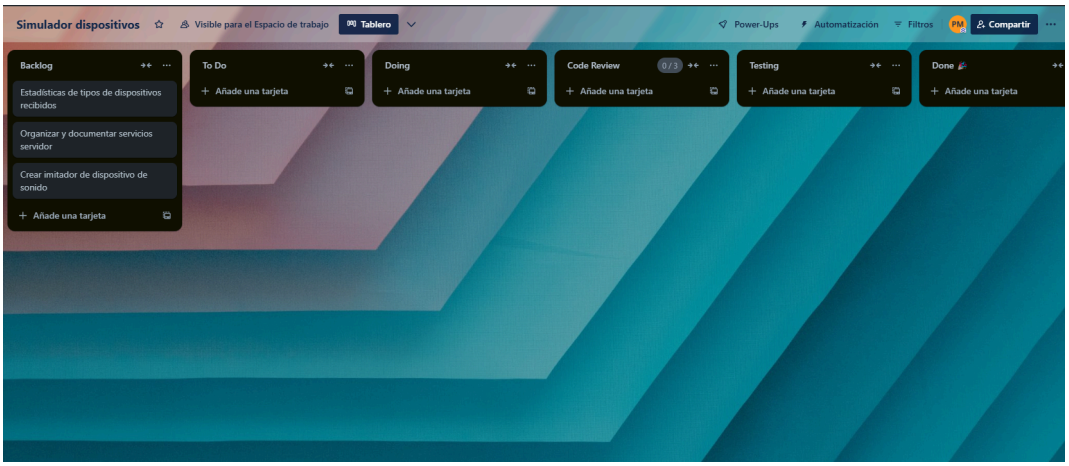


Figura 30. Tablón Kanban Etapa 2 Inicial. Creación propia con trello.

Al igual que en la Etapa 1, en la Etapa 2 se crearán tareas más específicas y detalladas basadas en el diseño inicial. Estas tareas estarán enfocadas en el desarrollo del programa en Arduino para simular datos de dispositivos, siguiendo un formato realista y estructurado. Dado que todas las tareas son esenciales para alcanzar el objetivo de esta etapa, no se asignarán prioridades diferenciadas. Además, estarán orientadas a anticipar y resolver posibles problemas, asegurando una implementación estable y controlada.

7.2.1 Tareas a realizar

En esta etapa, se desarrollará un programa en Arduino Uno con un shield Ethernet para simular la transmisión de datos de sonido. El Arduino enviará paquetes de datos a través de TCP usando una IP fija y un puerto Ethernet. Conectado a un portátil configurado para compartir internet, el dispositivo se comunicará con el servidor.

El programa generará paquetes de datos de sonido en formatos hexadecimales y UTF-8, que se convertirán a bytes. El servidor analizará el código para determinar el formato de los paquetes y el tipo de datos enviados.

La primera versión incluirá un menú desde el puerto serie para:

- Enviar un paquete de datos.
- Leer y configurar la hora en el Arduino.
- Guardar la hora actual en la EEPROM junto al reloj interno.

Además, el servidor mejorará el registro de paquetes, clasificándolos en categorías: CAP 21 X, CAP 21 Y, Otros funcionales y errores de formato, los cuales se almacenarán en archivos .txt en la carpeta logsData.

Historia de usuario	Tareas
Un programa en Arduino que envíe paquetes de datos de sonido simulados por TCP	1. Configurar el Arduino Uno con shield Ethernet para conectividad TCP con IP fija. 2. Crear la lógica de generación de paquetes de datos de sonido, incluyendo el ensamblaje de variables en diferentes formatos (hex y UTF-8) y la conversión a bytes. 3. Enviar paquetes simulados al servidor desde el portátil compartiendo internet por Ethernet. 4. Probar el envío y recepción de los paquetes, verificando que el servidor los procese sin errores.
Como administrador del sistema, quiero que el Arduino permite configurar y guardar la hora en la EEPROM	5. Implementar un menú en el puerto serie que permita leer y guardar la hora actual. 6. Utilizar la EEPROM de Arduino para almacenar la hora, sincronizando con el reloj interno
Como usuario del sistema, quiero que el servidor cuente y clasifique todos los paquetes recibidos en varias categorías y guarde los erróneos	7. En el servidor, se desarrolla el procedimiento que cuenta los paquetes recibidos y los clasifica por sus tipos. 8. Implementar la funcionalidad para registrar los paquetes con errores en archivos .txt dentro de una carpeta logsData.
Como desarrollador, quiero crear una	9. Crear una segunda versión del programa en Arduino, que permite arrancar y mantener el arduino funcionando sin envio por puerto serie,

segunda versión del programa que permita individualidad	usando peticiones a un servidor de fecha y hora europa. 10. Ajustar por horarios de locales los datos. 11. Aumentar el realismo de los datos mandados.
Revisar la conexión entre el arduino y el servidor en red	12. Probar su funcionamiento con ethernet desde router.
Pruebas de funcionamiento	13. Los datos se envían correctamente tanto al local, como al servidor, con fecha y hora correcta.

Tabla 5. Tabla tareas etapa 2. Creación propia.

Al igual que en la Etapa 1, estas tareas detalladas se trasladarán al tablero Kanban para gestionarlas de forma visual y ordenada. Esto facilitará el seguimiento del progreso y la identificación de bloqueos o prioridades en la implementación del simulador de datos en Arduino.

Las tareas se abordarán en el orden establecido, comenzando por la configuración de la conexión del Arduino y el desarrollo de la lógica para generar y enviar los datos de sonido. Este primer paso establecerá la base para el resto de actividades, garantizando que los paquetes generados sean compatibles con el formato requerido por el servidor.

Debido a la cantidad de tareas, se han dividido en dos columnas: "To Do" y "To Do Later", ordenadas por prioridad de forma descendente.

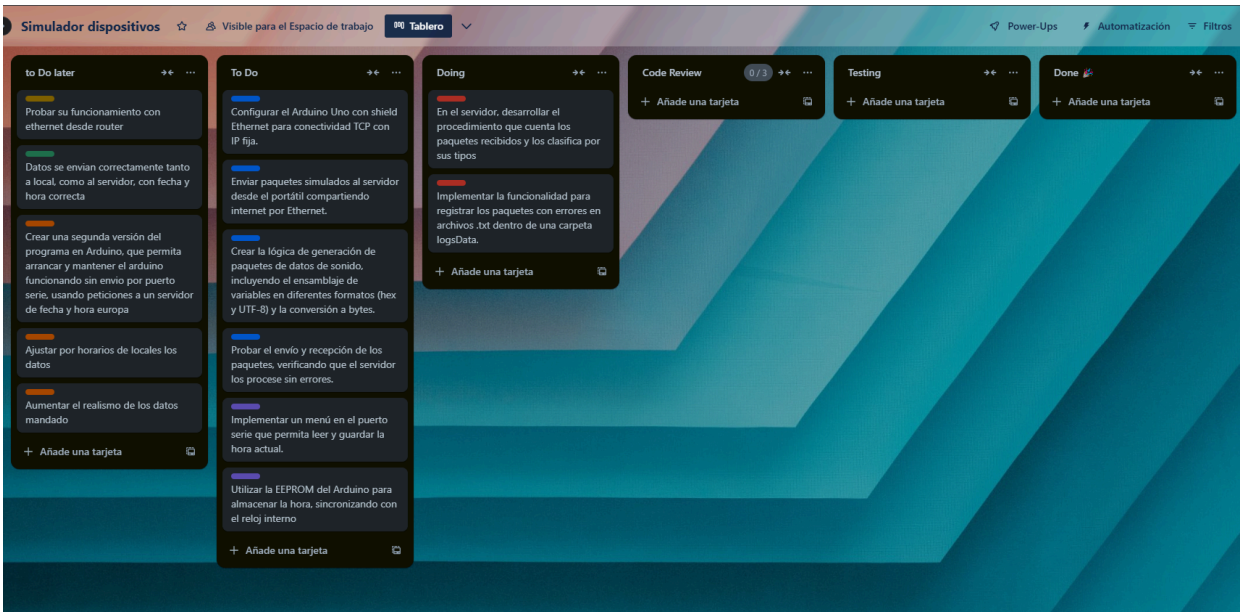


Figura 31. Tablón Kanban Etapa 2 con tareas. Creación propia con trello.

7.2.2 Registro y clasificación de paquetes de datos

Como se refleja en la figura 31, las primeras tareas a realizar en esta etapa serán la clasificación de paquetes a la aplicación, esto es debido a un problema encontrado en la realización del proyecto, debido a que los limitadores usados por los locales son creados por empresas privadas la manera que implementan el empaquetado de los datos está clasificada, y debido a la falta de documentación y comentarios en el código actual resulta complicado ver el formato de dichos datos.

El funcionamiento de los datos de la aplicación luego de estudiarlos es una cadena en formato bytes la cual se traduce por tipos de dispositivos, se pasan los primeros 5 bytes a utf-8, los pasos se muestran en el siguiente diagrama (figura 32):

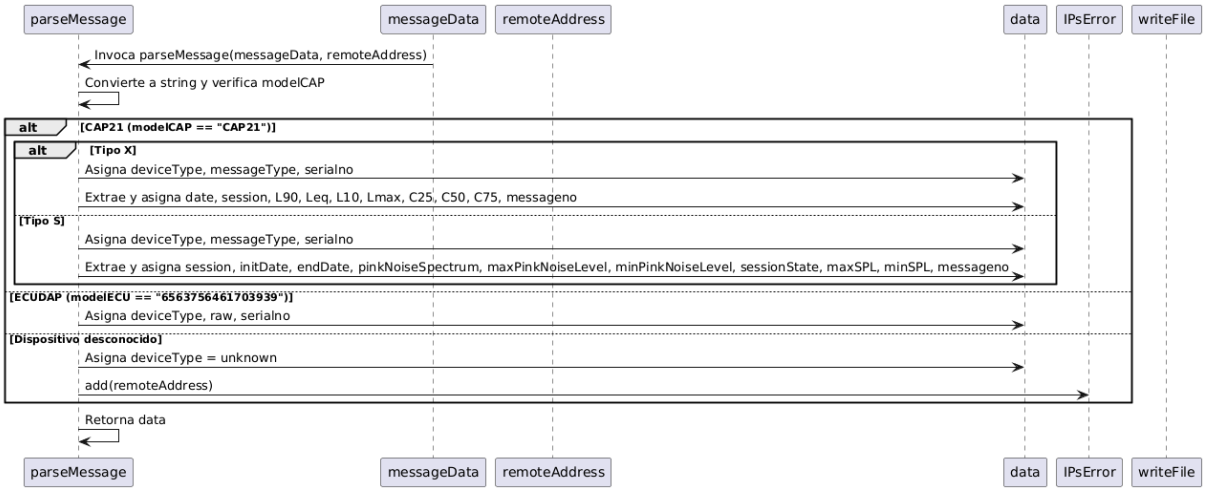


Figura 32. Diagrama de secuencia simple parseo de datos.

A continuación, mostraré en forma de código como traduce cada sección del paquete posición de los bytes, por comodidad solo se mostrará los tipo CAP 21 tipo X que serán los simulados en el programa.

```
    var data =   {}
    //TCP MODE
    var modelCAP = messageData.toString("utf-8", 0, 6)
    var modelCAPYo10 = messageData.toString('hex').substring(0, 12) // 010101010001 or 000000000000
    var modelECU = messageData.toString().substring(0, 16)
    console.log("modelCAP: " + modelCAP);

    if (modelCAP == " CAP21" || modelCAPYo10 == "010101010001" || modelCAPYo10 == "000000000000") {
        var type = messageData.toString("utf-8", 15, 16)
        console.log("type: " + type);
        if (type == "X") {
            counter.cap21X++;
            data.deviceType = "CAP21"
            data.messageType = type

            data.serialno = messageData.toString("utf-8", 7, 15)

            console.log("serialnoX: " + data.serialno);
            data.date = (2000 + parseInt(messageData.toString("hex", 18, 19), 16)).toString() + "-" + //year
                pad(parseInt(messageData.toString("hex", 17, 18), 16), 2) + "-" + //month
                pad(parseInt(messageData.toString("hex", 16, 17), 16), 2) + " " + //day
                pad(parseInt(messageData.toString("hex", 20, 21), 16), 2) + ":" + //hour
                pad(parseInt(messageData.toString("hex", 19, 20), 16), 2) + ":00" //minute

            data.session = (parseInt(messageData.toString("hex", 22, 23), 16) * 256) +
parseInt(messageData.toString("hex", 21, 22), 16)
            data.session = isNaN(data.session) ? -1 : data.session;

            data.L90 = parseInt(messageData.toString("hex", 23, 24), 16)
            data.L90 = isNaN(data.L90) ? -1 : data.L90;

            data.Leq = parseInt(messageData.toString("hex", 24, 25), 16)
            data.Leq = isNaN(data.Leq) ? -1 : data.Leq;

            data.L10 = parseInt(messageData.toString("hex", 25, 26), 16)
            data.L10 = isNaN(data.L10) ? -1 : data.L10;

            data.Lmax = parseInt(messageData.toString("hex", 26, 27), 16)
            data.Lmax = isNaN(data.Lmax) ? -1 : data.Lmax;

            data.C25 = parseInt(messageData.toString("hex", 27, 28), 16)
            data.C25 = isNaN(data.C25) ? -1 : data.C25;

            data.C50 = parseInt(messageData.toString("hex", 28, 29), 16)
            data.C50 = isNaN(data.C50) ? -1 : data.C50;

            data.C75 = parseInt(messageData.toString("hex", 29, 30), 16)
            data.C75 = isNaN(data.C75) ? -1 : data.C75;

            data.messageno = parseInt(messageData.toString("hex", 30, 31), 16)
            data.messageno = isNaN(data.messageno) ? -1 : data.messageno;

```

Figura 33. Segmento función parsemessage.

En la función parseMessage, se ha integrado un objeto counter que lleva un seguimiento de la cantidad de dispositivos procesados por tipo y de los dispositivos desconocidos con el siguiente formato:

```
var counter = {
    unknown : 0,
    cap21X : 0,
    cap21S : 0,
    cap21Unknown: 0,
    modelECU : 0,
    total : function(){
        return this.unknown +
               this.cap21X +
               this.cap21S +
               this.cap21Unknown +
               this.modelECU;
}
```

Figura 34. Estruct counter. Creación propia.

El siguiente diagrama de secuencia muestra cómo se procesa cada paquete de datos, cómo se clasifican según su tipo, y cómo se gestionan las estadísticas y los registros de errores en función de los datos recibidos. Además, se ilustran las interacciones entre las funciones responsables de manejar estos procesos, incluyendo la creación de archivos de logs y la actualización de estadísticas.

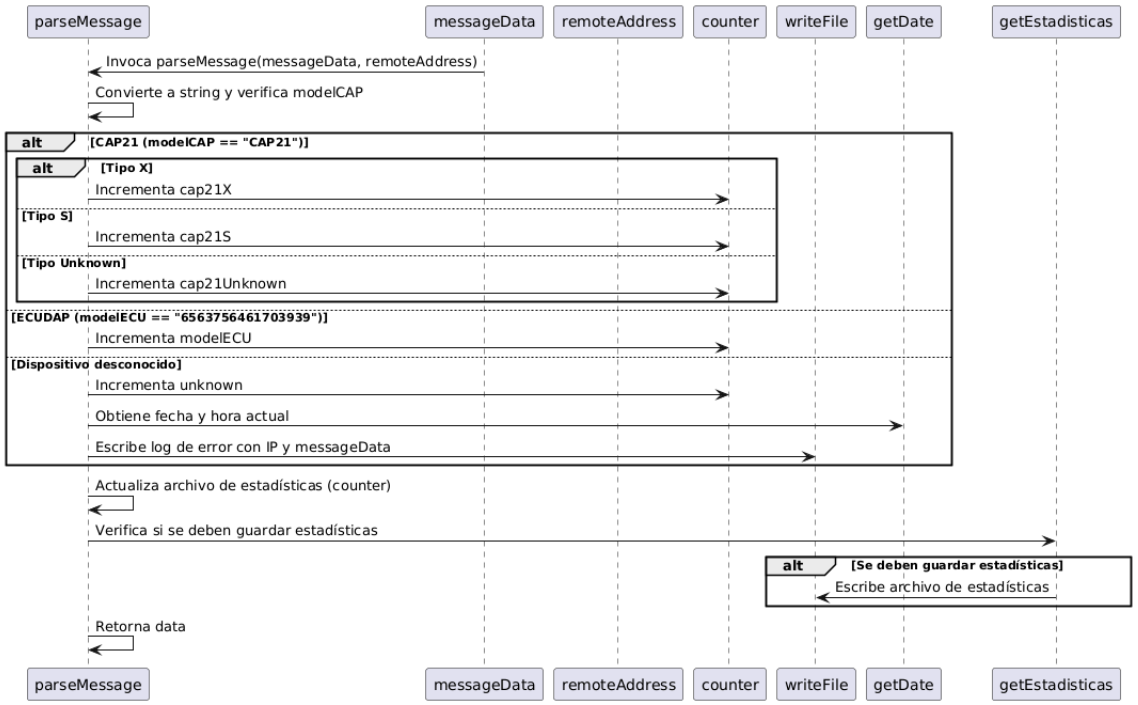


Figura 35. Diagrama secuencia recogida estadísticas parseMessage. Creación propia.

Como se puede ver en la figura 35, cuando llega un paquete de tipo desconocido, el sistema guarda un archivo en la carpeta **logsData** que contiene la IP y la fecha del origen del error. Este archivo de logs ayuda a identificar la fuente de los paquetes desconocidos. Sin embargo, una vez que se alcanza un límite predefinido de archivos guardados, el sistema deja de registrar estos errores, lo que ayuda a controlar el tamaño del almacenamiento utilizado por los logs. A su vez, para no tener un contador infinito en el contador de tipos de paquetes, cada cierta cantidad de paquetes, deja de escribir en el mismo fichero y crea uno nuevo, dejando así una lista de **logsDataEstadisticas** en el tiempo.

7.2.3 Datos en el paquete

Como ya he mencionado anteriormente, en este caso sólo se considerará el modelo **CAP21 tipo X** (A pasear de mandar datos de otros tipos, que serán un valor casi nulo debido a que es innecesario su valor). A continuación, se detallan los datos que conforman el paquete de este modelo, listados en el orden en el que se encuentran en el mensaje. Estos datos incluyen información tanto sobre el dispositivo como sobre las mediciones de sonido realizadas, y son utilizados para procesar y registrar el evento de manera precisa.

- **modelCAP**: Modelo del equipo. Es una cadena de caracteres que representa que es un dispositivo tipo CAP21.
- **modelCAPYo10**: Identificador único del modelo. Es un número largo que se utiliza para identificar de manera exclusiva el modelo del dispositivo.
- **modelECU**: Identificador para identificar si es un dispositivo tipo ECU.
- **type**: Tipo de mensaje. Es un solo carácter que indica el tipo de mensaje que se está enviando, como "X", "S", etc.
- **serialno**: Número de serie del dispositivo. Es una cadena de caracteres que sirve para identificar de manera única al dispositivo.
- **year, month, day, hour, minute**: Fecha y hora del paquete.
- **session**: Número de sesión. Es un número entero que identifica de forma única la sesión durante la cual se realiza la medición.
- **L90, Leq, L10, Lmax**: Valores medio de sonido por tiempo.
- **C25, C50 y C75:** Otros valores**.**
- **messageno**: Número de mensaje. Es un número entero que actúa como identificador secuencial para ordenar los paquetes de datos.

7.2.4 Programa envio datos arduino

Ya finalizado el cálculo y ver los tipos de datos que contiene el paquete procedemos a la siguiente parte de la etapa, donde crearemos un programa arduino, usando un **arduino uno con un shield de ethernet.** Como se muestra en la figura 36.

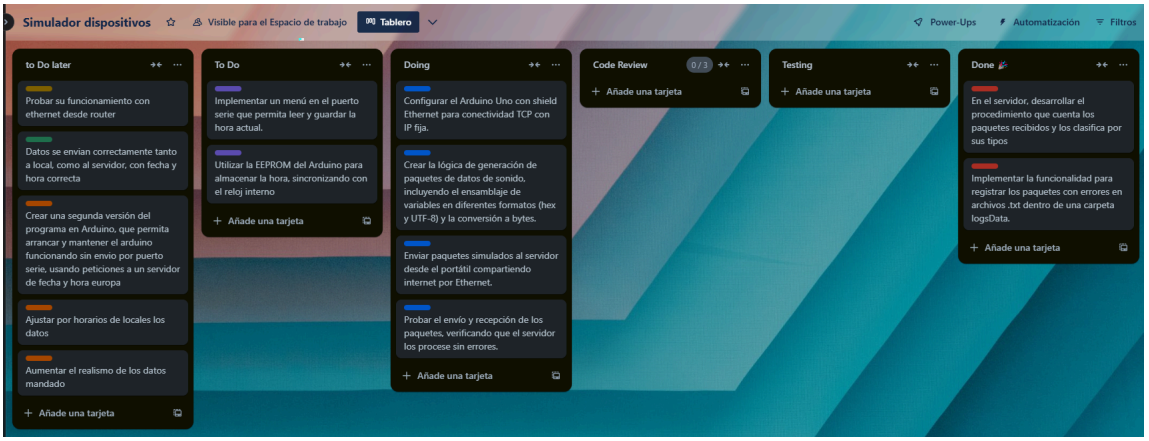


Figura 36. Tablón Kanban Etapa 2 parte 1. Creación propia con trello.

La primera tarea será configurar el portátil para permitir que desde un arduino conectado por ethernet tenga acceso a internet, para ello desde

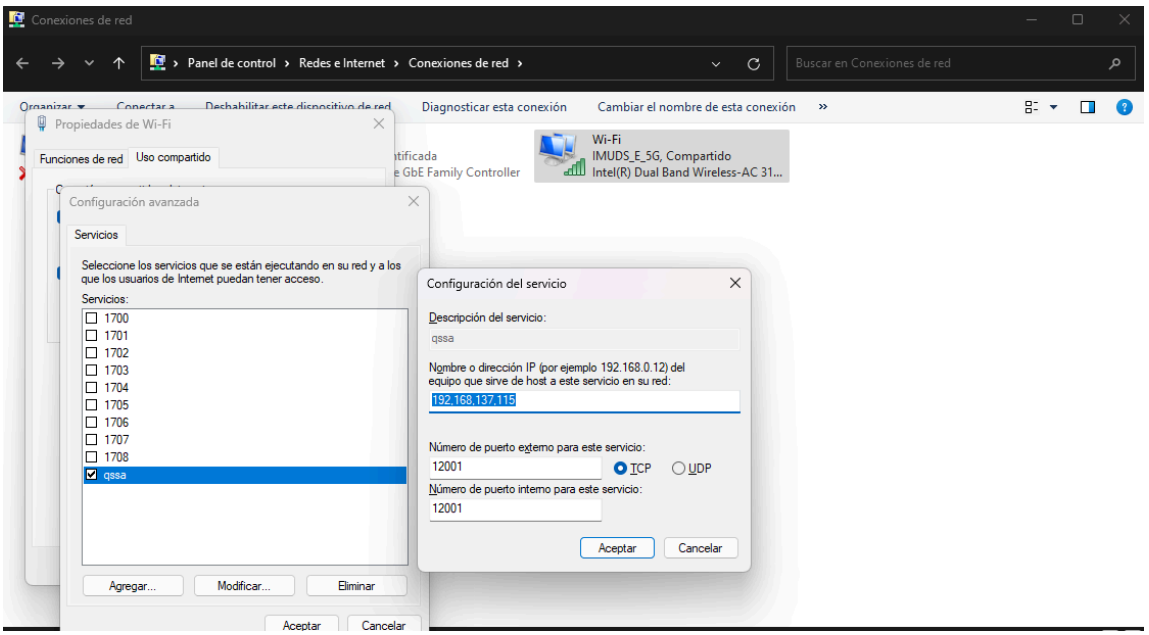


Figura 36. Darle acceso a internet en windows. Creación propia.

Dado que se trata de un arduino que requiere acceso a la red (local o internet) para el envío de datos al servidor es necesario configurar el puerto ethernet para su correcto envío. Principalmente para facilitar el trabajo y no requerir modificar la ip del programa se pondrá como ip fija la ip del puerto ethernet.

- En Windows desde Panel de **control\Redes e Internet\Conexiones** de red seleccionas tu ethernet de la lista de conexiones y en propiedades seleccionas el elemento TCP/IPv4 y en sus propiedades marcas para IP fija y pones la que mejor te convenga, en este caso 192.168.137.1 y mascara 255.255.255.0

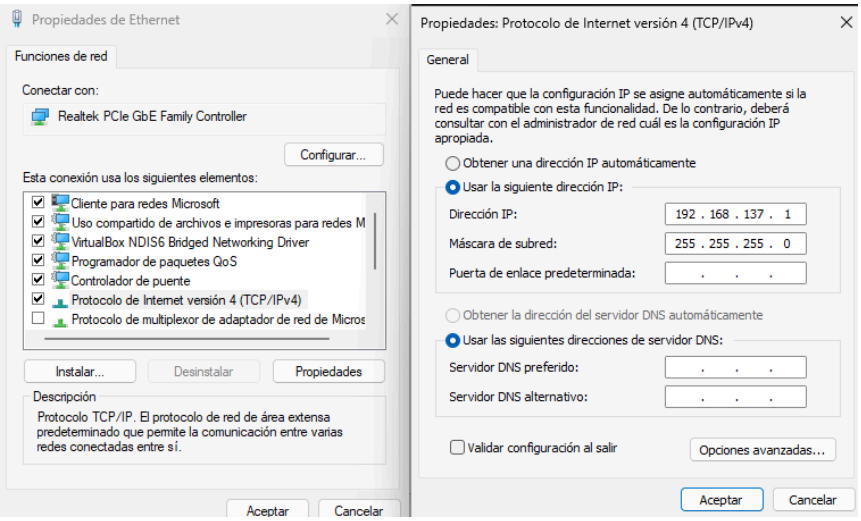


Figura 37. Ip fija en windows. Creación propia.

Variables envio de config.h

A continuación se presenta el archivo **config.h**, que define las configuraciones de red para el dispositivo. Este archivo contiene las direcciones esenciales necesarias para establecer la comunicación del dispositivo con la red, como la dirección MAC, la IP estática del dispositivo, la dirección del gateway, el servidor DNS, la máscara de subred, y las direcciones de los servidores a los que el dispositivo debe conectarse. También incluye configuraciones adicionales, como el puerto de transmisión y el modo de depuración, que facilitan la interacción con otros dispositivos y servidores dentro de la red.

Las siguientes variables son utilizadas para gestionar las conexiones de red necesarias para la comunicación del dispositivo:

- **IPADDR**: Dirección IP estática asignada al Arduino.
- **GWADDR**: Dirección IP del gateway o la conexión Ethernet del PC (IP fija de la interfaz Ethernet).
- **SERVERPORT**: Puerto TCP utilizado para la conexión con el servidor.
- **SERVERADDRIP**: Dirección IP del servidor al que se conecta el dispositivo.

```
/**
 * @file config.h
 * @brief Definición de las configuraciones de red para el dispositivo.
 *
 * Este archivo contiene las configuraciones de red, como la dirección MAC, IP,
 * dirección del gateway, DNS, máscara de subred, y otras configuraciones necesarias
 * para establecer la comunicación del dispositivo.
 *
 * @author Pablo Puga Martínez
 * @date 2024-06-15
 */

#define MACADDR {0xDE, 0xAD, 0x85, 0x65, 0xFE, 0xED}    ///< Dirección MAC del dispositivo
#define IPADDR {192,168,1,115}                          ///< Dirección IP del dispositivo
#define GWADDR {92,168,1,1}                            ///< Dirección IP del gateway
#define DNS {8,8,8,8}                                   ///< Dirección IP del servidor DNS
#define SUBNET {255, 255, 255, 0}                       ///< Máscara de subred
#define RESPONSELEN 31                                  ///< Cantidad de bytes en el paquete de respuesta
#define TESTSERVERADDR {192,168,0,2}                    ///< Dirección IP del servidor de pruebas
#define TESTSERVERPORT 12000                            ///< Puerto TCP del servidor de pruebas
#define SERVERPORT 12001                                ///< Puerto TCP del servidor de transmisión
#define DEBUG true                                      ///< Modo de depuración: añade 268 bytes a la SRAM
#define SERVERADDRIP {146, 59, 146, 98}                 ///< Dirección IP del servidor de transmisión
```

Figura 38. Variables de entorno de conexión. Creación propia.

Clase JsonUtil.h

Este archivo facilita la creación y manipulación de objetos JSON para enviar datos entre el Arduino y otros dispositivos, utilizando las propiedades y métodos definidos en la estructura JsonObject, así como un array donde se almacenan los números de serie que el arduino simula.

A su vez está el archivo JsonUtil.cpp con los constructores necesarios usados en el código principal pero innecesarios ya que solo asigna los valores a sus campos correspondientes.

```cpp
/**
 * @file jsonUtils.h
 * @brief Definición de estructuras y funciones relacionadas con el manejo del paquete de datos a mandar.
 *
 * Este archivo contiene las definiciones para manejar objetos del archivo a mandar, así como funciones
 * para manipular y serializar los datos que se utilizarán en la comunicación entre dispositivos.
 *
 * @version 1.0
 * @date 2024-06-15
 */


#ifndef JSON_UTILS_H
#define JSON_UTILS_H

#include <Arduino.h>  ///< Incluye las definiciones necesarias para trabajar con Arduino

/// Tamaño del array de números de serie
const int TAMSERIAL = 10;
/// Lista de números de serie almacenados
const char serialNumbers[TAMSERIAL][9] = {
  "13070603", "19031021", "14030408", "14070106", "14070106",
  "1009N556", "1002N304", "12100706", "09110401", "16121891"
};

/**
 * @struct JsonObject
 * @brief Estructura para almacenar datos en formato JSON.
 *
 * Esta estructura contiene diversos campos que representan los datos capturados
 * y enviados en formato JSON, como la fecha, hora, valores de medición, y otros.
 */
struct JsonObject {
    String modelCAP;         ///< Modelo del equipo
    long int modelCAPYo10;   ///< Identificador del modelo en formato largo
    char modelECU[20];       ///< Identificador de la ECU
    char type;               ///< Tipo de mensaje
    String serialno;         ///< Número de serie
    int year, month, day, hour, minute; ///< Fecha y hora del evento
    int session, L90, Leq, L10, Lmax;   ///< Valores de medición de sonido
    int C25, C50, C75;       ///< Otros parámetros
    int messageno;           ///< Número del mensaje

    /// Constructores
    JsonObject();
    //.... Hay mas
};

#endif

```

Figura 38. Clase JsonObject y array serialNumbers. Creación propia.

Conexión Arduino a la Aplicación:

El Arduino establece una conexión TCP a través de su puerto Ethernet, utilizando una IP fija configurada directamente en el código. Esta conexión se realiza mediante las variables de configuración previamente detalladas, donde se define la dirección IP (IPADDR). A continuación, se establece una conexión hacia el servidor, permitiendo que el dispositivo se comunique de manera efectiva con la aplicación.

7.2.5 Flujo del Programa:

El flujo del programa en Arduino se divide en dos partes principales: la función SETUP() y el bucle loop().

SETUP

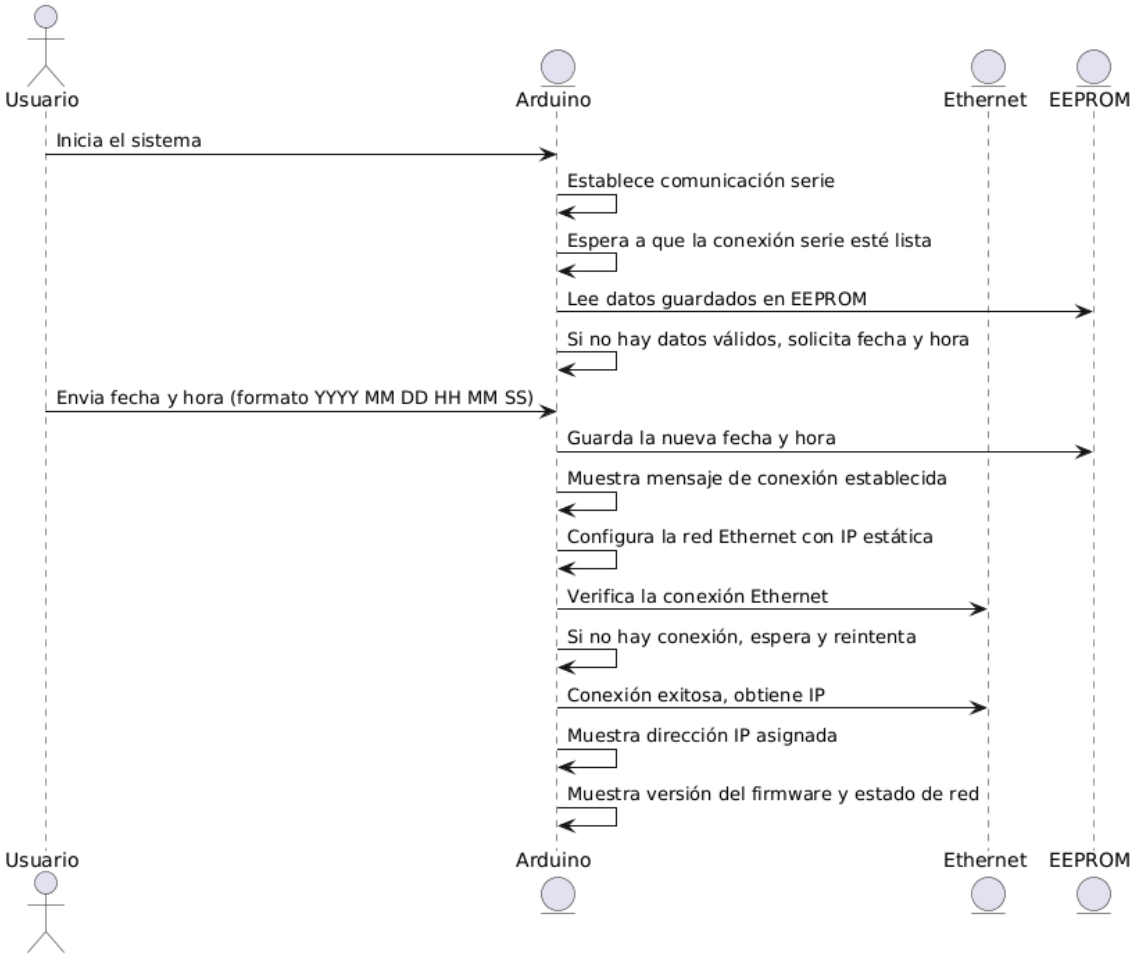


Figura 39. Diagrama de secuencia SETUP. Creación propia.

Este diagrama de secuencia describe el flujo de la función **setup()** en un programa de Arduino. Primero, el sistema inicia la comunicación serie y se asegura de que el puerto serie esté listo. Luego, verifica si hay una fecha guardada en la EEPROM. Si no la hay, solicita al usuario ingresar la fecha y hora. Una vez obtenida, la guarda en la EEPROM. A continuación, se establece la conexión Ethernet, que se configura con una IP estática y se mantiene en un bucle hasta que la conexión se complete. Finalmente, se imprime la dirección IP asignada y se muestra la versión del firmware en el puerto serie.

LOOP

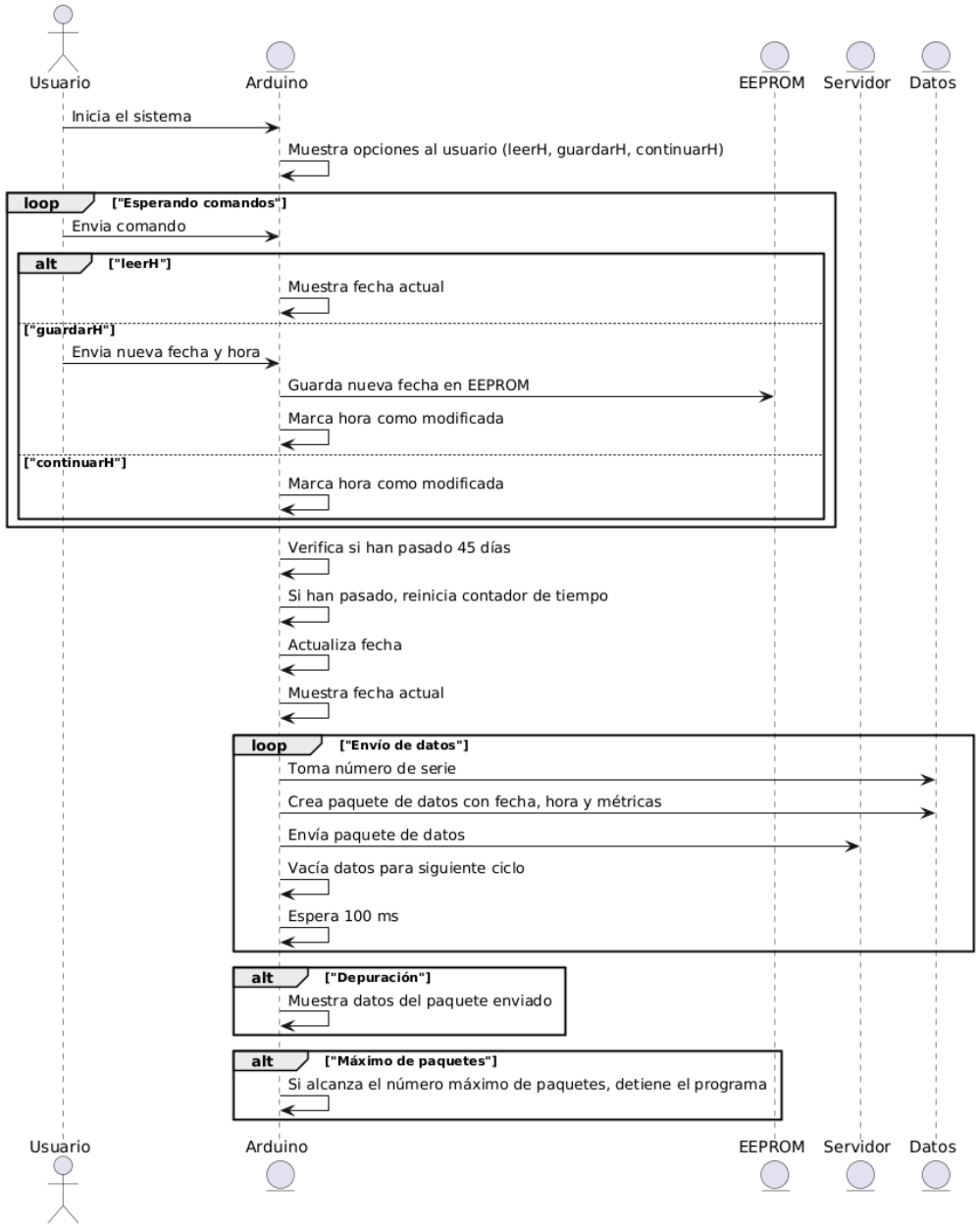


Figura 40. Diagrama de secuencia LOOP. Creación propia.

En la función **loop(),** el programa muestra un conjunto de opciones al usuario a través del puerto serie: leer la fecha actual, modificar la fecha, o continuar sin modificar la hora. El programa espera un comando del usuario durante un tiempo determinado, y dependiendo del comando ingresado, muestra la fecha, actualiza la fecha en la EEPROM o simplemente marca que la hora ha sido modificada.

Después de este paso, el programa verifica si han pasado más de 45 días desde que comenzó. Si es así, reinicia el contador de tiempo. A continuación, se aumenta la fecha actual y se muestra en el puerto serie.

En el siguiente paso, el programa genera y envía paquetes de datos con información sobre la fecha, hora y métricas aleatorias de ruido, para cada número de serie. Si el modo de depuración está habilitado, muestra el contenido del paquete antes de enviarlo. El proceso continúa hasta que se alcanza el número máximo de paquetes permitidos, momento en el que el programa se detiene.

Para entender el pleno funcionamiento de cómo se crea el paquete, se debe ver el codigo en sí, de cómo se crea, el paquete, pasándole el objeto JsonObject creado en el loop explicado anteriormente y pasando cada valor en bytes uno a uno en un orden exacto e incluso dejando huecos o poniendo letras para darle forma.

```
/**
 * @brief Convierte un entero a byte.
 *
 * @param valor Entero a convertir.
 * @return byte Valor convertido a byte.
 */
byte intToByte(int valor) {
    return valor & 0xFF;
}

/**
 * @brief Añade un byte al array auxiliar.
 *
 * @param byte Byte a añadir al array.
 */
void addByte(byte byte) {
    if (arrayLength < 31) {
        aux[arrayLength] = byte;
        arrayLength++;
    } else {
        Serial.println("El array de bytes está lleno. No se pueden añadir más bytes.");
    }
}

/**
 * @brief Crea un array de bytes a partir de un objeto JSON.
 *
 * @param data Objeto JSON que contiene los datos a enviar.
 */
void createBytesArray(JsonObject data) {
    String cadena = data.modelCAP + "C" + data.serialno + String(data.type);
    arrayLength += cadena.length();
    cadena.getBytes(aux, cadena.length() + 1);

    addByte(intToByte(data.day));
    addByte(intToByte(data.month));
    addByte(intToByte(data.year));
    addByte(intToByte(data.minute));
    addByte(intToByte(data.hour));
    addByte(intToByte(0xFF)); // Fin de sección
    addByte(intToByte(data.session));
    addByte(intToByte(data.L90));
    addByte(intToByte(data.Leq));
    addByte(intToByte(data.L10));
    addByte(intToByte(data.Lmax));
    addByte(intToByte(data.C25));
    addByte(intToByte(data.C50));
    addByte(intToByte(data.C75));
    addByte(intToByte(data.messageno));
}

```

Figura 41. Función createBytesArray. Creación propia.

Conforme se ha ido avanzando se han ido desarrollando muchas de las tareas marcadas en el tablón Kanban, por lo que debemos actualizarlo para llevar un seguimiento adecuado:

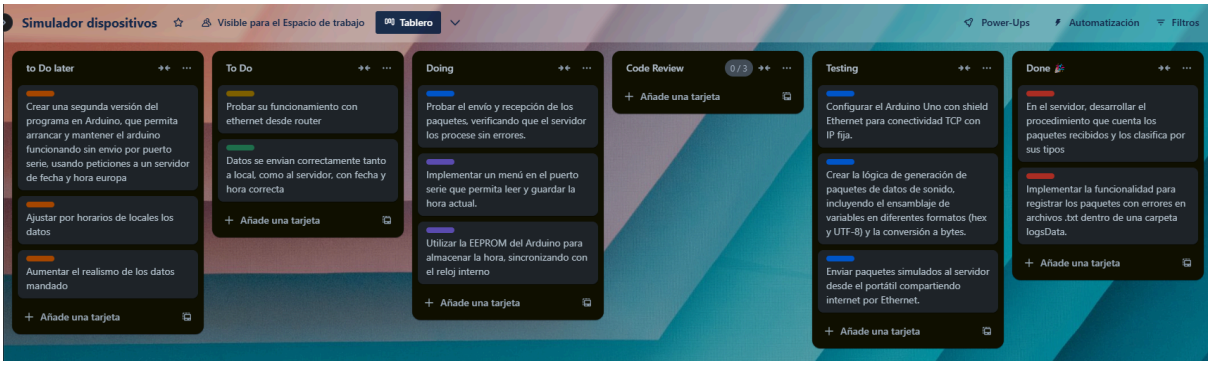


Figura 42. Tablón Kanban Etapa 2 fase 1. Creación propia.

Como se puede observar en la figura 42, las tareas de creación del código y funcionamiento del mismo se han completado, debido a que ha habido un avance intermedio de otras tareas como el uso de EEPROM o el menú las comprobaciones de funcionalidad se mostrarán más adelante.

7.2.6 Control de fecha

Continuaremos la implementación con el uso de la EEPROM y un control de fechas y horas que ya se habían mostrado en el punto pasado debido a que hubo un avance inesperado.

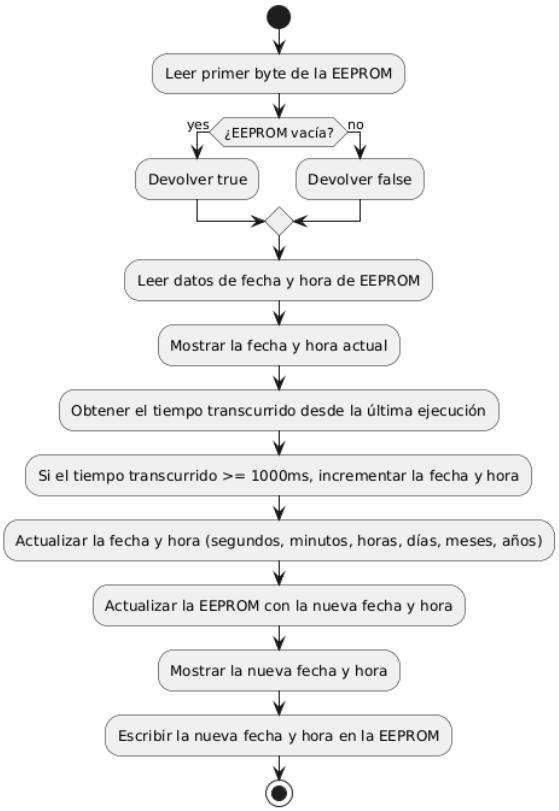


Figura 42. Diagrama de flujo control de fecha. Creación propia.

La figura 42 representa el flujo de cómo el programa maneja la fecha y hora, pidiéndola por el puerto serie, con el menú explicado anteriormente. Estos datos se manejan guardando en la memoria EEPROM de arduino con las siguientes funciones:

```
// Variables para almacenar la fecha y hora
int year, month, day, hour, minute, second; ///< Variables para fecha y hora

// Dirección de inicio en EEPROM
const int eepromStartAddr = 0; ///< Dirección inicial para almacenar fecha y hora

// Variables de tiempo
unsigned long previousMillis = 0; ///< Almacena el tiempo anterior
const long interval = 1000; ///< Intervalo de un segundo


/**
 * @brief Verifica si la EEPROM está vacía.
 *
 * @return true si la EEPROM está vacía, false en caso contrario.
 */
bool isEEPROMEmpty() {
    byte firstByte = EEPROM.read(0); // Leer el primer byte de la EEPROM
    return (firstByte == 255 || firstByte == 0xFF); // Verificar si está vacía
}

/**
 * @brief Borra el contenido de la EEPROM.
 *
 * Esta función establece todos los bytes de la EEPROM en 0xFF.
 */
void deleteEEPROM() {
    serialDebug("Borrando EEPROM...");
    int eepromSize = EEPROM.length(); // Obtener el tamaño total de la EEPROM
    for (int i = 0; i < eepromSize; i++) {
        EEPROM.write(i, 0xFF); // Borrar cada byte
    }
    serialDebug("EEPROM borrada.");
}

/**
 * @brief Actualiza la EEPROM con la fecha y hora actuales.
 */
void actualizarEEPROM() {
    EEPROM.put(eepromStartAddr, year);
    EEPROM.put(eepromStartAddr + sizeof(year), month);
    EEPROM.put(eepromStartAddr + sizeof(year) + sizeof(month), day);
    EEPROM.put(eepromStartAddr + sizeof(year) + sizeof(month) + sizeof(day), hour);
    EEPROM.put(eepromStartAddr + sizeof(year) + sizeof(month) + sizeof(day) + sizeof(hour), minute);
    EEPROM.put(eepromStartAddr + sizeof(year) + sizeof(month) + sizeof(day) + sizeof(hour) + sizeof(minute),
second);
}

/**
 * @brief Obtiene la fecha y hora desde la EEPROM.
 */
void obtenerEEPROM() {
    EEPROM.get(eepromStartAddr, year);
    EEPROM.get(eepromStatAddr + sizeof(year) + sizeof(month), day);
    EEPROM.get(eepromStartAddr + sizeof(year) + sizeof(month) + sizeof(day), hour);
    EEPROM.get(eepromStartAddr + sizeof(year) + sizeof(month) + sizeof(day) + sizeof(hour), minute);
    EEPROM.get(eepromStartAddr + sizeof(year) + sizeof(month) + sizeof(day) + sizeof(hour) + sizeof(minute),
second);
}

```

Figura 43. Funciones manejo de fecha en EEPROM. Creación propia.

Conjuntamente a dichas funciones, el manejo del tiempo en el programa avanza con el reloj interno del dispositivo, el cual cuando acaba el envío de un paquete actualiza la hora manualmente con un cálculo del tiempo interno.

```cpp
/**
 * @brief Muestra la fecha y hora actuales por el puerto serial.
 */
void mostrarFecha() {
    Serial.print("Fecha y Hora: ");
    Serial.print(year);
    Serial.print("/");
    Serial.print(month);
    Serial.print("/");
    Serial.print(day);
    Serial.print(" ");
    Serial.print(hour);
    Serial.print(":");
    Serial.print(minute);
    Serial.print(":");
    Serial.println(second);
}

/**
 * @brief Incrementa la fecha y hora actual cada segundo.
 *
 * Esta función también actualiza la EEPROM con la nueva fecha y hora.
 */
void aumentarFecha() {
    unsigned long currentMillis = millis(); // Obtener el tiempo actual
    if (currentMillis - previousMillis >= interval) {
        second += (currentMillis - previousMillis) / 1000; // Incrementar segundos
        previousMillis = currentMillis;

        minute += second / 60;
        second = second % 60;
        hour += minute / 60;
        minute = minute % 60;
        day += hour / 24;
        hour = hour % 24;

        // Manejo de los meses
        if (month == 1 || month == 3 || month == 5 || month == 7 ||
            month == 8 || month == 10 || month == 12) {
            month += day / 32;
            day = day % 32;
        } else if (month == 2) {
            month += day / 29;
            day = day % 29;
        } else {
            month += day / 31;
            day = day % 31;
        }

        // Ajuste de año
        year += month / 13;
        month = month % 13;

        // Corrección si los valores de mes o día son cero
        month = (month == 0) ? month + 1 : month;
        day = (day == 0) ? day + 1 : day;

        mostrarFecha(); // Mostrar la nueva fecha
        actualizarEEPROM(); // Actualizar la EEPROM
    }
}

```

Figura 44. Funciones manejo de fecha y hora. Creación propia.

SendData()

Por último para finalizar el programa se debe usar la función creada sendData, la cual trás la conexión con el servidor/local creada manda el array de bytes creado:

```
/**
 * @brief Envía datos a todos los servidores.
 *
 * @param d Datos a enviar.
 * @param l Longitud de los datos.
 */
void sendData(char d[], unsigned short l) {
    int arrayIP[4] = SERVERADDRIP;
    int port = SERVERPORT;
    serialDebug("SERVIDOR A CONECTAR:");
    serialDebug(String(arrayIP[0]) + "." + String(arrayIP[1]) + "." + String(arrayIP[2]) + "." +
String(arrayIP[3]) + ":" + String(port) + "\n");

    if (client.connect(SERVERADDRIP, SERVERPORT)) {
        serialDebug(F("connected\n"));
    } else {
        serialDebug(F("connection failed\n"));
    }

    if (client.connected()) {
        client.write(d, l);
        client.flush();
        client.stop();
        serialDebug(F("Heimdal OK\n"));
    } else {
        serialDebug(F("Heimdal Error\n"));
        client.stop();
        delay(1000);
    }
}
```

Figura 45. Función sendData. Creación propia.

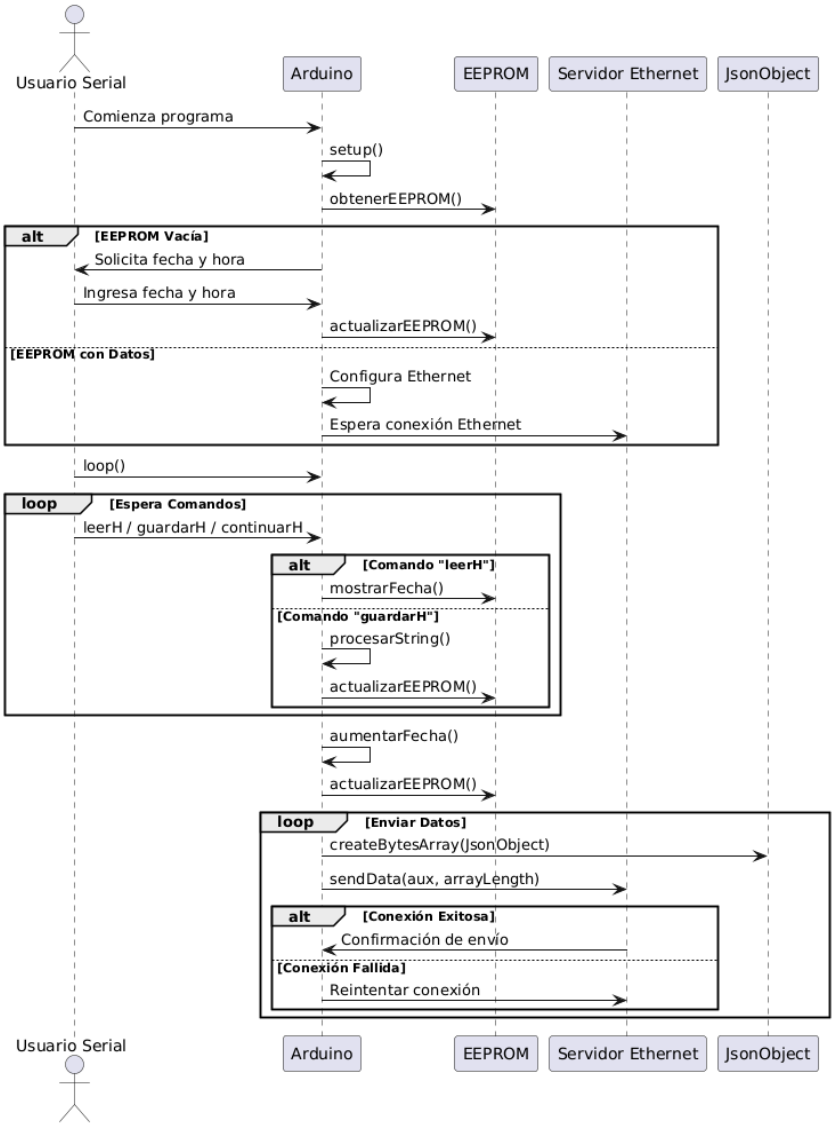


Figura 46. Diagrama de flujo completo . Creación propia.

7.2.7 Error EEPROM

Al ejecutar el código, observé que al obtener la fecha almacenada en la EEPROM, los valores recuperados no corresponden con los datos de fecha y hora que previamente había guardado. Tras revisar el código, identifiqué que el problema estaba en la función obtenerEEPROM. La lectura de los datos de la EEPROM no respetaba la misma estructura de memoria utilizada en la función actualizarEEPROM, específicamente en la posición del campo day. En lugar de calcular correctamente el desplazamiento para day (que debía considerar el tamaño de year y month), el código leía desde una posición incorrecta, lo cual provocaba un error en los valores obtenidos. Corrigiendo esta dirección en obtenerEEPROM y alineándose con la lógica de actualizarEEPROM, logré que los datos de fecha y hora se almacenaran y recuperaran de forma coherente.

```
/**
 * @brief Obtiene la fecha y hora desde la EEPROM.
 */
void obtenerEEPROM() {
    EEPROM.get(eepromStartAddr, year);
    EEPROM.get(eepromStartAddr + sizeof(year), month);
    EEPROM.get(eepromStartAddr + sizeof(year) + sizeof(month), day);
    EEPROM.get(eepromStartAddr + sizeof(year) + sizeof(month) + sizeof(day), hour);
    EEPROM.get(eepromStartAddr + sizeof(year) + sizeof(month) + sizeof(day) + sizeof(hour), minute);
    EEPROM.get(eepromStartAddr + sizeof(year) + sizeof(month) + sizeof(day) + sizeof(hour) + sizeof(minute),
second);
}

```

Figura 47. Solución problema EEPROM. Creación propia.

7.2.8 Funcionamiento.

Para poner en marcha el Arduino, conéctalo a la red Ethernet y a un puerto que permita enviar los datos al servidor de Heimdal, en este caso a mi servidor de pruebas en local. Luego, conecta el Arduino a una fuente de alimentación o por el puerto serie para poder utilizar el menú de fecha y hora, el programa esperará un intervalo predefinido de 5 minutos antes de comenzar a enviar los primeros paquetes de datos al servidor.

Dado que la hora almacenada en la EEPROM de Arduino puede no ser precisa, es recomendable configurarla manualmente utilizando los siguientes comandos a través de un monitor serie:

Comandos de configuración en el monitor serie

1. **Obtener la hora actual**: Muestra la hora almacenada en la EEPROM de Arduino.
    - Comando: **leerH**
2. **Configurar la hora manualmente**: Permite guardar una fecha y hora específica en el Arduino. Utiliza el siguiente formato para el comando:
    - Comando: **guardarH** YYYY-MM-DD HH:MM:SS
    - Ejemplo: guardarH 2024-11-10 15:30:00
3. **Enviar datos de inmediato**: Ejecuta un envío de datos inmediato al servidor sin esperar el intervalo predeterminado.
    - Comando: **continuarH**

La siguiente figura 48, muestra los mensajes de envío por el puerto serie luego de activar el arduino y pasarle por su menú la cadera “guardarH 2024-11-10 11:30:00” tal y como se ve representada arriba.

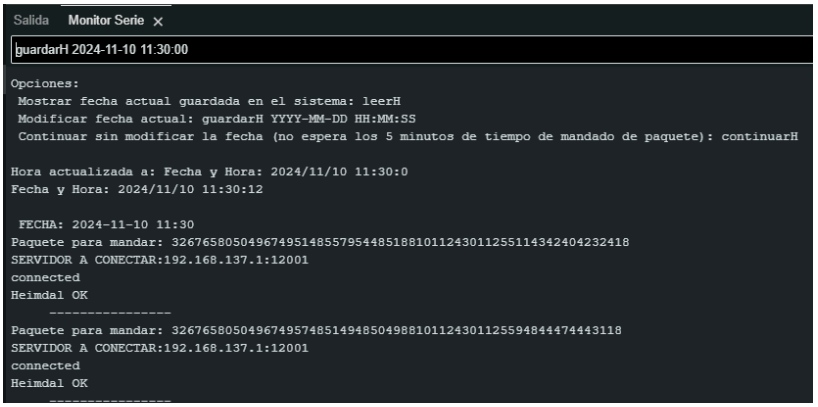


Figura 48. Ejecución programa arduino v1. Creación propia.

También se puede ver en la figura 49, como desde la aplicación en el entorno de pruebas (local), como un dispositivo ficticio (test) recibe los datos enviados. La representación es corta y debido a que se ha usado la opción del menú “continuarH” hay paquetes que se han recibido a la misma hora o muy cercanas entre sí.

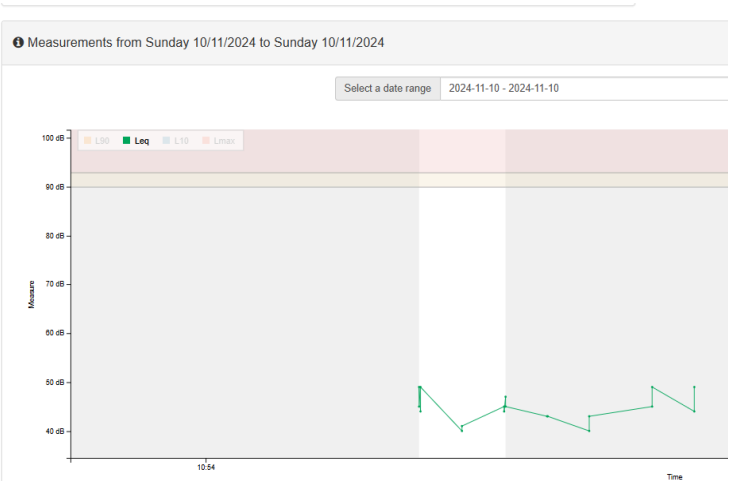


Figura 49. Ejecución programa arduino v1 App. Creación propia.

7.2.9 Arduino Versión NTP Y Router

En esta etapa final, se desarrollará una segunda versión del programa Arduino para mejorar la precisión temporal, obteniendo la hora directamente de un servidor NTP configurado para devolver la hora europea, eliminando la dependencia de la EEPROM. Además, se documentará la configuración para conectar el Arduino a la red a través de un router, facilitando su comunicación con el servidor desde redes externas. Se incluirán diagramas de secuencia y flujo que muestran cómo el programa interactúa con el servidor NTP, ajustando la hora a la hora española, considerando las diferencias horarias y cambios de horario de verano.

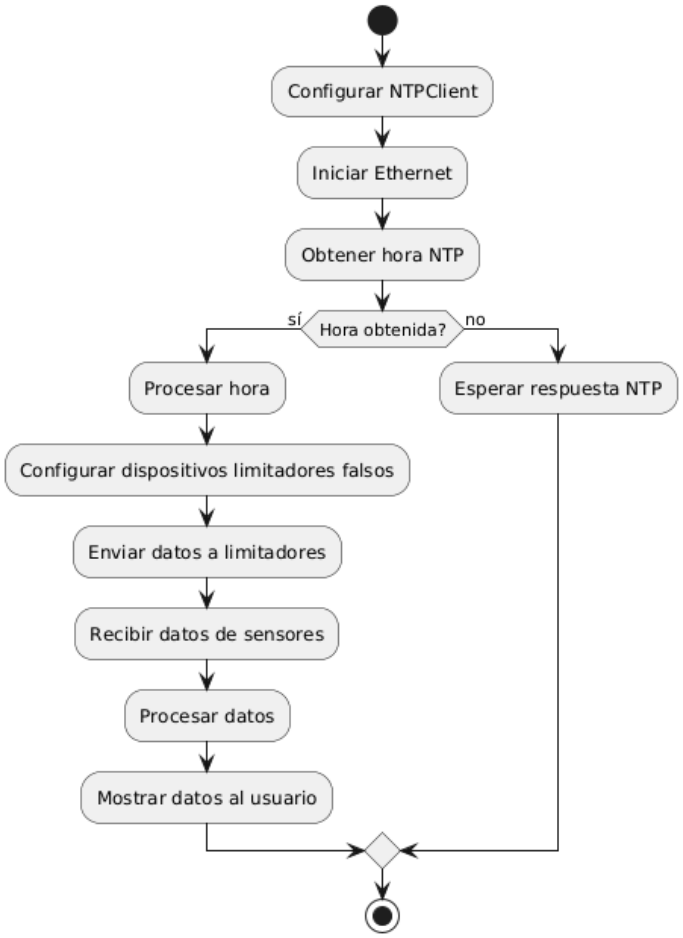


Figura 50. Diagrama de flujo Arduino V2. Creación propia.

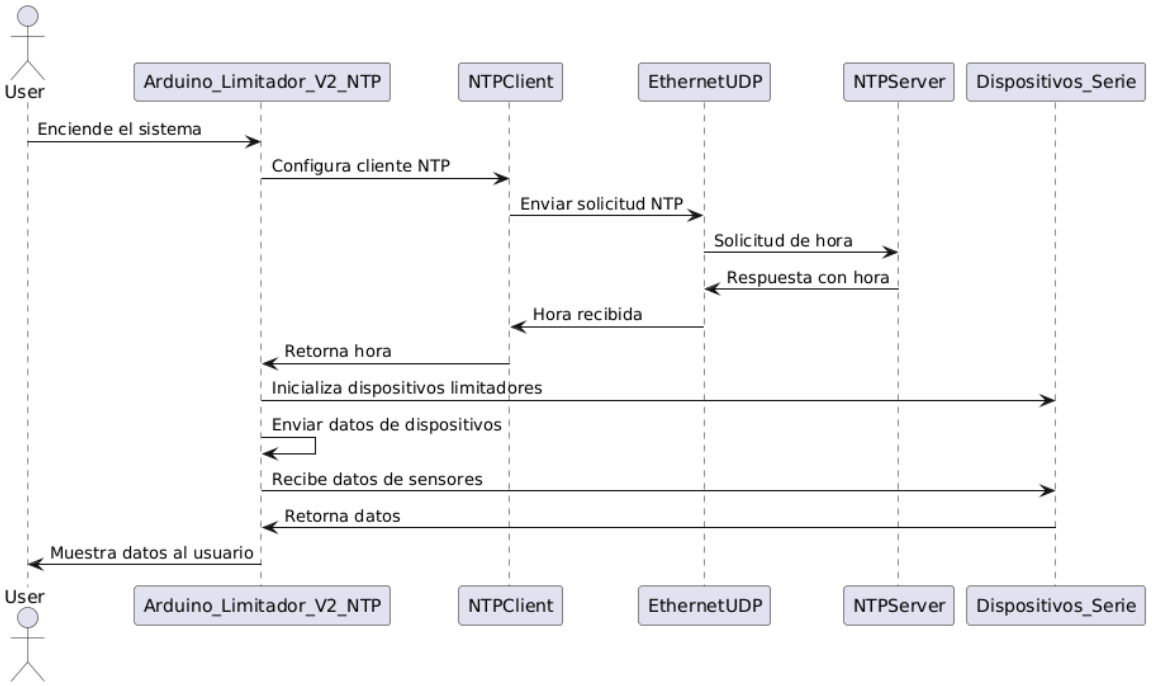


Figura 51. Diagrama de secuencia Arduino V2. Creación propia.

En esta versión del proyecto, se ha mejorado la calidad de los datos enviados al usar un array de estructuras en lugar de un simple array de dispositivos. Cada estructura incluye campos como el número de serie, el límite de sonido, la hora de apertura y cierre del local, y el número de sesión. Esta modificación mejora la organización y consistencia de los datos, asegurando que la información relevante se almacene correctamente en la base de datos, facilitando futuros análisis y gestión.

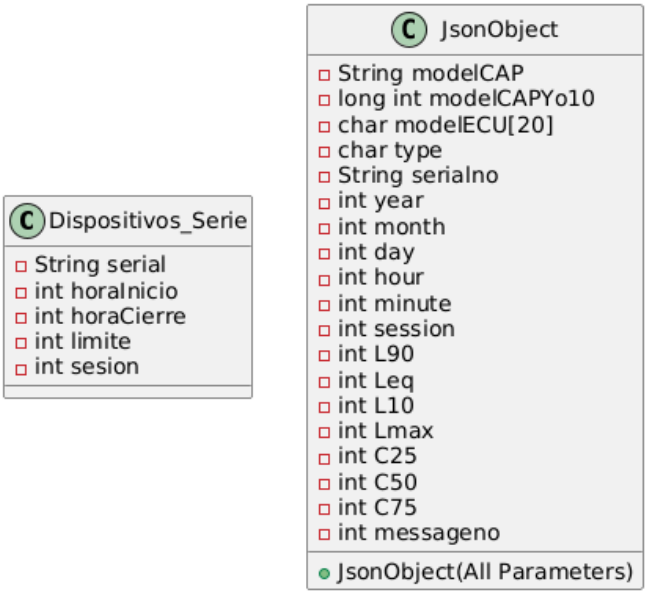


Figura 52. Diagrama de clases Arduino V2. Creación propia.

Control de fecha y hora

En el código mostrado en la figura 53, la hora se obtiene mediante una solicitud NTP a un servidor de hora, específicamente pool.ntp.org. Esta hora se actualiza cada minuto para garantizar que los datos reflejan el tiempo real. Una vez obtenida la hora, se ajusta para la zona horaria española. El ajuste se realiza calculando si corresponde al horario de verano o invierno, ya que el servidor NTP proporciona la hora en formato UTC. Dependiendo de si es verano o invierno, se añade 1 o 2 horas, respectivamente, para obtener la hora local de España. Este cálculo asegura que los datos reflejen correctamente las variaciones horarias de acuerdo con la normativa vigente en España.

```
1 // Puerto UDP y cliente NTP
2 EthernetUDP udp; ///< Cliente UDP para conexión NTP
3 NTPClient timeClient(udp, "pool.ntp.org", 0, 60000);
  ///< Cliente NTP para obtener la hora
```

Figura 54. NTPClient. Creación propia.

```ino
/**
 * @brief Comprueba si la fecha actual está en horario de verano.
 *
 * Esta función determina si la fecha actual está en el horario de verano según las reglas aplicables en España.
 * @param epochTime Tiempo en formato epoch (segundos desde 1970).
 * @return true Si es horario de verano.
 * @return false Si no es horario de verano.
 */
bool isSummerTime(unsigned long epochTime) {
  int month1 = month(epochTime); // Obtiene el mes actual
  int day1 = day(epochTime);     // Obtiene el día actual
  int year1 = year(epochTime);   // Obtiene el año actual

  // Determina si estamos en horario de verano (último domingo de marzo hasta el último domingo de octubre)
  if (month1 > 3 && month1 < 10) {
    return true; // Entre abril y septiembre es siempre horario de verano
  }
  if (month1 == 3) {
    return (day1 >= lastSundayOfMarch(year1)); // Comprobar si el último domingo de marzo ha pasado
  }
  if (month1 == 10) {
    return (day1 < lastSundayOfOctober(year1)); // Comprobar si el último domingo de octubre no ha pasado
  }
  return false; // Fuera de rango de horario de verano
}


/**
 * @brief Calcula el último domingo de marzo.
 *
 * @param year1 Año actual.
 * @return int Día del último domingo de marzo.
 */
int lastSundayOfMarch(int year1) {
  int dayOfLastSunday = 31;
  int month_number = 3;
  while (dayOfTheWeek(year1, month_number, dayOfLastSunday) != 0) { // 0 es domingo
    dayOfLastSunday--;
  }
  return dayOfLastSunday;
}

/**
 * @brief Calcula el último domingo de octubre.
 *
 * @param year1 Año actual.
 * @return int Día del último domingo de octubre.
 */
int lastSundayOfOctober(int year1) {
  int dayOfLastSunday = 31;
  int month_number = 10;
  while (dayOfTheWeek(year1, month_number, dayOfLastSunday) != 0) { // 0 es domingo
    dayOfLastSunday--;
  }
  return dayOfLastSunday;
}

/**
 * @brief Calcula el día de la semana dado un año, mes y día.
 *
 * Esta función devuelve el día de la semana donde 0 es domingo, 1 es lunes, etc.
 *
 * @param year1 Año.
 * @param month_number Mes.
 * @param day Día.
 * @return int Día de la semana.
 */
int dayOfTheWeek(int year1, int month1, int day1) {
  // Algoritmo de Zeller
  if (month1 < 3) {
    month1 += 12;
    year1--;
  }
  int K = year1 % 100;
  int J = year1 / 100;
  int f = day1 + (13 * (month1 + 1)) / 5 + K + (K / 4) + (J / 4) - 2 * J;
  return (f % 7 + 7) % 7; // 0 es sábado, 1 es domingo, ..., 6 es viernes
}

```

Figura 53. Control de fecha y hora Arduino V2. Creación propia.

Realismo datos de sonido

Para aumentar el realismo de los datos que recibe el programa de Arduino, se ha optado por incorporar los horarios de cierre de cada local. Como se mencionó anteriormente, el comportamiento del ruido se ajusta en función de estos horarios. Esto significa que, durante el horario de apertura, los niveles de ruido se simulan en valores más altos, mientras que, cuando el local está cerrado, los valores de ruido disminuyen o incluso se estabilizan, simulando la falta de actividad en el lugar. Esta dinámica mejora la precisión de los datos simulados, haciendo que los valores sean más representativos de situaciones reales, donde el ruido varía dependiendo de la hora del día y las actividades en los locales.

Como se puede ver en la figura 55, el bucle recorre todos los dispositivos simulados (limitadores) y, en función de la hora actual, calcula si el local está abierto o cerrado. Este cálculo se realiza comparando la hora con los horarios de inicio y cierre de cada local. En caso de que el local esté abierto, se aplica un multiplicador horario de 2, lo que incrementa los valores del ruido para simular un ambiente más ruidoso. Por el contrario, cuando el local está cerrado, el multiplicador se mantiene en 1, resultando en valores de ruido más bajos. Además, para aumentar la variabilidad, los valores del ruido iteran entre un rango de volumen bajo y volumen alto, acercándose al límite establecido, lo que permite simular un comportamiento realista en función de la actividad del local.

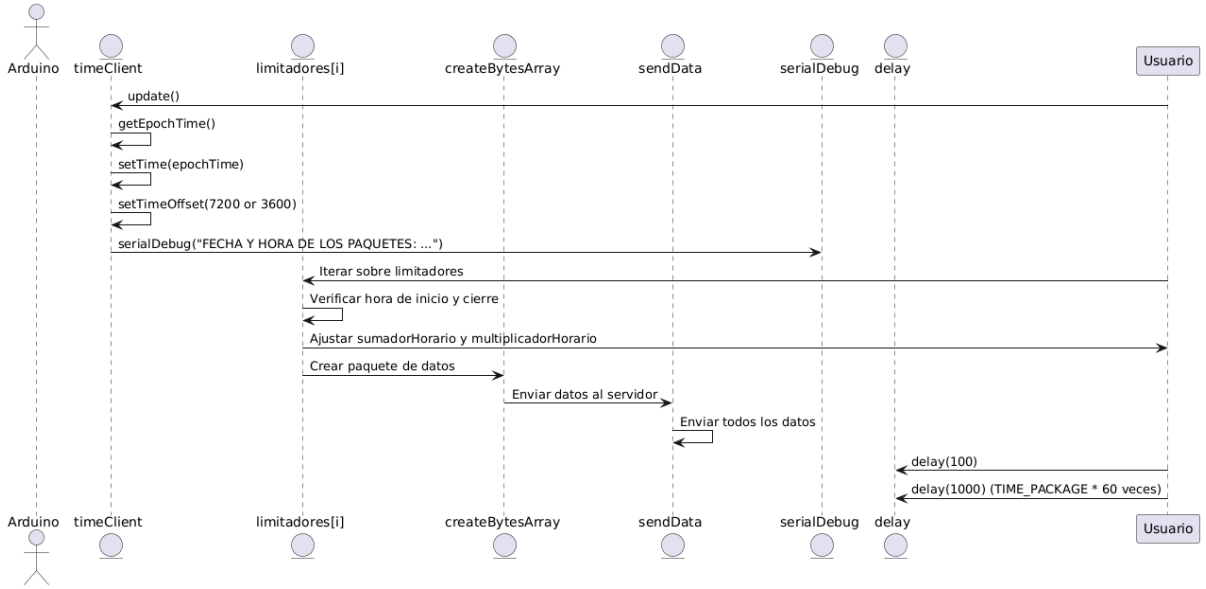


Figura 56. Diagrama de secuencia loop Arduino V2. Creación propia.

```
for (int i = 0; i < TAMSERIAL; i++) {

    int sumadorHorario = 30;
    int multiplicadorHorario = 1;
    if(hour() >= limitadores[i].horaInicio && hour() <
limitadores[i].horaCierre){
      multiplicadorHorario = 2;
      if(random() % 20 == 1)
        sumadorHorario = 80;
      else
        sumadorHorario = 70;
    }

    // Crear array de bytes para enviar el paquete de datos
    createBytesArray(JsonObject(
        " CAP21", // modelCAP
        010101010001, // _modelCAPYo10
        "", // _modelECU
        'X', // type
        limitadores[i].serial, // serialno
        year() % 100, // year
        month(), // month
        day(), // day
        hour(), // hour
        minute(), // minute
        limitadores[i].sesion, // session
        sumadorHorario + (random() % 5 * multiplicadorHorario), // L90
        sumadorHorario + (random() % 5 * multiplicadorHorario), // Leq
        sumadorHorario + (random() % 5 * multiplicadorHorario), // L10
        sumadorHorario + (random() % 5 * multiplicadorHorario), // Lmax
        sumadorHorario + (random() % 3 * multiplicadorHorario), // C25
        sumadorHorario + (random() % 3 * multiplicadorHorario), // C50
        sumadorHorario + (random() % 3 * multiplicadorHorario), // C75
        18 // messageno
    ));
```

Figura 55. Creación paquete Arduino V2. Creación propia.

```
/// Definición de los dispositivos limitadores
const int TAMSERIAL = 3; ///< Tamaño del array de números de serie
Dispositivos_Serie limitadores[TAMSERIAL] = {
  {"14030409", 10, 19, 85, 65289 + (random() % 100)}, ///< Dispositivo 1
  {"33061019", 10, 19, 85, 65289 + (random() % 100)}, ///< Dispositivo 2
  {"14070106", 10, 19, 85, 65289 + (random() % 100)}, ///< Dispositivo 3
  //{"33061019", 10, 19, 85, 65289 + (random() % 100)}  ///< Dispositivo 4
};

// Puerto UDP y cliente NTP
EthernetUDP udp; ///< Cliente UDP para conexión NTP
NTPClient timeClient(udp, "129.6.15.28", 0, 60000); ///< Cliente NTP para obtener la hora En caso de fallo cambien ip
```

Figura 57. dispositivos y configuración ntp. Creación propia.

Por último para la versión de pruebas se han elegido para el ejemplo 3 dispositivos creados en producción anteriormente llamados test. Además para el acceso al servidor horario NTP se ha elegido la IP **"129.6.15.28"**, debido a que el servidor pool distribuido adecuado **pool.ntp.org** generaba errores al mantener el arduino con acceso desde el portátil.

Conexión desde router
El método más eficiente para conectar el dispositivo Arduino es hacerlo directamente al router mediante Ethernet. Esto proporciona una conexión estable, permitiendo al Arduino acceder a servidores externos, como los de NTP, sin complicaciones. Solo es necesario ajustar la dirección **gateway** en el archivo **config.h** con la IP del router y seleccionar una IP libre dentro del rango de la red para el Arduino, optimizando así la conectividad y el acceso a servicios en línea.

Documentación con Doxygen
En esta versión del programa, se ha implementado Doxygen para generar documentación automática de las funciones, clases y métodos del sistema. Esto facilita la comprensión del código por futuros desarrolladores y mejora la colaboración entre el equipo. La integración de Doxygen asegura que la documentación se mantenga actualizada conforme se realicen cambios en el código, permitiendo su consulta rápida tanto para la revisión del código como para la formación de nuevos miembros.

Resultados

```
Configurando Ethernet estática
Hora obtenida desde NTP.
Dirección IP asignada: 192.168.137.115
Firm version verde v3
FECHA Y HORA DE LOS PAQUETES: 11-11-2024 12:7
SERVIDOR A CONECTAR:146.59.146.98:12000
connected
Heimdal OK
     ----------------
SERVIDOR A CONECTAR:146.59.146.98:12000
connected
Heimdal OK
     ----------------
SERVIDOR A CONECTAR:146.59.146.98:12000
connected
Heimdal OK
     ----------------
```

Figura 58. Resultados arduino v2. Creación propia.

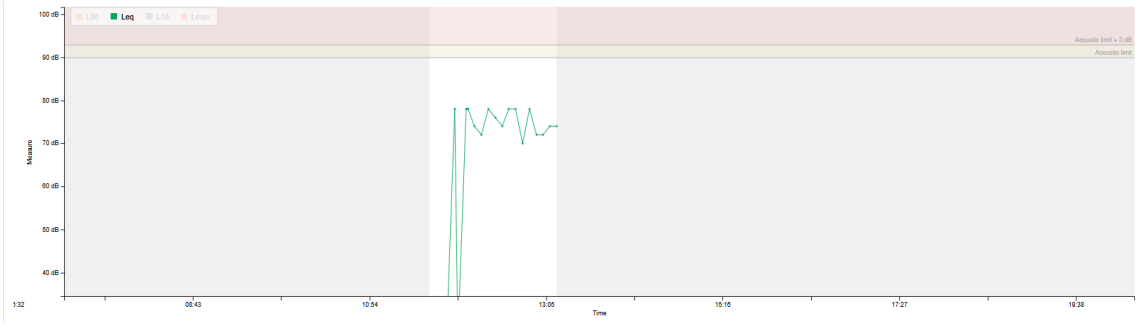


Figura 59. Resultados datos arduino v2. HeimdalSoundControl.

7.2.10 Retrospectiva

La Etapa 2 ha sido más extensa que la anterior, enfocándose en la documentación detallada de los limitadores y el análisis de los datos que generan. Esta fase incluyó el desarrollo de dos programas de Arduino para simular el envío de datos en paquetes, replicando el comportamiento de los dispositivos en el entorno de monitoreo. Se dedicó tiempo a garantizar que el flujo de datos fuera coherente y realista, implementando lógica para simular condiciones de funcionamiento según la hora del día y el estado de los locales.

Además, se utilizó Doxygen para mantener la documentación del código organizada y accesible, facilitando futuras revisiones y modificaciones. Con la documentación actualizada, esta etapa está completada y lista para su integración en el sistema.

7.3 Etapa 1: Aplicación Angular

La **Etapa 3** se enfoca en la mejora de la aplicación web en Angular junto a su api local de nodejs y mysql. Esta etapa implica realizar ajustes y mejoras tanto en el frontend como en el backend para mejorar la interacción entre ambos, asegurando un rendimiento más eficiente y una experiencia de usuario más fluida. Se busca, además, fortalecer la comunicación entre los diferentes componentes de la aplicación, implementando mejoras que faciliten su mantenimiento y escalabilidad a largo plazo.

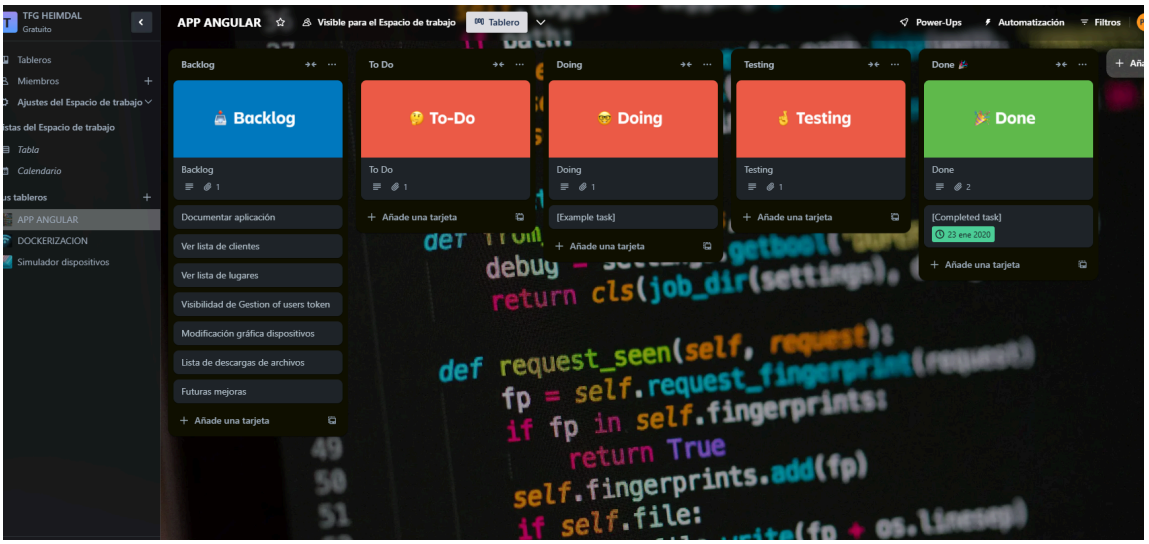


Figura 60. Tablón Kanban inicial etapa 3. Creación propia con trello.

Partiendo de las historias de usuario básicas establecidas en la planificación inicial, la **Etapa 3** se centra en la mejora de la aplicación de Angular y su integración con la API local de Node.js. Dado que las tareas en esta etapa son prácticamente aisladas y específicas, no se crearán tareas generales. En su lugar, a medida que se avance en el desarrollo de cada historia de usuario, se irán generando las tareas necesarias para abordar los aspectos técnicos y funcionales relacionados. Esto garantizará que cada

historia se implemente de manera eficaz y que se resuelvan los posibles problemas a medida que surjan, asegurando un proceso de desarrollo ágil y bien estructurado.

7.3.1 Clientes, tokens y documentación

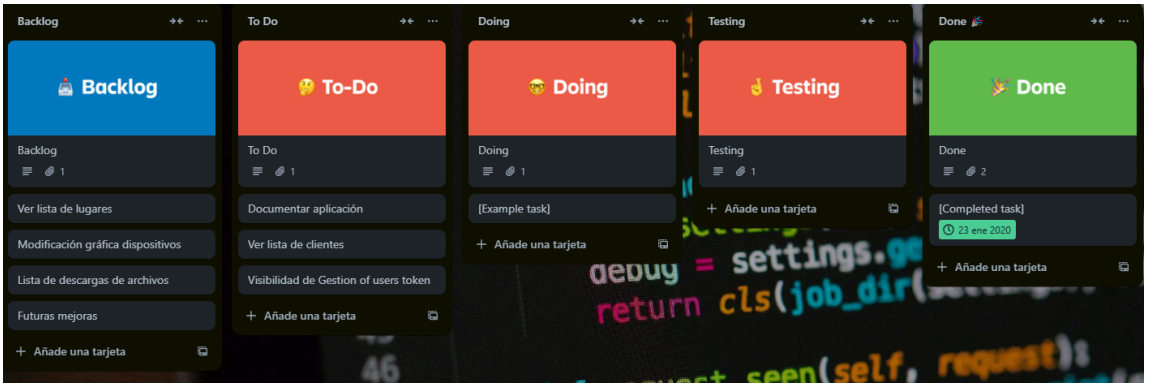


Figura 61. Tablón Kanban inicial etapa 3 parte 1. Creación propia con trello.

Documentación

La documentación de la aplicación no solo mantiene el código bien documentado para facilitar su comprensión y mantenimiento, sino que también proporciona una visión clara de la estructura del proyecto. Se detalla la organización de las carpetas y los módulos del frontend, explicando la funcionalidad de cada uno. Esto permite a los desarrolladores y colaboradores futuros comprender cómo está estructurado el proyecto.

La estructura de carpetas se describe en el punto 3 de este documento, donde se ofrece un análisis del estado inicial del proyecto. Se destacan los principales directorios y archivos, explicando la jerarquía del frontend y las relaciones entre los componentes. Esta información es esencial para trabajar de manera eficiente y ampliar la aplicación en el futuro.

UpdateData()

La comunicación entre la api local y el frontend respecto a la obtención de datos se hace en el archivo principal **app.js,** donde tiene una función la cual actualiza los datos haciendo las **peticiones get** a la api de todos los campos necesarios en función del tipo de usuario, siendo esta una función **updateData()** dentro del factory de la aplicación Angular que será llamada cada vez que haya una interacción del usuario o una modificación en esta.

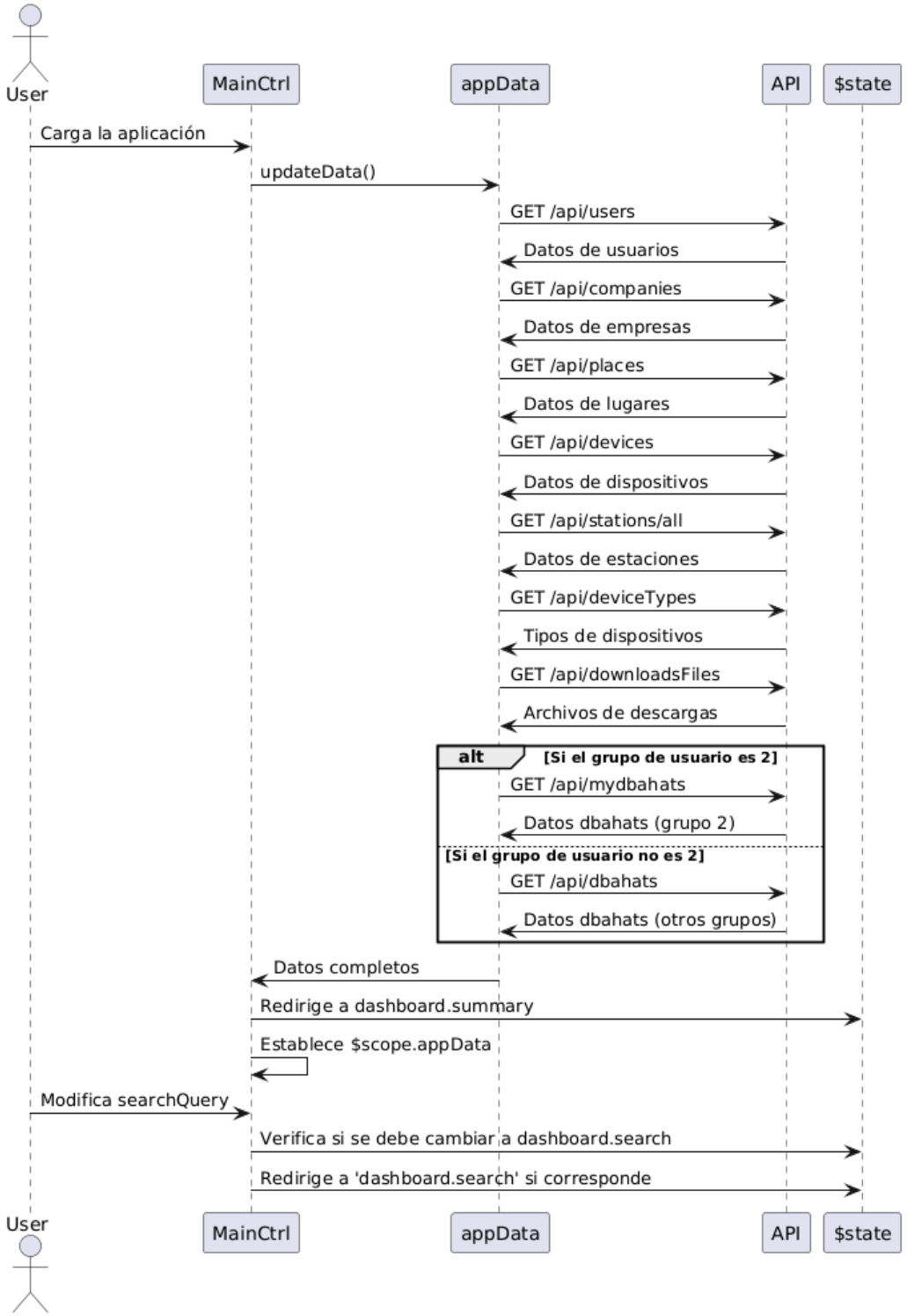


Figura 62. Diagrama flujo UpdateData() app.js. Creación propia con trello

Lista de clientes

Para la creación de la vista de lista de clientes, el enfoque en el backend se mantuvo igual, ya que el sistema ya contaba con las funciones necesarias para obtener la información de los clientes. No fue necesario realizar modificaciones significativas en el **backend**, dado que ya se disponía de los endpoints que proporcionan los datos requeridos.

En el **frontend**, se realizaron modificaciones específicas para integrar la vista de lista de clientes. Primero, se actualizó el sidebar para añadir un acceso directo a esta nueva vista, permitiendo a los usuarios navegar fácilmente hacia ella. Además, se creó un controlador que se encarga de manejar la lógica de la vista y de obtener y mostrar la lista de clientes en la interfaz. Este controlador se comunica con el backend para obtener la información de los clientes y la presenta en una tabla o formato similar, proporcionando una visualización clara y accesible de todos los registros.

```
1        $stateProvider.state('dashboard.clients', {
2            url: '/clients',
3            controller: 'ClientsCtrl as c',
4            templateUrl: 'private/views/clients.html'
5        })
```

Figura 63. configuración controlador clients.js. Creación propia.

Una vez, creado la ruta de configuración para el controlador de clientes lo añadimos al sidebar y posteriormente creamos clients.js y clients.html

```
1 <!-- CLIENTS -->
2 <li ui-sref-active="active" ng-if="appData.userInfo.user_group != 2 ">
3     <a ui-sref="dashboard.clients"><i class="fa fa-user"></i> Clients</a>
4 </li>
```

Figura 64. sidebar clients.js. Creación propia.

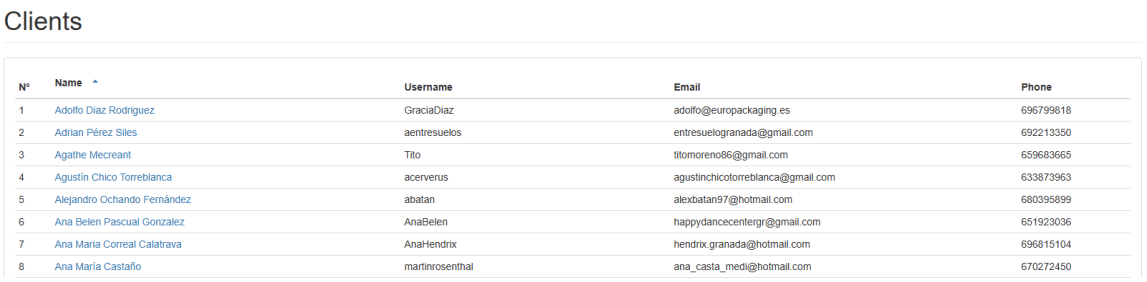
```javascript
angular.module('heimdalApp')
    .controller('ClientsCtrl', function ($scope, appData, $http) {
        var t = this;
        t.loadData = function () {
            // Verifica que los datos se carguen correctamente
            t.users = _($scope.appData.users)
                .filter({ user_group: 2 })   // Filtra por user_group
                .filter(user => user.installer > 0) // Filtra por installer no null
                .value();



        }

        if ($scope.appData && $scope.appData.users) {
            t.loadData();
        } else {
            appData.updateData().then(function (data) {
                $scope.appData = data;
                t.loadData();
            });
        }
    });
```

Figura 65. clients.js. Creación propia.

```html
<div>
    <div class="row">
        <div class="col-lg-12">
            <h1 class="page-header">Clients</h1>  <!-- Título de la página 'Clients' -->
        </div>
    </div>

    <div class="row">
        <div class="col-md-12">
            <!-- Panel que contiene la tabla de clientes -->
            <div class="panel panel-default">
                <div class="panel-body">
                    <!-- Tabla que muestra la lista de clientes filtrados -->
                    <table class="table table-hover table-condensed">
                        <thead>
                            <tr>
                                <th>Nº</th>       <!-- Columna para el número del cliente -->
                                <th>Name</th>     <!-- Columna para el nombre del cliente -->
                                <th>Username</th> <!-- Columna para el nombre de usuario -->
                                <th>Email</th>    <!-- Columna para el correo electrónico -->
                                <th>Phone</th>    <!-- Columna para el teléfono -->
                            </tr>
                        </thead>
                        <tbody>
                            <!-- Itera sobre la lista de usuarios (filtrados por ng-repeat), y para cada usuario
muestra los datos -->
                            <tr ng-repeat="p in c.users" ui-sref="dashboard.user({id: p.id})" style="cursor:
pointer;">
                                <td>{{ $index + 1}}</td>  <!-- Muestra el número de fila, usando el índice
incrementado -->
                                <td><a ui-sref="dashboard.user({id: p.id})">{{ p.firstname + ' ' + p.lastname }}</a>
</td>  <!-- Muestra el nombre completo del usuario y enlaza al dashboard -->
                                <td>{{ p.username }}</td> <!-- Muestra el nombre de usuario -->
                                <td>{{ p.email }}</td>    <!-- Muestra el email -->
                                <td>{{ p.phone }}</td>    <!-- Muestra el teléfono -->
                            </tr>
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>
</div>
```

Figura 66. clients.html. Creación propia.

El funcionamiento de la vista de **clients.js** es bastante sencillo. En esta vista, se cargan los datos necesarios en el **$scope** y, a continuación, se verifica si el usuario tiene privilegios de administrador o superior. Si es así, se muestra una lista filtrada de usuarios, limitándose a aquellos que son instaladores. Cada entrada de la lista muestra información clave como el nombre, usuario, teléfono y correo electrónico de los instaladores. Además, permite que al hacer clic sobre un usuario, se dirija a la pestaña correspondiente de ese cliente, facilitando así el acceso a más detalles.


Clients

Nº	Name	Username	Email	Phone
1	Adolfo Diaz Rodriguez	GraciaDiaz	adolfo@europackaging.es	696799818
2	Adrian Pérez Siles	aentresuelos	entresuelogranada@gmail.com	692213350
3	Agathe Mecreant	Tito	titomoreno86@gmail.com	659683665
4	Agustín Chico Torreblanca	acerverus	agustinchicotorreblanca@gmail.com	633873963
5	Alejandro Ochando Fernández	abatan	alexbatan97@hotmail.com	680395899
6	Ana Belen Pascual Gonzalez	AnaBelen	happydancecentergr@gmail.com	651923036
7	Ana Maria Correal Calatrava	AnaHendrix	hendrix.granada@hotmail.com	696815104
8	Ana María Castaño	martinrosenthal	ana_casta_medi@hotmail.com	670272450

Figura 67. vista de clientes en la aplicación. HeimdalSoundControl.

Tokens

Los tokens en la web de **HeimdalSoundControl** son códigos únicos generados para autenticar a los usuarios en la aplicación móvil, permitiendo vincular a cada usuario con sus datos en la plataforma. Los administradores tienen la capacidad de generar nuevos tokens o eliminar los existentes para cada usuario desde la interfaz web. Estos tokens son proporcionados a los instaladores de dispositivos para que puedan visualizar y gestionar sus datos a través de la aplicación móvil.

Dado que la vista y el controlador de tokens ya están creados, solo será necesario modificar el archivo **sidebar.html** para añadir un acceso a esta funcionalidad, permitiendo a los administradores gestionar los tokens de manera sencilla.

```
<!-- Gestion of users tokens -->
<li ng-if="appData.userInfo.user_group == 3 || appData.userInfo.user_group == 1" ui-sref-active="active">
    <a ui-sref="dashboard.usersTokens()">
      <i class="fa fa-key"></i>
      Gestion of users tokens</a>
</li>
```

Figura 68. sidebar gestión de tokens. HeimdalSoundControl.

7.3.2 Lugares (places), ordenar clientes y gráfico dispositivos.

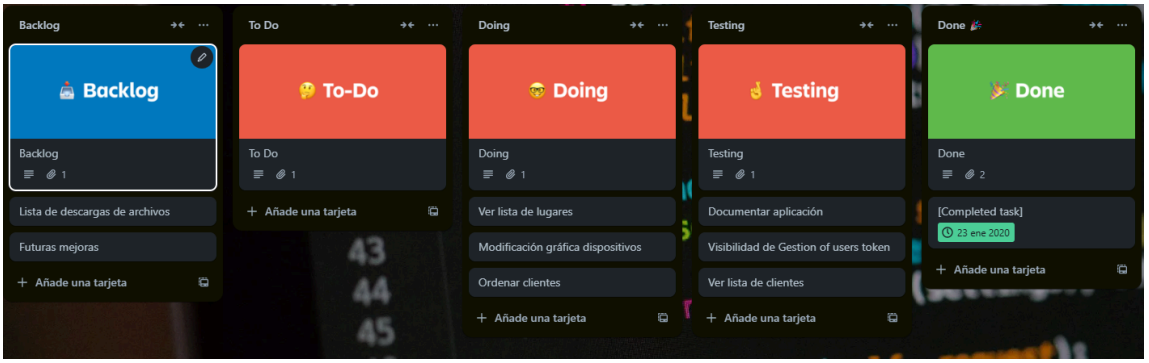


Figura 69. Tablón trello Etapa 3, parte 2. HeimdalSoundControl.

Conforme se han ido avanzando las tareas, se puede observar en el tablón Kanban que se han iniciado varias nuevas actividades. Una de ellas es la creación de una vista para la **lista de lugares**, que busca mejorar la visualización y gestión de estos elementos dentro de la aplicación.

Además, se ha trabajado en la **modificación de la gráfica de dispositivos**. El objetivo principal es hacerla más adaptada al uso de los usuarios, permitiendo que se pueda desplazar lateralmente en función de las fechas, lo que facilita su navegación y comprensión.

Por otro lado, se ha añadido **una nueva tarea**, propuesta por el desarrollador, que consiste en permitir a los usuarios **ordenar la lista de clientes**. Esta nueva funcionalidad ofrecerá mayor flexibilidad en la organización y visualización de la información.

Places

La vista de la lista de lugares (places) es muy similar a la de clientes, por lo que en esta explicación se omiten ciertos detalles y se centrará exclusivamente en la vista principal de JavaScript y HTML. La estructura y lógica subyacente son prácticamente las mismas que en la lista de clientes, con la única diferencia de que se cambia el nombre de los elementos para adaptarse a los lugares. El resto de la funcionalidad, como la carga de datos y la representación en la tabla, sigue el mismo patrón, por lo que se evitarán repeticiones innecesarias y se destacarán solo lo esencial para la implementación de la vista.

```
angular.module('heimdalApp')
    .controller('PlacesCtrl', function ($scope, appData) {
        var t = this;

        // Función que carga los datos de los lugares ('places') y reemplaza el ID del cliente por su nombre de usuario
        t.loadData = function () {
            t.places = $scope.appData.places; // Asigna los lugares (places) a 't.places' desde el objeto 'appData' cargado en el $scope

            // Carga los usuarios desde 'appData'
            var users = appData.data().users; // Extrae los usuarios de los datos proporcionados por 'appData'


            // Recorre cada lugar (place) para reemplazar el ID del cliente con su nombre de usuario
            _.each(t.places, (place) => {
                // Encuentra el usuario que corresponde al 'client' del lugar utilizando la función 'find' de lodash (_)
                var user = _.find(users, function(u) { return u.id === place.client; }); // Busca en la lista de usuarios el que tenga el mismo 'id' que 'place.client'

                if (user) { // Si se encuentra un usuario que coincida
                    place.clientName = user.firstname + ' ' + user.lastname; // Reemplaza el ID del cliente por su nombre completo
                } else {
                    // En caso de no encontrar un usuario, se podría mostrar un error (aquí está comentado)
                    // console.error("No user found with ID:", place.client);
                }
            });

        }

        // Verifica si los datos ya existen en $scope.appData
        if ($scope.appData && $scope.appData.places) {
                    t.loadData();
        } else {
            // Si no, solicita los datos a través de 'appData.updateData'
            appData.updateData().then(function (data) {
                if (data && data.places) {
                    t.loadData();

                } else {
                    console.error("Failed to load places data.");
                }
            }).catch(function (error) {
                console.error("Error while updating appData:", error);
            });
        }
    });

```

Figura 70. places.js. Creación propia.

Principalmente obtiene los lugares de los dispositivos de “places” en $scope, y sustituye el valor del usuario de esa tabla por el nombre del usuario que coincida con el mismo identificador.

```html
<div class="row">
    <div class="col-lg-12">
        <h1 class="page-header">Places</h1> <!-- Título de la página 'Places' -->
    </div>
</div>

<div class="row">
    <div class="col-md-12">
        <!-- Tabla que muestra la lista de lugares (places) -->
        <div class="panel panel-default">
            <div class="panel-body">
                <table class="table table-hover table-condensed">
                    <thead>
                        <tr>
                            <th>Nº</th> <!-- Columna para mostrar el número de lugar -->
                            <th>Name</th> <!-- Columna para mostrar el nombre del lugar -->
                            <th>Client</th> <!-- Columna para mostrar el cliente (nombre de usuario) -->
                            <th>Address</th> <!-- Columna para mostrar la dirección del lugar -->
                        </tr>
                    </thead>
                    <tbody>
                        <!-- Recorre y muestra cada lugar (place) en la tabla -->
                        <tr ng-repeat="p in c.places">
                            <td>{{ $index + 1 }}</td> <!-- Muestra el número de fila basado en el índice -->
                            <td><a ui-sref="dashboard.place({id: p.id})">{{ p.name }}</a></td> <!-- Enlace al lugar
-->
                            <td>{{ p.clientName }}</td> <!-- Muestra el nombre de usuario del cliente -->
                            <td>{{ p.address }}</td> <!-- Muestra la dirección del lugar -->
                        </tr>
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
```

Figura 71. places.html. Creación propia.

Método ordenar lista

A medida que avanzaba el proyecto, se identificó la necesidad de añadir una nueva tarea: ordenar por nombre a los clientes. Dado que el desarrollo de esta funcionalidad era similar al de otras tareas anteriores, se decidió extender esta mejora también a los lugares, permitiendo así ordenar alfabéticamente tanto los clientes como los lugares para mejorar la organización y visualización de los datos en la aplicación.

Dado que ambas vistas son parecidas el método es similar por lo que solo se muestra el proceso en clients.js.

```javascript
angular.module('heimdalApp')
    .controller('ClientsCtrl', function ($scope, appData, $http) {
        var t = this;

        // Establecer los valores iniciales
        t.sortField = 'firstname'; // Orden por nombre de forma predeterminada
        t.sortOrder = 'asc'; // Ascendente por defecto

        // Función para cargar los datos de los usuarios
        t.loadData = function () {
            // Verifica que los datos se carguen correctamente
            t.users = _($scope.appData.users)
                .filter({ user_group: 2 })   // Filtra por user_group
                .filter(user => user.installer > 0) // Filtra por installer no null
                .value();
        }

        // Función para alternar el orden por nombre
        t.sortByName = function () {
            // Si el campo ya está ordenado por 'firstname', alternamos la dirección
            if (t.sortField === 'firstname') {
                t.sortOrder = (t.sortOrder === 'asc') ? 'desc' : 'asc';
            } else {
                t.sortField = 'firstname';  // Asegura que el campo de ordenación es 'firstname'
                t.sortOrder = 'asc';        // Orden ascendente al cambiar de campo
            }
        }

        // Cargar los datos iniciales
        if ($scope.appData && $scope.appData.users) {
            t.loadData();
        } else {
            appData.updateData().then(function (data) {
                $scope.appData = data;
                t.loadData();
            });
        }
    });

```

Figura 72. ordenar por nombre clients.js. Creación propia.

```html
<th>
    Name
    <!-- Botón para ordenar por nombre -->
    <button class="btn btn-link btn-sm" ng-click="c.sortByName()">
        <i class="fa"
            ng-class="{'fa-sort-asc': c.sortOrder === 'asc', 'fa-sort-desc': c.sortOrder === 'desc'}"></i>
    </button>
</th> <!-- Columna para el nombre del cliente -->
```

Figura 73. ordenar por nombre sección clients.html. Creación propia.

Places

Nº	Name	Client	Address
1	Allbody Health & Sport Center - Alcalá de Guadaíra	Jose Manuel Bulnes Velo	Calle Mairena 56b
2	Almacenes del Pósito	Javier Espinosa Cano (Pife)	Plaza del Pósito, nº1
3	Anytime Fitness	Anytime Fitness	
4	Ave Turuta	Eduardo delMoral	C/ Milagro 5
5	Azotea de Paripé	Javier	Calle Arabial 45, Centro comercial Neptuno, Planta Primera Local 1, Granada
6	Backstage	Rafael Sánchez Ramírez	Plaza del Campillo N2 esquina C/ Moras; C.P. 18009, Granada
7	Bar Auditorio Sarao	Pablo Rivas	Calle Moras
8	Bar Boomerang	Gillian Douglas	Plaza de los tientos 3, Urb. Pueblo Blanco, 29620, Torremolinos (Málaga)
9	Bar La Isleta de la Viña	Silvano Rossignoli	Calle Corralon de los Carros, N54
10	BeOne Fitness & Sport - Rincón de la Victoria	Pablo Galvez	Camino viejo de Vélez S/N
11	Bohemia Jazz Café	Antonio Cantudo Porcel	Plaza de los Lobos, N11
12	Centro de Danza María Carricondo	María Carricondo	Av. de las Marinas 103; 04740, Roquetas de Mar, Almería
13	Discoteca Lux	Luis Gil de las Heras	Frailes 7

Figura 74. Vista lista de lugares. Creación propia.

Gráfica dispositivos

La tarea a realizar consiste en mejorar la gráfica de la vista de cada dispositivo, que muestra los niveles de ruido registrados en un periodo de tiempo determinado. Esta gráfica es fundamental para observar las variaciones de los niveles de ruido a lo largo de los días u horas, y muestra parámetros como los niveles máximos, mínimos y promedios, detallados en la Etapa 2 del proyecto.

Hasta ahora, la gráfica sólo permitía seleccionar un rango de fechas, limitando la interacción. La mejora implementada permite al usuario desplazarse lateralmente por la gráfica, facilitando la navegación a través de los datos a lo largo del tiempo. Esto mejora la experiencia de usuario, haciendo más dinámico el análisis de los niveles de ruido en diferentes días u horas.

```
// Agregar un evento de clic al gráfico
d3.select('#measureChart').on('click', function () {
    // Obtener la posición del clic
    var coords = d3.mouse(this);
    var xPosition = coords[0];

    // Obtener las fechas de inicio y fin actuales
    var currentStartDate = moment(t.date.startDate);
    var currentEndDate = moment(t.date.endDate);

    // Calcular la diferencia entre las fechas en horas
    var dateDifferenceHours = currentEndDate.diff(currentStartDate, 'hours');

    // Determinar el desplazamiento en función de la diferencia de tiempo
    var shiftHours;
    if (dateDifferenceHours > 72) { // Si el rango es más de 3 días
        shiftHours = 24; // Mueve un día
    } else if (dateDifferenceHours > 24) { // Si el rango es más de 1 día
        shiftHours = 12; // Mueve medio día
    } else {
        shiftHours = 6; // Si es menor o igual a un día, mueve 6 horas
    }

    // Si el clic está en la parte izquierda, retrocede en el tiempo
    if (xPosition < (this.getBoundingClientRect().width / 2)) {
        currentStartDate.subtract(shiftHours, 'hours');
        currentEndDate.subtract(shiftHours, 'hours');
    }
    // Si el clic está en la parte derecha y la fecha de fin está dentro de 12 días del presente
    else if (currentEndDate.isBefore(moment().add(12, 'day'))) {
        currentStartDate.add(shiftHours, 'hours');
        currentEndDate.add(shiftHours, 'hours');
    }

    // Actualiza las fechas en tu objeto t.date
    t.date.startDate = currentStartDate.format('YYYY-MM-DD HH:mm:ss');
    t.date.endDate = currentEndDate.format('YYYY-MM-DD HH:mm:ss');

    // Volver a renderizar el gráfico con las nuevas fechas
    t.updateDateGraph(); // Llama a la función que renderiza los datos del gráfico
});
```

Figura 75. Movimiento gráfico dispositivos. Creación propia.

La función está dentro del método **initGraph**, que se ejecuta al crear el gráfico. Su objetivo es agregar un evento de clic para ajustar las fechas mostradas según la posición del clic. Si el clic ocurre en el primer quinto del gráfico, retrocede el tiempo (restando horas), y si es en el último quinto, avanza el tiempo (sumando horas). Después, el gráfico se actualiza con las nuevas fechas, avanzando medio día o un día dependiendo del rango de tiempo seleccionado.

7.3.3 Sección descargas

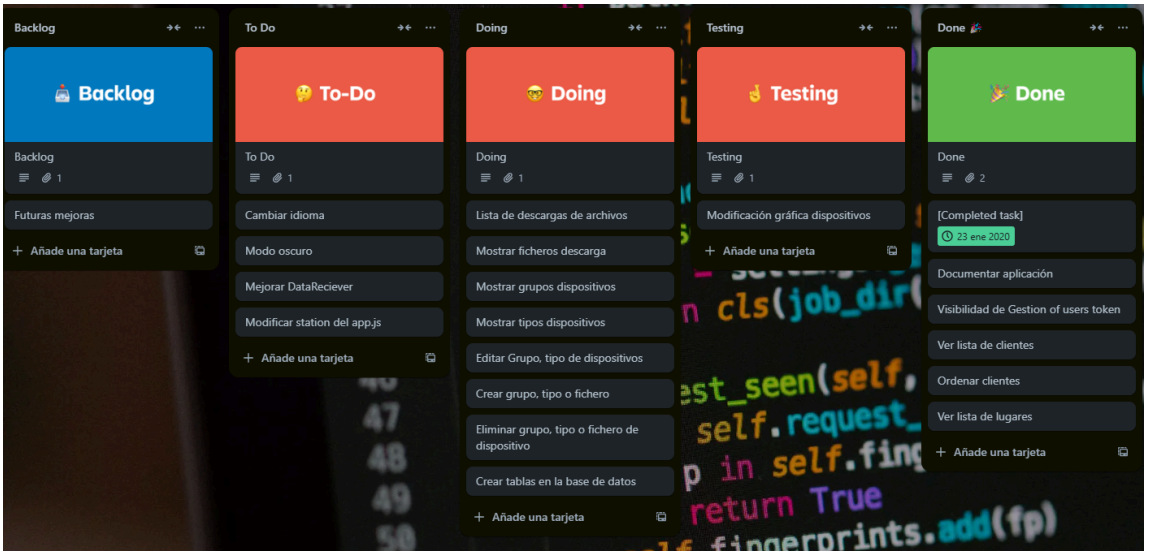


Figura 76. Tablón Kanban Etapa 3 fase final. Creado con Trello.

Habiendo completado la mayoría de las tareas a realizar se ha modificado el tablero para adaptarlo a la siguiente tarea, crear la sección de ficheros de descarga. Además se han añadido tareas a realizar a futuro.

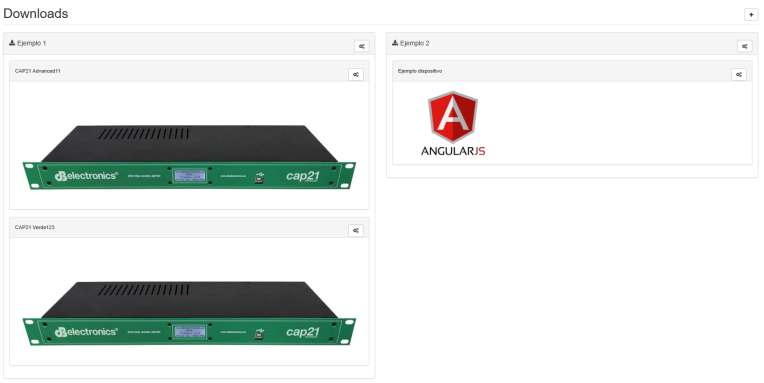


Figura 77. Diseño ejemplo vista descargas. Creación propia.

Creación de tablas mysql

Para comenzar el desarrollo se implementan nuevas tablas en la base de datos, debido a la mala gestión de los datos anteriormente se decide crear dichas tablas de manera independientes a las otras, ya que serán valores visuales y solo servirán para la ventana de descargas.

```sql
CREATE TABLE devicesTypes (
    name VARCHAR(255) PRIMARY KEY, -- Este campo debe ser único
    photo VARCHAR(255),
    group_name VARCHAR(255) NOT NULL
);

CREATE TABLE devicesDownloads (
    id INT AUTO_INCREMENT PRIMARY KEY,
    device VARCHAR(255) NOT NULL, -- Este campo almacenará el nombre del dispositivo de devicesTypes
    name VARCHAR(255),
    name_File VARCHAR(255),
    version VARCHAR(255),
    link VARCHAR(255), -- Nuevo campo para almacenar la URL
    FOREIGN KEY (device) REFERENCES devicesTypes (name) ON DELETE CASCADE ON UPDATE CASCADE -- Relación con devicesTypes
);
```

Figura 78. Creación de tablas. Creación propia.

Tabla devicesTypes

- **name**: Nombre único del tipo de dispositivo, que sirve como clave primaria.
- **photo**: Ruta o enlace de la imagen asociada al dispositivo.
- **group_name**: Nombre del grupo al que pertenece el dispositivo, obligatorio.

Tabla devicesDownloads

- **id**: Identificador único de cada registro de descarga, autoincremental y clave primaria.
- **device**: Nombre del dispositivo (referencia a devicesTypes.name), indica para qué dispositivo es la descarga.
- **name**: Nombre descriptivo de la descarga.
- **name_File**: Nombre del archivo que se descarga.
- **version**: Versión del archivo o de la descarga.
- **link**: URL donde se encuentra el archivo para su descarga.

BackEnd y API

Para esta vista se han creado peticiones a la api correspondiente a las tareas de crear, insertar o borrar en las 2 tablas:

```
1
2  /*****************************************************************/
3  /*                    QUERIES FOR DOWNLOADS FILES                */
4  /*****************************************************************/
5  /**
6   * Estas querys llaman a las tablas dónde se almacena la información
7   * de los devices. Su nombre, imagen, nombre de los documentos y manuales
8   */
9
10
11 dbManager.prototype.getDeviceTypes = function(user) {
12     return new Promise((resolve, reject) => {
13             db.query('SELECT * FROM devicesTypes', function(error, results, fields) {
14             if (error) {
15                 logger.error("Error getting device type: " + error);
16                 return reject(error); // Devuelve el error si ocurre
17             }
18                 return resolve(results)
19             })
20     })
21 }
22
23 dbManager.prototype.insertDeviceTypes = function(data) {
24     return new Promise((resolve, reject) => {
25
26         const photoValue = data.photo || null; // Asigna NULL si photo está vacío
27             db.query("INSERT INTO devicesTypes (name,photo,group_name) \
28         VALUES (?,?,?)", [data.name, photoValue, data.group_name], function(error, results, fields) {
29                 if (error) {
30                     logger.error("Error in insert device type: " + error);
31                     return reject(error); // Devuelve el error si ocurre
32                 }
33                 return resolve(results.insertId)
34             });
35     })
36 }
37
38
39 dbManager.prototype.modifyDeviceTypes = function(data) {
40     return new Promise((resolve, reject) => {
41         db.query(
42             "UPDATE devicesTypes SET name = ?, photo = ?, group_name = ? WHERE name = ?",
43             [data.name, data.photo, data.group_name, data.oldName],
44             function(error, results, fields) {
45                 if (error) {
46                     logger.error("Error in updating device type: " + error);
47                     return reject(error); // Devuelve el error si ocurre
48                 }
49                 return resolve(results.affectedRows); // Devuelve el número de filas afectadas
50             }
51         );
52     });
53 }
54
55 // Método en `dbManager` para eliminar el dispositivo
56 dbManager.prototype.deleteDeviceTypes = function(data) {
57     return new Promise((resolve, reject) => {
58         db.query("DELETE FROM devicesTypes WHERE name = ?", [data.name], function(error, results, fields) {
59             if (error) {
60                 logger.error("Error in deleting device type: " + error);
61                 return reject(error); // Devuelve el error si ocurre
62             }
63             return resolve(results.affectedRows); // Devuelve el número de filas afectadas
64         });
65     });
66 };
67
68
69
70 // Método en `dbManager` para eliminar el grupo de dispositivo
71 dbManager.prototype.deleteDeviceGroup = function(data) {
72     return new Promise((resolve, reject) => {
73         db.query("DELETE FROM devicesTypes WHERE group_name = ?", [data.group_name], function(error, results, fields) {
74             if (error) {
75                 logger.error("Error in deleting device group: " + error);
76                 return reject(error); // Devuelve el error si ocurre
77             }
78             return resolve(results.affectedRows); // Devuelve el número de filas afectadas
79         });
80     });
81 };
82
83
84 //Metodo en `dbManager` para modificar el grupo de dispositivo
85 dbM
86     });
87 }
88
89
90
91
92 dbManager.prototype.deleteDownloadsFiles = function(data) {
93     return new Promise((resolve, reject) => {
94         db.query("DELETE FROM devicesDownloads WHERE id = ?",
95                  [data.id],
96                  function(error, results, fields) {
97             if (error) {
98                 logger.error("Error in deleting device type: " + error);
99                 return reject(error); // Devuelve el error si ocurre
100            }
101            return resolve(results.affectedRows); // Devuelve el número de filas afectadas
102        });
103    });
104 };
```

Figura 79. Funciones DB devicesType. Creación propia.

```
1
2  /// <---------------------------------->
3
4  dbManager.prototype.getDownloadsFiles = function( ) {
5      return new Promise((resolve, reject) => {
6              db.query('SELECT * FROM devicesDownloads', function(error, results, fields) {
7              if (error) {
8                  logger.error("Error getting downloads files: " + error);
9                  return reject(error); // Devuelve el error si ocurre
10             }
11                 return resolve(results)
12             })
13     })
14 }
15
16
17 dbManager.prototype.insertDownloadFile = function(data) {
18     return new Promise((resolve, reject) => {
19         // Reemplaza las cadenas vacías por null
20         const device = data.device;
21         const name = data.name || " ";  // Si name está vacío, se convierte a null
22         const link = data.link || null;  // Si link está vacío, se convierte a null
23         const name_File = data.path || null; // Si name_File está vacío, se convierte a null
24         const version = data.version || " "; // Si version está vacío, se convierte a null
25
26         db.query("INSERT INTO devicesDownloads (device, name, link, name_File, version) VALUES (?, ?, ?, ?, ?)",
27             [device, name, link, name_File, version], function(error, results, fields) {
28                 if (error) {
29                     logger.error("Error in inserting downloads files: " + error);
30                     return reject(error); // Devuelve el error si ocurre
31                 }
32                 return resolve(results.insertId);
33         });
34     });
35 }
36
37
38
39
40 dbManager.prototype.deleteDownloadsFiles = function(data) {
41     return new Promise((resolve, reject) => {
42         db.query("DELETE FROM devicesDownloads WHERE id = ?",
43                 [data.id],
44                  function(error, results, fields) {
45             if (error) {
46                 logger.error("Error in deleting device type: " + error);
47                 return reject(error); // Devuelve el error si ocurre
48             }
49             return resolve(results.affectedRows); // Devuelve el número de filas afectadas
50         });
51     });
52 };
```

Figura 80. Funciones DB devicesDownloads. Creación propia.

Las funciones de acceso a la base de datos creadas para la vista de ficheros de descarga son:

- **Grupo de Dispositivo**
  - Eliminar por nombre del grupo: **deleteDeviceGroup**
  - Modificar nombre del grupo: **modifyDeviceGroup**
- **Tipo de Dispositivo**
  - Obtener todos los tipos de dispositivos: **getDeviceTypes**
  - Insertar un nuevo tipo de dispositivo: **insertDeviceTypes**
  - Modificar tipo de dispositivo por nombre: **modifyDeviceTypes**
  - Eliminar tipo de dispositivo por nombre: **deleteDeviceTypes**
- **Archivos de Descarga**
  - Obtener todos los archivos de descarga: **getDownloadsFiles**
  - Insertar un nuevo archivo de descarga: **insertDownloadFile**
  - Eliminar archivo de descarga por ID: **deleteDownloadsFiles**

La api sigue la misma estructura:

1. **Obtener todos los tipos de dispositivos** (/api/deviceTypes):
    - Método: GET que devuelve todos los tipos de dispositivos en JSON.
2. **Insertar un tipo de dispositivo** (/api/insertDeviceTypes):
    - Método: POST que recibe name y group_name. Verifica que estén presentes y, si son válidos, los inserta en la base de datos.
3. **Modificar un tipo de dispositivo** (/api/modifyDeviceTypes):
    - Método: POST que actualiza un tipo de dispositivo con los datos name y group_name, verificando su presencia.
4. **Eliminar un tipo de dispositivo** (/api/deleteDeviceTypes):
    - Método: POST que elimina un dispositivo usando name como identificador, y verifica que esté presente.

5. **Modificar el nombre de un grupo de dispositivos** (/api/modifyDeviceGroup):
    - Método: POST que actualiza el nombre de un grupo, verificando newName y oldName.
6. **Eliminar un grupo de dispositivos** (/api/deleteDeviceGroup):
    - Método: POST que elimina un grupo usando group_name, y verifica que esté presente.

7. **Obtener todos los archivos de descarga** (/api/downloadsFiles):
    - Método: GET que devuelve todos los archivos de descarga.
8. **Insertar un archivo de descarga** (/api/insertDownloadFile):
    - Método: POST que recibe device, y si está presente, lo inserta en la base de datos.
9. **Eliminar un archivo de descarga** (/api/deleteDownloadsFiles):
    - Método: POST que elimina un archivo usando id y verifica que esté presente.

```
1
2  /*****************************************************************/
3  /*                  API QUERIES FOR DOWNLOADS                    */
4  /*****************************************************************/
5  // con /api/stations/all/:user  --> muestra todo las estaciones del user enabled o no
6  // con /api/stations/all        --> muestra todos las estciones de ruido
7
8
9  router.get('/api/deviceTypes',isAuthenticated, function(req, res) {
10     dbManager.getDeviceTypes().then((response) => {
11         res.json(response);
12     }).catch((error) => {
13         res.status(500).json({
14             "error": "Error getting device types",
15             "details": error
16         });
17     });
18 });
19
20 // Router para manejar la inserción de dispositivos
21 router.post('/api/insertDeviceTypes',isAuthenticated, function(req, res) {
22     var data = req.body;
23     if (data.name && data.group_name) {
24         dbManager.insertDeviceTypes(data).then((response) => {
25             res.json({
26                 "message": "Device type inserted successfully",
27                 "insertedId": response
28             });
29         }).catch((error) => {
30             res.status(500).json({
31                 "error": "Error inserting device type",
32                 "details": error
33             });
34         });
35     } else {
36         res.status(400).json({
37             "error": "Invalid data. 'name' and 'group_name' are required"
38         });
39     }
40 });
41
42 router.post('/api/modifyDeviceTypes',isAuthenticated, function(req, res) {
43     var data = req.body;
44     if (data.name && data.group_name) {
45         dbManager.modifyDeviceTypes(data).then((response) => {
46             res.json({
47                 "message": "Device type inserted successfully",
48                 "insertedId": response
49             });
50         }).catch((error) => {
51             res.status(500).json({
52                 "error": "Error inserting device type",
53                 "details": error
54             });
55         });
56     } else {
57         res.status(400).json({
58             "error": "Invalid data. 'name' and 'group_name' are required"
59         });
60     }
61 });
62
63 // Router para manejar la eliminación de dispositivos
64 router.post('/api/deleteDeviceTypes', isAuthenticated, function(req, res) {
65     var data = req.body;
66     if (data.name) {
67         dbManager.deleteDeviceTypes(data).then((response) => {
68             res.json({
69                 "message": "Device type deleted successfully",
70                 "affectedRows": response
71             });
72         }).catch((error) => {
73             res.status(500).json({
74                 "error": "Error deleting device type",
75                 "details": error
76             });
77         });
78     } else {
79         res.status(400).json({
80             "error": "Invalid data. 'name' is required"
81         });
82     }
83 });
```

Figura 81. Funciones api devicetypes. Creación propia.

```
// Router para manejar la edicion del grupo de dispositivos
router.post('/api/modifyDeviceGroup', isAuthenticated, function(req, res) {
    var data = req.body;
    if (data.newName && data.oldName){
        dbManager.modifyDeviceGroup(data).then((response) => {
            res.json({
                "message": "Device group modify successfully",
                "insertedId": response
            });
        }).catch((error) => {
            res.status(500).json({
                "error": "Error modify device group",
                "details": error
            });
        });
    } else {
        res.status(400).json({
            "error": "Invalid data. 'group_name' is required"
        });
    }
});

// Router para manejar la eliminación de grupos
router.post('/api/deleteDeviceGroup', isAuthenticated, function(req, res) {
    var data = req.body;
    if (data.group_name) {
        dbManager.deleteDeviceGroup(data).then((response) => {
            res.json({
                "message": "Device group deleted successfully",
                "affectedRows": response
            });
        }).catch((error) => {
            res.status(500).json({
                "error": "Error deleting device group",
                "details": error
            });
        });
    } else {
        res.status(400).json({
            "error": "Invalid data. 'group_name' is required"
        });
    }
});


```

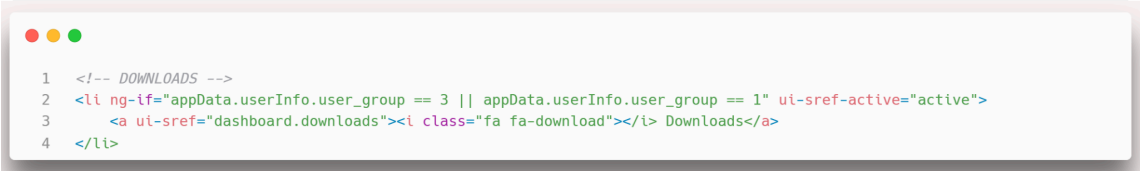
Figura 82. Funciones api deviceGroup. Creación propia.

```
//<<<<<<<<<------------------------------------------>>>>>>>>>

router.get('/api/downloadsFiles',isAuthenticated, function(req, res) {
    dbManager.getDownloadsFiles().then((response) => {
        res.json(response);
    }).catch((error) => {
        res.status(500).json({
            "error": "Error getting device downloads files",
            "details": error
        });
    });
});

router.post('/api/insertDownloadFile',isAuthenticated, function(req, res) {
    var data = req.body;
    if(data.device){
        dbManager.insertDownloadFile(data).then((response) => {
            res.json(response);
        }).catch((error) => {
            res.status(500).json({
                "error": "Error getting device downloads files",
                "details": error
            });
        });
    } else
        res.status(400).json({
            "error": "Invalid data. 'device' is required"
        });
});

router.post('/api/deleteDownloadsFiles',isAuthenticated, function(req, res) {
    var data = req.body;
    if(data.id){
        dbManager.deleteDownloadsFiles(data).then((response) => {
            res.json(response);
        }).catch((error) => {
            res.status(500).json({
                "error": "Error getting device downloads files",
                "details": error
            });
        });
    } else
        res.status(400).json({
            "error": "Invalid data. 'id' is required"
        });
});

```

Figura 83. Funciones api DownloadsFiles. Creación propia.

FrontEnd

A continuación, se implementará el frontend de acuerdo al diseño previamente establecido en el ejemplo. Este desarrollo se enfocará en integrar cada funcionalidad de la API descrita anteriormente, adaptando la interfaz de usuario para facilitar la administración de dispositivos y archivos de descarga. Se utilizarán componentes y formularios interactivos que permitan visualizar, insertar, modificar y eliminar tipos de dispositivos y grupos, asegurando una experiencia de usuario intuitiva y acorde a los objetivos del proyecto.



```
<!-- DOWNLOADS -->
<li ng-if="appData.userInfo.user_group == 3 || appData.userInfo.user_group == 1" ui-sref-active="active">
    <a ui-sref="dashboard.downloads"><i class="fa fa-download"></i> Downloads</a>
</li>
```

Figura 84. Sección sidebar downloads. Creación propia.

```
$stateProvider.state('dashboard.downloads', {
    url: '/downloads/',
    controller: 'DownloadsCtrl as c',
    templateUrl: 'private/views/downloads.html'
});

```

Figura 85. controller downloads app.js. Creación propia.

Para la vista de descargas se crea un html simple donde se muestre lo deseado enviado por el controlador, un array de grupos de documentos que tendrá la lista de tipos de dispositivos y estos a su vez, tienen la lista de documentos en su interior.

La vista de “downloads” contiene una lista de elementos, cada uno representando un grupo de dispositivos. Cada grupo tiene un encabezado con su nombre y un menú desplegable de acciones (Agregar, Editar, Eliminar) accesible solo para administradores o usuarios especiales. Dentro de cada grupo, se despliega un listado de dispositivos, cada uno mostrando su nombre, imagen, y una tabla de archivos descargables que incluye nombre, versión, enlace, y botones de acción (Descargar o Eliminar), dependiendo del rol del usuario.

Página: Descargas

- **Encabezado** con el título *'Descargas'* y un botón *"Agregar Dispositivo"* (si el usuario es administrador o usuario especial)

Para cada elemento en la lista de dispositivos:
Panel:
- Título con el **nombre del grupo de dispositivos** y un **menú desplegable** de opciones (Agregar Dispositivo, Editar Grupo, Eliminar Grupo, si el usuario es administrador o usuario especial)

Para cada dispositivo en los dispositivos del grupo:
Panel:
- Título con el **nombre del dispositivo** y un **menú desplegable** de opciones (Agregar Archivos, Editar Dispositivo, Eliminar Dispositivo, si el usuario es administrador o usuario especial)
- Mostrar **foto** (si está disponible)
- **Tabla** con los archivos de descarga:
- Columnas: **Nombre del archivo**, **enlace/versión**, **botón de descarga**, **estado**, **botón de eliminación** (si el usuario es administrador o usuario especial)

Tabla 6. Pseudocódigo vista descargas. Creación propia.

```
<h4><i class="fa fa-download"></i> {{m.name}}
    <div class="btn-group pull-right" ng-if="c.isAdminOrSpecialUser">
        <div class="dropdown">
            <button class="btn btn-default dropdown-toggle" type="button" data-toggle="dropdown" aria-haspopup="true">
                <i class="fa fa-cogs"></i>
            </button>
            <ul class="dropdown-menu dropdown-menu-right" style="text-align:center;">
                <li>
                    <a ng-click="c.createDeviceType(m.name)"> <i class="fa fa-plus"></i> Crear dispositivo</a>
                </li>
                <li>
                    <a ng-click="c.modifyDeviceGroup(m.name)"> <i class="fa fa-pencil"></i> Editar grupo</a>
                </li>
                <li>
                    <a ng-click="c.deleteGroup(m.name)"> <i class="fa fa-trash"></i> Borrar grupo</a>
                </li>

            </ul>
        </div>
    </div>
</h4>
```

Figura 86. Vista grupos dispositivos. Creación propia.

```
<h5>
    {{d.name}}

    <div class="btn-group pull-right" ng-if="c.isAdminOrSpecialUser">
        <div class="dropdown">
            <button class="btn btn-default dropdown-toggle" type="button" data-toggle="dropdown" aria-haspopup="true">
                <i class="fa fa-cogs"></i>
            </button>
            <ul class="dropdown-menu dropdown-menu-right" style="text-align:center;">
                <li>
                    <a ng-click="c.createFileDevice(d)"> <i class="fa fa-plus"></i> Añadir archivos</a>
                </li>
                <li>
                    <a ng-click="c.editDeviceType(d, m.name)"> <i class="fa fa-pencil"></i> Editar dispositivo</a>
                </li>
                <li>
                    <a ng-click="c.deleteDeviceType(d)"> <i class="fa fa-trash"></i> Borrar dispositivo</a>
                </li>

            </ul>
        </div>
    </div>

</h5>
```

Figura 87. Vista dispositivos. Creación propia.

```
<table ng-if="d.downloads.length > 0" class="table table-condensed">
    <tbody>
        <tr ng-repeat="i in d.downloads">
            <td>{{i.name}}</td>
            <td><a ng-if="i.link" href="{{i.link}}">Link </a>{{i.version}}</td>
            <td >
                <button ng-if="i.path" ng-click="c.downloadFile(i.path)" type="button" class="btn btn-default btn-circle">
                  <i class="fa fa-file"></i>
              </button>
            </td>
            <td ng-if="!i.path">
                Pending
            </td>
            <td ng-if="c.isAdminOrSpecialUser">
                <button ng-click="c.deleteFileDevice(i)" type="button" class="btn btn-default btn-circle"><i class="fa fa-trash"></i></button>
            </td>
        </tr>
    </tbody>
</table>
```

Figura 88. Vista archivos de descarga. Creación propia.

Los datos que recibe la vista son cargados desde el objeto **appData**, tal como se explicó en el punto anterior. Estos datos son enviados desde el controlador, que se encarga de parsearlos desde las dos tablas correspondientes al array adecuado. Además, el controlador organiza estos datos en función de ciertos criterios, como la presencia de imágenes y la cantidad de archivos asociados a cada dispositivo, asegurando así que se muestren de forma ordenada y relevante en la interfaz de usuario.

La Figura 89 presenta el controlador de descargas, el cual se encuentra incompleto debido a la ausencia de la función **t.loadData()**, la cual se detalla en la Figura 90. Asimismo, las funciones asociadas a los botones de edición en la vista serán mostradas en las figuras posteriores a medida que se desarrollen.

```
'use strict';

angular.module('heimdalApp').controller('DownloadsCtrl', function($scope, $uibModal, $stateParams, $http, $sce, appData, FileSaver) {
    var t = this;

    this.isAdminOrSpecialUser = function() {
        // Function logic here
    };

    t.loadData = function () {
        // Function logic here
    };

    if ($scope.appData && $scope.appData.deviceTypes && $scope.appData.downloadsFiles) {
        t.loadData();
    } else {
        // Si no, solicita los datos a través de 'appData.updateData'
        appData.updateData().then(function (data) {
            if (data && data.deviceTypes && data.downloadsFiles) {
                t.loadData();

            } else {
                console.error("Failed to load deviceTypes and downloadsFiles data.");
            }
        }).catch(function (error) {
            console.error("Error while updating appData:", error);
        });
    }

    this.downloadFile = angular.bind(this, function(id) {
        // Function logic here
    });

    this.deleteFile = function(filename) {
        // Function logic here
    };

    t.uploadFile = function(file, oldPhotoName) {
        // Function logic here
    };

    t.createDeviceType = function(groupName) {
        // Function logic here
    };

    t.editDeviceType = function(deviceType, group_name) {
        // Function logic here
    };

    t.deleteDeviceType = function (deviceType) {
        // Function logic here
    };

    t.deleteGroup = function (group_name) {
        // Function logic here
    };

    t.modifyDeviceGroup = function (group_name) {
        // Function logic here
    };

    t.createFileDevice = function(device) {
        // Function logic here
    };
});

```

Figura 88. Controlador Downloads limpio. Creación propia.

La función **t.loadData()** en el controlador de descargas organiza los datos de los tipos de dispositivos y sus archivos de descarga en una estructura jerárquica. Primero, agrupa los dispositivos por su nombre de grupo (**group_name**), luego, por cada dispositivo, filtra y ordena sus archivos de descarga (por nombre y versión). Después, se asignan estos archivos de descarga a sus dispositivos correspondientes. Finalmente, organiza los dispositivos dentro de cada grupo en función de si tienen foto y ordena los grupos según la cantidad de dispositivos con foto.

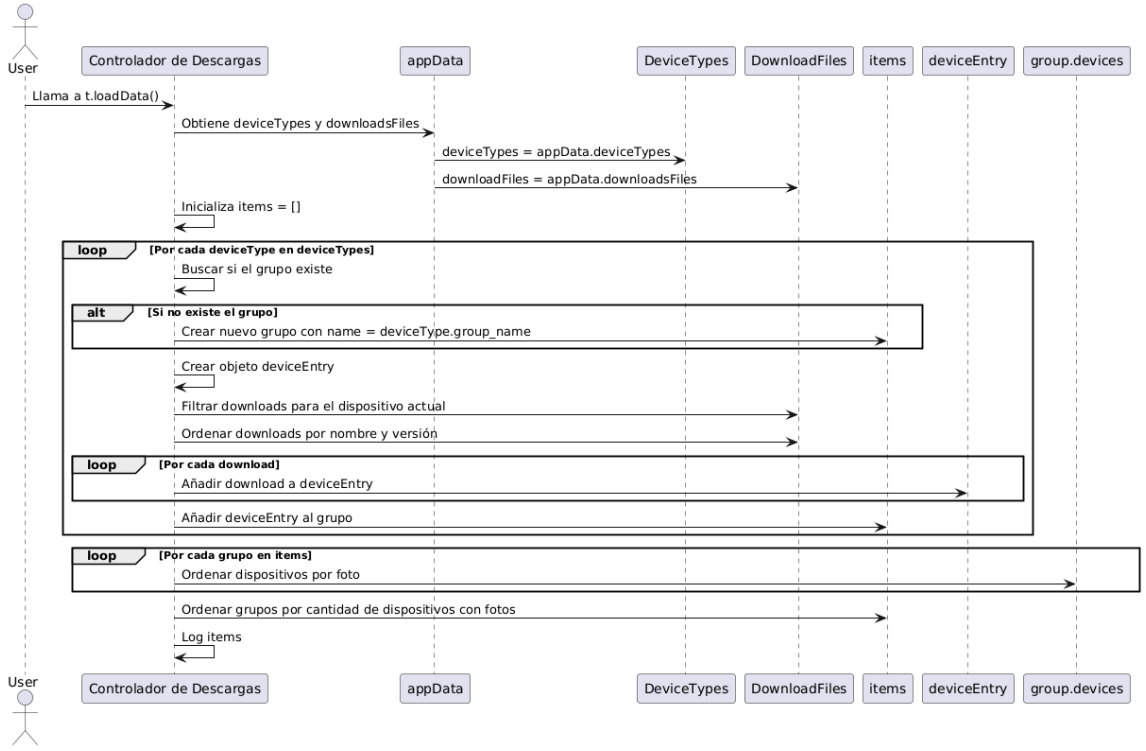


Figura 89. Diagrama de secuencia loadData. Creación propia.

La figura 89 muestra cómo la función **t.loadData()** interactúa con los datos y realiza las tareas de agrupamiento, filtrado y ordenación de los dispositivos y sus archivos de descarga.

Funciones útiles

Antes de describir los modales utilizados en la aplicación, es necesario entender tres funciones clave que forman la base de la implementación del código:

- **downloadFile**: Esta función permite descargar archivos desde el servidor (backend) al usuario. Gestiona la transferencia de archivos, asegurando que el archivo correcto sea enviado al cliente cuando se solicita.
- **deleteFile**: Se encarga de eliminar un archivo almacenado en el servidor. Es esencial para la gestión de archivos, permitiendo que el sistema borre archivos específicos cuando ya no sean necesarios o deban ser reemplazados.
- **uploadFile**: Esta función sube una imagen al servidor, reemplazando la foto anterior si es necesario. Permite la carga de archivos en el sistema, asegurándose de que el archivo recién subido se guarde correctamente en el servidor, y también borra la foto antigua si corresponde.

```
this.downloadFile = angular.bind(this, function(id) {
    $http({
        url: "/downloads/"+id,
        method: "GET",
        responseType: 'arraybuffer',
        params: {
            id: id
        }
    }).success(angular.bind(this, function(response) {
        var file = new Blob([response], {
            type: 'application/' + id.split(".")[id.split(".").length - 1],
        });
        FileSaver.saveAs(file, id);
    }))
})

```

Figura 90. Función downloadFile . Creación propia.

```
// Función para eliminar un archivo del servidor
this.deleteFile = function(filename) {
    $http.post('/deleteDownloadFile', { fileName: filename
}).then(function(response) {
        console.log(response.data.message);
    }).catch(function(error) {
        console.error(error.data.error);
    });
};
```

Figura 91. Función deleteFile. Creación propia.

```
// Función para subir la imagen
t.uploadFile = function(file, oldPhotoName) {
    console.log('Archivo seleccionado:', file);
    if (!file) {
        console.error('No se ha seleccionado ningún archivo para subir.');
        return Promise.reject('No se ha seleccionado ningún archivo'); // Retornar una promesa
rechazada si no hay archivo
    }

    var formData = new FormData();
    formData.append('file', file);
    formData.append('oldPhotoName', oldPhotoName); // Añadir el nombre de la foto antigua

       // Mostrar el contenido de formData
        console.log('FormData contenido:', formData.get('file'), formData.get('oldPhotoName'));

    // Retorna la promesa de $http.post
    return $http.post('/upload/image', formData, {
        headers: { 'Content-Type': undefined },
        transformRequest: angular.identity
    }).then(function(response) {
        console.log('Archivo subido exitosamente:', response.data);
        return response.data; // Retorna el resultado de la subida
    }).catch(function(error) {
        console.error('Error al subir el archivo:', error);
        return Promise.reject(error); // Retorna un rechazo si ocurre un error
    });
};
```

Figura 91. Función uploadFile. Creación propia.

Para que los procesos de descarga, eliminación y carga de archivos funcionen correctamente, es necesario realizar algunas modificaciones en el backend, específicamente en el archivo **appServer.js**. En este archivo, se encuentran las peticiones GET y POST que permiten acceder a las carpetas del servidor y manejar archivos.

Primero, se debe instalar y configurar la biblioteca **multer[1]**, que es responsable de gestionar la carga de archivos. **Multer** permite especificar los directorios en los que se guardarán los archivos subidos, y también puede manejar la eliminación de archivos antiguos. A continuación, se deben añadir las rutas GET y POST correspondientes para permitir:

1. **Acceder a las carpetas**: Configurar las rutas para servir archivos al usuario cuando se hace una solicitud de descarga.
2. **Subir archivos**: Crear una ruta POST que maneje la carga de archivos, utilizando **multer** para guardar los archivos en el directorio adecuado.
3. **Eliminar archivos**: Crear una ruta POST que permite eliminar archivos específicos del servidor, asegurando que los archivos se eliminen correctamente.

El código configura **multer** para guardar imágenes en la carpeta **/public/images** y mantiene el nombre original del archivo. Al subir una nueva imagen, se verifica si hay una foto antigua asociada y, de ser así, se elimina antes de guardar la nueva. Si no se sube un archivo, se devuelve un error. Este proceso asegura la gestión adecuada de las imágenes en el servidor.

```
// Configura Multer para guardar archivos en la carpeta /private/images
const storage = multer.diskStorage({
    destination: function (req, file, cb) {
        cb(null, path.join(__dirname, '/public/images')); // Guardar archivos en /public/images
    },
    filename: function (req, file, cb) {
        cb(null, file.originalname); // Puedes personalizar el nombre del archivo si es necesario
    }
});

const upload = multer({ storage: storage });
app.post('/upload/image', upload.single('file'), (req, res) => {
    const oldPhotoName = req.body.oldPhotoName;

    if (oldPhotoName && oldPhotoName != req.file.name) {
        const  oldPhotoPath = path.join(__dirname, 'public/images', oldPhotoName);
        // Comprueba si el archivo existe antes de intentar borrarlo
        fs.access(oldPhotoPath, fs.constants.F_OK, (err) => {
            if (!err) {
                fs.unlink(oldPhotoPath, (unlinkErr) => {
                    if (unlinkErr) console.error('Error al eliminar el archivo antiguo:', unlinkErr);
                    else console.log('Archivo antiguo eliminado:', oldPhotoName);
                });
            }
        });
    }

    if (!req.file) {
        return res.status(400).send('No se ha subido ningún archivo.');
    }

    res.send({ message: 'Archivo subido exitosamente.', file: req.file });
});
```

Figura 92. Función subir imágenes al servidor. Creación propia

```
//Downloads section and Static Files serve
app.use('/downloads', function(req, res, next) {
    if (req.isAuthenticated() && (req.user.user_group == 1 || req.user.user_group == 3)) {
        return next();
    } else {
        res.status(404).end();
    }
}, express.static('app/downloads'));


// Configuración de multer para almacenar archivos
const downloadStorage  = multer.diskStorage({
    destination: function(req, file, cb) {
        cb(null, path.join(__dirname, 'app/downloads')); // Carpeta donde se guardarán los archivos
    },
    filename: function(req, file, cb) {
        cb(null, file.originalname); // Mantener el nombre original del archivo
    }
});


const downloadUpload  = multer({ storage: downloadStorage });

app.post('/uploadDownloadFile', downloadUpload.single('file'), (req, res) => {
    if (req.isAuthenticated() && (req.user.user_group == 1 || req.user.user_group == 3)) {
        if (!req.file) {
            return res.status(400).json({ error: 'File is required' });
        }
        res.json({ message: 'File uploaded successfully', file: req.file });
    } else {
        res.status(403).json({ error: 'Access denied' });
    }
});


app.post('/deleteDownloadFile', (req, res) => {
    if (req.isAuthenticated() && (req.user.user_group == 1 || req.user.user_group == 3)) {
        const { fileName } = req.body;
        if (!fileName) {
            return res.status(400).json({ error: 'Filename is required' });
        }

        const filePath = path.join(__dirname, 'app/downloads', fileName);

        fs.access(filePath, fs.constants.F_OK, (err) => {
            if (err) {
                return res.status(404).json({ error: 'File not found' });
            }

            fs.unlink(filePath, (unlinkErr) => {
                if (unlinkErr) {
                    return res.status(500).json({ error: 'Error deleting file', details: unlinkErr });
                }
                res.json({ message: 'File deleted successfully' });
            });
        });
    } else {
        res.status(403).json({ error: 'Access denied' });
    }
});
```

Figura 93. Funciones backend control de archivos. Creación propia.

Creación y edición de dispositivos / Creación nuevo grupo

La creación y edición de dispositivos se gestionan mediante funciones específicas en el controlador, accesibles desde un menú desplegable en la vista principal. Según la acción seleccionada, se invoca un controlador que maneja cada operación, utilizando el mismo archivo HTML pero con diferentes parámetros para adaptarse a las

necesidades de cada caso.

```
<div class="modal-header">
    <h3 class="modal-title">{{c.text}}</h3>
</div>
<div class="modal-body">
    <div class="row">
        <div class="col-md-12">
            <div class="row">
                <div class="col-md-6">
                    <div class="form-group">
                        <label>Name</label>
                        <input ng-model="c.editDeviceType.name" class="form-control" placeholder="">
                    </div>
                </div>

                <div class="col-md-6" ng-if="!c.group_name">
                    <div class="form-group">
                        <label>Group Name</label>
                        <input ng-model="c.editDeviceType.group_name" class="form-control" placeholder="">
                            <p class="alert alert-info" role="alert">
                                <strong>Notice:</strong> Write different group to be created.
                            </p>
                    </div>
                </div>

            </div>
            <div class="row">
                <div class="col-md-6">
                    <div class="form-group">
                        <label>Photo</label>
                        <!-- Usa solo file-model sin ng-model para inputs tipo archivo -->
                                <!-- Foto actual -->
                        <div style="display: flex; align-items: center; margin-bottom: 10px;">
                            <!-- Foto actual -->
                            <div ng-if="c.editDeviceType.photo" style="margin-right: 20px;">
                                <p>Foto actual:</p>
                                <img ng-src="public/images/{{c.editDeviceType.photo}}" alt="Foto actual" style="max-width: 200px; max-height:
200px;">
                            </div>

                            <!-- Vista previa de la nueva foto -->
                            <div ng-if="c.previewPhoto">
                                <p>Vista previa de la nueva foto:</p>
                                <img ng-src="{{c.previewPhoto}}" alt="Vista previa" style="max-width: 200px; max-height: 200px;">
                            </div>
                        </div>
                        <input type="file" class="form-control" onchange="angular.element(this).scope().c.onFileChange(this)">
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
<div class="modal-footer">
    <div class="row">
        <div class="col-md-6">
            <button ng-click="c.ok()" class="btn btn-success btn-block">Update</button>
        </div>
        <div class="col-md-6">
            <button ng-click="c.cancel()" type="reset" class="btn btn-primary btn-block">Cancel</button>
        </div>
    </div>
</div>
```

Figura 94. Vista edición y creación de dispositivos. Creación propia.

Esta vista es utilizada por tres controladores para realizar diferentes acciones: **crear un dispositivo**, **editar un dispositivo** y **crear un nuevo grupo junto a un dispositivo inicial**. En el caso de **crear un dispositivo**, permite introducir un nombre y una foto. Para **editar un dispositivo**, se puede modificar el nombre y, si se selecciona una nueva foto, se muestra una vista previa de la foto que reemplaza a la anterior. Al **crear un nuevo grupo**, se especifica un nombre para el grupo y se asocia con un dispositivo inicial. La misma vista se usa en los tres casos, con diferentes parámetros para cada acción.

Como se muestra en la figura 94, la vista proyecta una imagen del dispositivo y

además, dispone de un “picker” de imágenes, para adaptarlo a angular se hace uso de **“onchange="angular.element(this).scope().c.onFileChange(this)""** en cada controlador**.**

```
this.onFileChange = function(input) {
    var file = input.files[0]; // Obtén el archivo seleccionado
    var self = this;
    if (file) {

        this.editDeviceType.oldPhoto = this.editDeviceType.photo; // Guardar el nombre de la foto antigua
        this.editDeviceType.newPhoto = file; // Asigna el archivo a newPhoto
        var reader = new FileReader();
        this.previewPhoto = URL.createObjectURL(file); // Asigna la vista previa

        reader.onload = function(e) {
            $scope.$apply(function() {
                self.previewPhoto = e.target.result; // Asigna la vista previa
            });
        };

        reader.readAsDataURL(file); // Leer el archivo como URL base64
    } else
        // Si no hay archivo seleccionado, limpia la vista previa
        this.previewPhoto = null;
};
```

Figura 95. Control selector de imágenes. Creación propia.

Controladores

Las siguientes figuras muestran las funciones de **crear dispositivo**, **editar dispositivo** y **crear grupo**, las cuales llaman al mismo controlador. Para crear un dispositivo, la función pasa un dispositivo vacío con un grupo en sus variables. En el caso de editar un dispositivo, se envía el dispositivo ya existente. Por último, la función de crear grupo utiliza la función de crear dispositivo pasándolo completamente null.

```
t.editDeviceType = function(deviceType, group_name) {
    console.log('Dispositivo seleccionado para editar:', deviceType);
    deviceType.group_name = group_name;

    var modalInstance = $uibModal.open({
        animation: true,
        templateUrl: 'EditDeviceType.html',
        controller: 'EditDeviceTypeInstanceCtrl as c',
        size: "md",
        resolve: {
            editDeviceType: function() {
                return _.cloneDeep(deviceType);
            },
            text: function() {
                return "Edit Device Type " + deviceType.name;
            }
        }
    });

    modalInstance.result.then(function(moddata) {
        if (moddata.newPhoto) {

            moddata.photo = moddata.newPhoto.name; // Actualiza el nombre de la foto en moddata
            $http.post('/api/modifyDeviceTypes', moddata).then(function(response) {

            // Llama a uploadFile y pasa la nueva y antigua foto como argumentos
            t.uploadFile(moddata.newPhoto, moddata.oldPhoto).then(function() {
               console.log('Modificando dispositivo:', moddata);
                appData.updateData().then(function(data) {
                    $scope.appData = data;
                    t.loadData();
                });
            }).catch(function(error) {
                console.error('Error al subir el archivo:', error);
                // Puedes manejar el error aquí, como mostrar un mensaje al usuario
            });

            }).catch(function(error) {
                console.error('Error al modificar el dispositivo en la base de datos:', error);
            });

        } else{
            $http.post('/api/modifyDeviceTypes', moddata).then(function(response) {
                appData.updateData().then(function(data) {
                    $scope.appData = data;
                    t.loadData();
                });
            }).catch(function(error) {
                console.error('Error al modificar el dispositivo en la base de datos:', error);
            });
        }

    }, function(response) {
        console.log('Modal cerrado:', response);
    });

};
```

Figura 95. Función invoca al controlador de edición. Creación propia.

```
t.createDeviceType = function(groupName) {
    var modalInstance = $uibModal.open({
        animation: true,
        templateUrl: 'EditDeviceType.html',
        controller: 'EditDeviceTypeInstanceCtrl as c',
        size: "md",
        resolve: {
            editDeviceType: function() {
                return {
                    name: "",
                    group_name: groupName,
                    photo: "",
                    newPhoto: null
                };
            },
            text: function() {
                if (groupName === null) {
                    return "Create Device Type and Group";
                }
                return "Create Device Type of the group " + groupName;
            }
        }
    });

    modalInstance.result.then(function(newDeviceType) {
        console.log('Creando dispositivo:', newDeviceType);
        if (newDeviceType.newPhoto) {
            newDeviceType.photo = newDeviceType.newPhoto.name; // Actualiza el nombre de la foto en newDeviceType
            $http.post('/api/insertDeviceTypes', newDeviceType).then(function(response) {
                // Llama a uploadFile y pasa la nueva foto como argumento
                t.uploadFile(newDeviceType.newPhoto).then(function() {
                    appData.updateData().then(function(data) {
                        $scope.appData = data;
                        t.loadData();
                    });
                }).catch(function(error) {
                    console.error('Error al subir el archivo:', error);
                    // Puedes manejar el error aquí, como mostrar un mensaje al usuario
                });
            }).catch(function(error) {
                console.error('Error al crear el dispositivo en la base de datos:', error);
            });
        } else {
            $http.post('/api/insertDeviceTypes', newDeviceType).then(function(response) {
                appData.updateData().then(function(data) {
                    $scope.appData = data;
                    t.loadData();
                });
            }).catch(function(error) {
                console.error('Error al crear el dispositivo en la base de datos:', error);
            });
        }

    }, function(response) {
        console.log('Modal cerrado:', response);
    });

};
```

Figura 96. La función invoca al controlador de creación del dispositivo. Creación propia.

```
1
2 ////////////////////////////////////////////
3 // Controlador para editar un dispositivo
4 ////////////////////////////////////////////
5
6 angular.module('heimdalApp').controller('EditDeviceTypeInstanceCtrl', function ($scope, $http, $uibModalInstance, text,
  editDeviceType, appData, FileSaver) {
7     this.editDeviceType = editDeviceType;
8     this.appData = appData.data();
9     this.text = text;
10    this.group_name = editDeviceType.group_name; // Guardar el nombre del grupo para el condicional de creacion de grupo
11    this.editDeviceType.oldName = editDeviceType.name;  // Guardar el nombre original
12
13
14
15    this.onFileChange = function(input) {
16        var file = input.files[0]; // Obtén el archivo seleccionado
17        var self = this;
18        if (file) {
19
20            this.editDeviceType.oldPhoto = this.editDeviceType.photo; // Guardar el nombre de la foto antigua
21            this.editDeviceType.newPhoto = file; // Asigna el archivo a newPhoto
22            var reader = new FileReader();
23            this.previewPhoto = URL.createObjectURL(file); // Asigna la vista previa
24
25            reader.onload = function(e) {
26                $scope.$apply(function() {
27                    self.previewPhoto = e.target.result; // Asigna la vista previa
28                });
29            };
30
31            reader.readAsDataURL(file); // Leer el archivo como URL base64
32        } else
33            // Si no hay archivo seleccionado, limpia la vista previa
34            this.previewPhoto = null;
35    };
36
37    this.ok = function () {
38        var self = this;
39        $uibModalInstance.close(self.editDeviceType);
40    };
41
42    this.remove = function () {
43        this.editDeviceType.remove = true;
44        $uibModalInstance.close(this.editDeviceType);
45    };
46
47    this.cancel = function () {
48        $uibModalInstance.dismiss();
49    };
50 });
```

Figura 97. Controlador dispositivo. Creación propia.

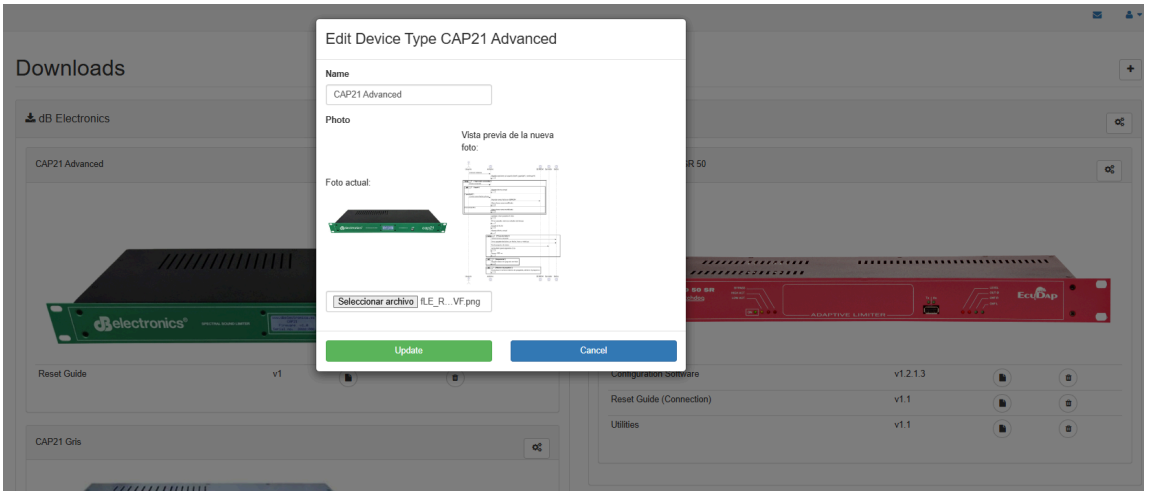


Figura 98. Edición de dispositivos. Creación propia.

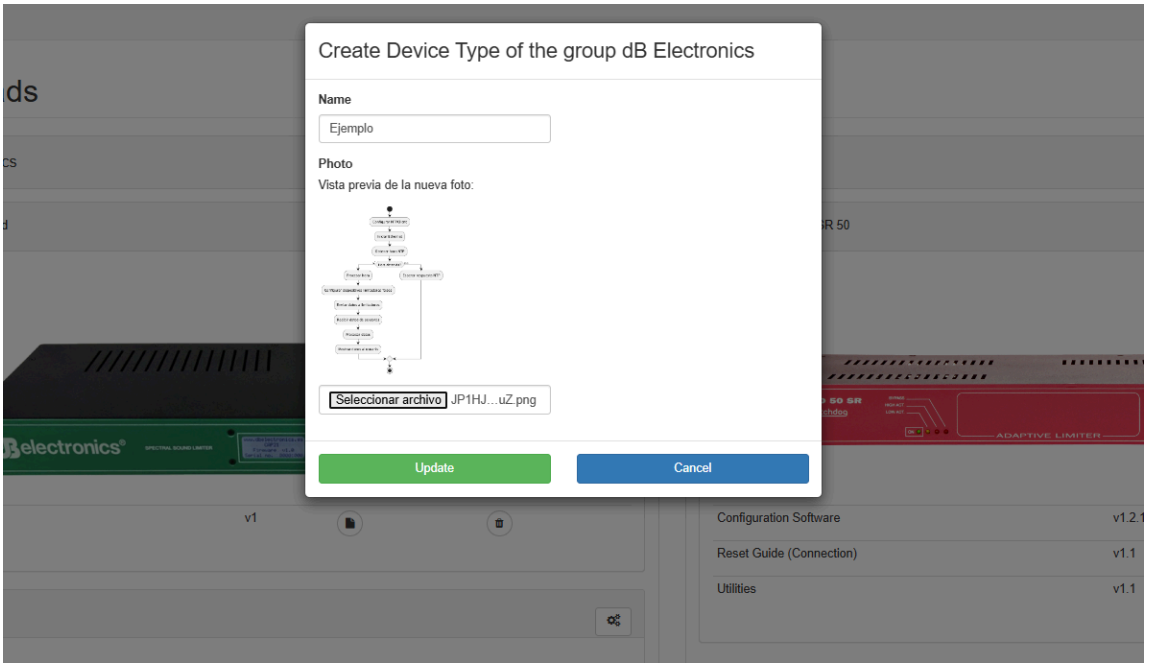


Figura 99. Creación de dispositivos. Creación propia.

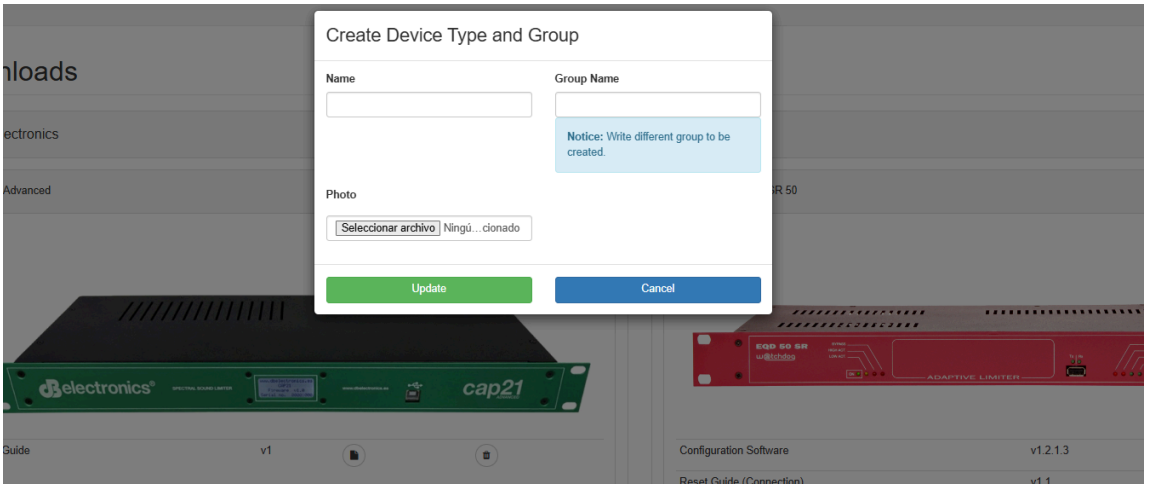


Figura 100. Creación de grupo. Creación propia.

Modificar grupo

A continuación, se procederá a crear el modal para **editar grupo**, donde únicamente será necesario modificar el nombre del grupo. Se creará un modal que mostrará el nombre actual del grupo, permitiendo su edición. Al aceptar los cambios, se realizará una petición GET a la API, que actualizará el nombre del grupo en todos los dispositivos asociados a él.

```
// Funcion para abrir el modal de modificacion de grupo de dispositivos
t.modifyDeviceGroup = function (group_name) {
    console.log('Grupo seleccionado para modificar:', group_name); // Verificar datos antes de abrir modal
    var modalInstance = $uibModal.open({
        animation: true,
        templateUrl: 'ModifyDeviceGroup.html',
        controller: 'ModifyDeviceGroupInstanceCtrl as c',
        size: "md",
        resolve: {
            group_name: function() {
                return group_name;
            }
        }
    });

    modalInstance.result.then(function (moddata) {
        console.log('Modificando grupo:', moddata);
        $http.post('/api/modifyDeviceGroup', moddata).then(function (response) {
            appData.updateData().then(function (data) {
                $scope.appData = data;
                t.loadData();
            });
        });
    }, function (response) {
        console.log("Cancelado:", response);
    });
}

```

Figura 101. Función editar grupo. Creación propia.

```
////////////////////////////////////////////
// Controlador para modificar grupo de dispositivos
////////////////////////////////////////////

angular.module('heimdalApp').controller('ModifyDeviceGroupInstanceCtrl', function ($scope, $http, $uibModalInstance,
  group_name, appData) {
    this.appData = appData.data();
    this.group_name = group_name;
    this.newName = group_name; // Inicializar el nuevo nombre con el nombre actual

    this.ok = function () {
        $uibModalInstance.close({ oldName: group_name, newName: this.newName });
    };

    this.cancel = function () {
        $uibModalInstance.dismiss();
    };
});
```

Figura 102. Controlador editar grupo. Creación propia.

Creación ficheros de descarga

Ahora se procederá a crear el modal para la **creación de ficheros de descarga**. A la función se le enviará el dispositivo correspondiente y, en el modal, se podrá añadir el nombre del archivo, el archivo en sí, un enlace y la versión del fichero. Al cerrar el modal, los datos del archivo se enviarán al servidor, donde se guardará el fichero y se actualizará la base de datos con la nueva entrada, incluyendo todos los detalles proporcionados.

```
        // Función para abrir el modal de creación de fichero de descarga
        t.createFileDevice = function(device) {
            var modalInstance = $uibModal.open({
                animation: true,
                templateUrl: 'CreateFileDevice.html',
                controller: 'CreateFileDeviceInstanceCtrl as c',
                size: "md",
                resolve: {
                    file: function() {
                        return {
                            id: null,
                            name: "",
                            version: "",
                            link: "",
                            path: "",
                            newPhoto: null,
                            device: device.name
                        };
                    },
                    text: function() {
                        return "Create File for the device " + device.name; // Cambiar el texto según el dispositivo
                    }
                }
            });

            modalInstance.result.then(function(newFile) {
                console.log('Creando fichero:', newFile);
                $http.post('/api/insertDownloadFile', newFile).then(function(response) {
                    const formData = new FormData();
                    formData.append('file', newFile.file); // `newFile.file` es el archivo que quieres subir

                    fetch('/uploadDownloadFile', {
                        method: 'POST',
                        body: formData,
                        credentials: 'include' // Asegúrate de enviar cookies de sesión si es necesario
                    }).then(response => response.json())
                    .catch(error => console.error('Error:', error));

                    appData.updateData().then(function(data) {
                        $scope.appData = data;
                        t.loadData();
                    });
                });
            }, function(response) {
                console.log('Modal cerrado:', response);
            });
        }
```

Figura 103. Controlador crear archivo. Creación propia.

```
1
2 ////////////////////////////////////////////
3 // Controladores para añadir los ficheros de descarga
4 ////////////////////////////////////////////
5
6 angular.module('heimdalApp').controller('CreateFileDeviceInstanceCtrl', function ($scope, $http, $uibModalInstance, file,text,
  appData) {
7     this.file = file;
8     this.text = text;
9     this.appData = appData.data();
10
11
12     this.onFileChange = function(input) {
13         var file = input.files[0];
14         var self = this;
15         if (file) {
16             this.file.file = file;
17             var reader = new FileReader();
18
19             reader.onload = function(e) {
20                 $scope.$apply(function() {
21                     self.file.path = file.name; // Actualiza el nombre en la vista previa
22                 });
23             };
24
25             reader.readAsDataURL(file); // Leer el archivo como URL base64
26         } else {
27             this.file.file = null; // Limpia la vista previa si no hay archivo
28         }
29     };
30
31
32
33     this.ok = function () {
34         $uibModalInstance.close(this.file);
35     };
36
37     this.cancel = function () {
38         $uibModalInstance.dismiss();
39     };
40 });
41
42
```

Figura 104. Controlador crear archivo. Creación propia.

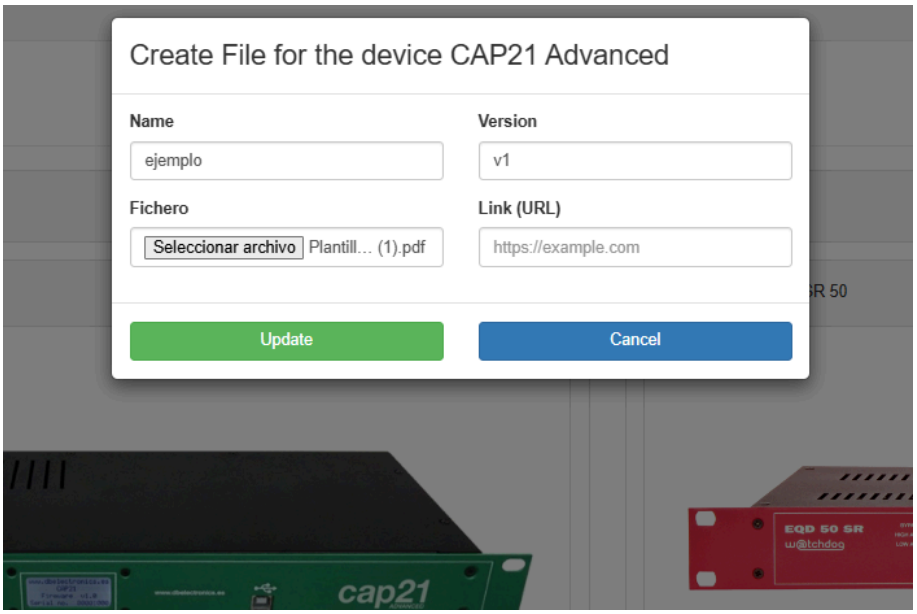


Figura 105. Resultado crear archivo. Creación propia.

Funciones de borrado

Ahora se implementarán las tres funciones de **borrado**, las cuales utilizan el mismo modal. Este modal presenta un mensaje de confirmación que indica si el usuario desea eliminar el elemento en cuestión. Al confirmar la eliminación, la función correspondiente realiza la operación de borrado ya sea de un dispositivo, un grupo o un archivo.

```
////////////////////////////////////////////
// Controlador para borrado de dispositivo
////////////////////////////////////////////

angular.module('heimdalApp').controller('DeleteDataInstanceCtrl', function ($scope, $http, $uibModalInstance, text, appData) {
    this.appData = appData.data();
    this.text = text;
    this.ok = function () {
        $uibModalInstance.close();
    };

    this.cancel = function () {
        $uibModalInstance.dismiss();
    };
});

```

Figura 106. Controlador modal de borrado. Creación propia.

Para comenzar, se implementará la función de **eliminar fichero**. El modal solicitará confirmación al usuario para eliminar el fichero, mostrando su nombre. Al aceptar, se realizará una petición API que eliminará el fichero tanto en el servidor como en la base de datos.

```
// Funcion para abrir modal de borrado de fichero
t.deleteFileDevice = function (file) {
    console.log('Fichero seleccionado para borrar:', file); // Verificar datos antes de abrir modal
    var modalInstance = $uibModal.open({
        animation: true,
        templateUrl: 'DeleteData.html',
        controller: 'DeleteDataInstanceCtrl as c',
        size: "md",
        resolve: {
            text: function() {
                return "Are you sure you want to delete the file " + file.name + "? This action cannot be undone.";
            }
        }
    });

    modalInstance.result.then(function () {

        console.log('Borrando fichero:', file);
        $http.post('/api/deleteDownloadsFiles', file).then(function (response) {
            if(file.path)
                t.deleteFile(file.path);
            appData.updateData().then(function (data) {
                $scope.appData = data;
                t.loadData();
            });
        });
    }, function (response) {
        console.log("Cancelado:", response);
    });
}
```

Figura 107. Función borrado de ficheros. Creación propia.

El proceso de eliminación de dispositivo sigue una lógica similar. El modal mostrará el nombre del dispositivo y pedirá confirmación para su eliminación. Al aceptar, se ejecutará una petición API que eliminará el dispositivo de la base de datos y también borrará la imagen asociada en el servidor.

```
// Función para abrir el modal de borrado de dispositivo
// Recibe el dispositivo a borrar
t.deleteDeviceType = function (deviceType) {
    console.log('Dispositivo seleccionado para borrar:', deviceType); // Verificar datos antes de abrir modal
    var modalInstance = $uibModal.open({
        animation: true,
        templateUrl: 'DeleteData.html',
        controller: 'DeleteDataInstanceCtrl as c',
        size: "md",
        resolve: {
            text: function() {
                return "Are you sure you want to delete the device " + deviceType.name + "? This action cannot be undone.";
            }
        }
    });

    modalInstance.result.then(function () {
        console.log('Borrando dispositivo:', deviceType);
        $http.post('/api/deleteDeviceTypes', _.cloneDeep(deviceType)).then(function (response) {
            appData.updateData().then(function (data) {
                $scope.appData = data;
                t.loadData();
            });
        });
    }, function (response) {
        console.log("Cancelado:", response);
    });
};
```

Figura 108. Función borrado de dispositivo. Creación propia.

El proceso de **eliminación de grupo** funciona de manera similar al de los dispositivos. El modal mostrará el nombre del grupo y pedirá confirmación para su eliminación. Al aceptar, se enviará una petición API que eliminará todos los dispositivos asociados a ese grupo en la base de datos.

```
// Funcion para abrir el modal de borrado de grupo de dispositivos
// Recibe el nombre del grupo a borrar
t.deleteGroup = function (group_name) {
    console.log('Grupo seleccionado para borrar:', group_name); // Verificar datos antes de abrir modal
    var modalInstance = $uibModal.open({
        animation: true,
        templateUrl: 'DeleteData.html',
        controller: 'DeleteDataInstanceCtrl as c',
        size: "md",
        resolve: {
            text: function() {
                return "Are you sure you want to delete the group " + group_name + "? This action cannot be undone.";
            }
        }
    });

    modalInstance.result.then(function () {
        console.log('Borrando grupo:', group_name);
        $http.post('/api/deleteDeviceGroup', { group_name: group_name }).then(function (response) {
            appData.updateData().then(function (data) {
                $scope.appData = data;
                t.loadData();
            });
        });
    }, function (response) {
        console.log("Cancelado:", response);
    });
};

```

Figura 109. Función borrado de grupo. Creación propia.

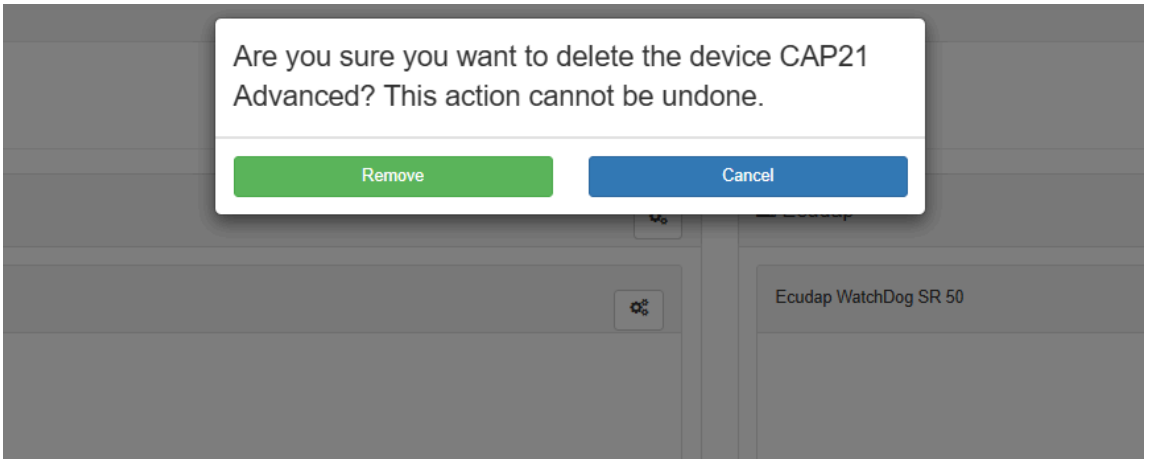


Figura 110. Resultado de borrado. Creación propia.

Por último, para dejar la vista preparada para producción, se añaden datos de grupos, dispositivos y ficheros con datos correctos que había anteriormente en forma de array sin implementación ni vista ninguna.

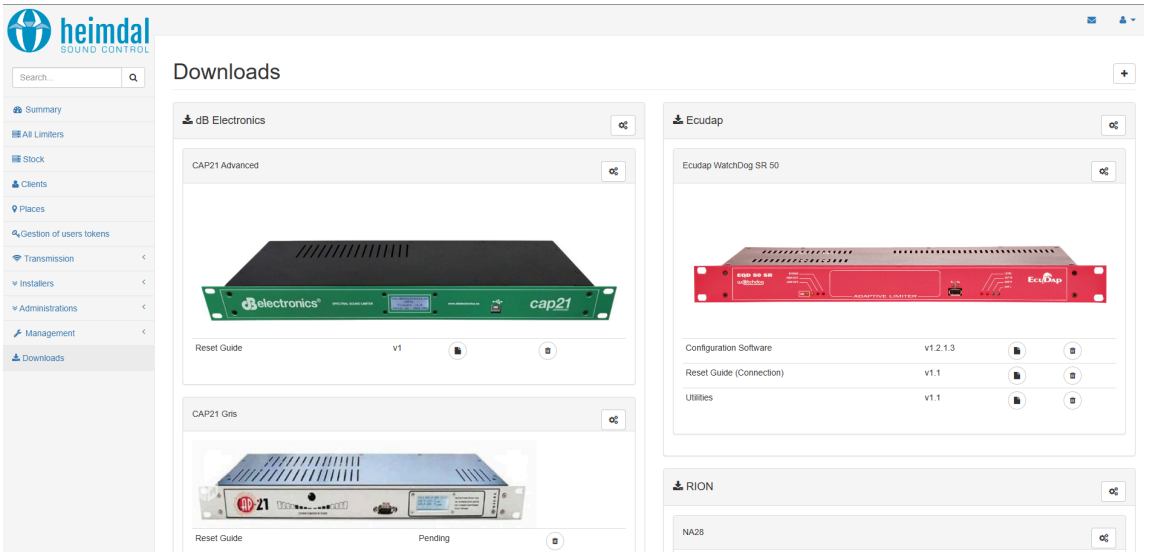


Figura 110. Resultado de borrado. Creación propia.

7.3.5 Arreglos en Despliegue

Durante el despliegue al servidor, se detectó un fallo crítico: la aplicación fallaba al recibir un alto volumen de paquetes con errores, deteniendo el sistema. Para resolverlo, se implementaron **promesas de Node.js** para manejar los paquetes con errores de forma adecuada. Además, se añadió **validación de datos** para evitar que información incorrecta o corrupta llegará a la base de datos, asegurando la estabilidad y la integridad del sistema.

7.3.5 Retrospectiva

La **Etapa 3** ha sido la más extensa y compleja, con mejoras en el **FrontEnd** y **BackEnd**. Esta fase incluyó una revisión profunda de la interfaz y optimizaciones en el backend para mejorar la eficiencia y gestión de datos. Aunque desafiante, todas las tareas principales se han completado, dejando algunas áreas abiertas para futuras mejoras.

Un **problema identificado** que se abordará en el futuro es que, al eliminar un grupo o dispositivo, las imágenes o archivos relacionados no se eliminan correctamente. Este inconveniente ha sido registrado como una tarea pendiente para próximas etapas del proyecto.

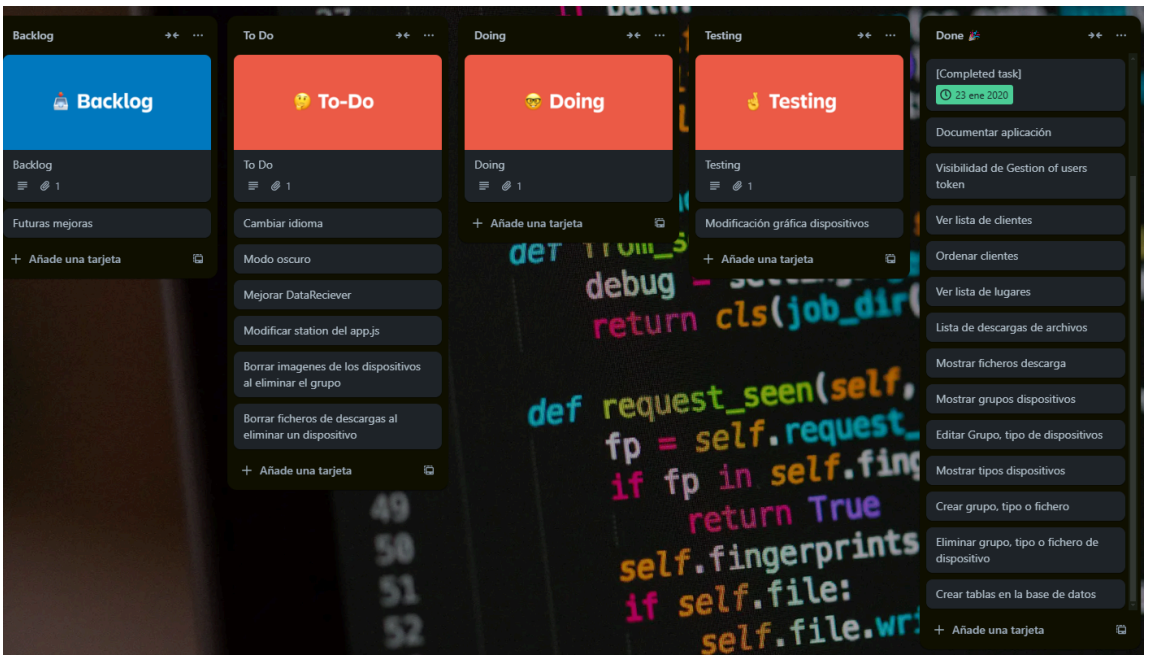


Figura 111. Tablero Kanban etapa 3 Final. Creación propia.

8. Despliegue

Para el despliegue de la aplicación en el servidor heimdalsoundcontrol.com, se realizaron varias modificaciones en el código para ajustarse a la infraestructura de red y la configuración del servidor. A continuación se detallan los cambios y la configuración necesaria para que todo funcione correctamente.

8.1 Configuración del Servidor y Nginx

El servidor **heimdalsoundcontrol.com** cuenta con un **proxy inverso de Nginx** que organiza las peticiones entre los diferentes servicios. Este proxy redirige las solicitudes HTTP a los puertos y servicios correspondientes según la configuración de Nginx. Las modificaciones en los puertos e IPs de los servicios de Docker se realizarán de la siguiente manera:

1. **Aplicación Web (Frontend y Backend en Docker):** La aplicación se ejecutará en el **puerto 8000**, para que coincida con la configuración preexistente de Nginx. De esta forma, las solicitudes del usuario serán manejadas por Nginx, que redirigirá las peticiones al contenedor de la aplicación en el puerto 8000.
2. **phpMyAdmin:** phpMyAdmin se mantendrá en el **puerto 8090**. Aunque no cambia el puerto, se ajustará la IP del servicio para que Nginx pueda gestionar las solicitudes de phpMyAdmin y dirigirlas al contenedor correspondiente.
    a. Realmente para su fácil acceso se configurará una ruta en heimdalsoundcontrol.com/phpmyadmin que dirigirá directamente a este servicio.
3. **Base de Datos MySQL:** La base de datos MySQL se mantendrá en el **puerto 3306**. Al igual que con los otros servicios, se configurará para que su tráfico sea manejado internamente por Nginx y los contenedores de Docker por lo que no se tendrá acceso externo a este servicio.

Configuración de IP Interna

Para todos los servicios anteriores (aplicación web, phpMyAdmin, y MySQL), se utilizará la **IP interna 127.0.0.1**. Esto significa que cada servicio escuchará localmente en el contenedor, y será Nginx quien se encargue de redirigir el tráfico externo a los contenedores correspondientes. Esta configuración asegura que los servicios no sean directamente accesibles desde fuera del servidor, sino que todo el tráfico pase a través del proxy inverso Nginx.

También como medida preventiva sólo se tendrá acceso al **phpmyadmin** del servidor desde una IP propia del tutor del proyecto y dueño de GranaSat, **Andrés Roldan**.

```
server {

        listen 80;
        return 301 https://heimdalsoundcontrol.com$request_uri;
}


server {

        server_name heimdalsoundcontrol.com www.heimdalsoundcontrol.com;

        listen 443 ;

        ssl on;
    ssl_certificate /etc/letsencrypt/live/heimdalsoundcontrol.com-0001/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/heimdalsoundcontrol.com-0001/privkey.pem; # managed by Certbot


        ssl_session_cache builtin:1000 shared:SSL:20m;
        ssl_session_timeout 180m;
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
       ssl_prefer_server_ciphers on;
        ssl_ciphers HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;

        location / {
          #try_files $uri $uri/ =404;


          proxy_set_header Host $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
          proxy_set_header X-Forwarded-Proto $scheme;

          proxy_pass http://127.0.0.1:8000;

        }

        location /phpmyadmin/ {
            # Restringir el acceso a las IPs permitidas
            #allow 150.214.102.51;       # Sustituye con la IP permitida real
            #allow 86.127.227.236;       # Otra IP permitida
            #deny all;                   # Denegar todas las demás IPs

            # Redirigir el tráfico al puerto donde corre phpMyAdmin
            proxy_pass http://127.0.0.1:8090;

            # Encabezados necesarios para pasar la información de la solicitud
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;

            # Sustituciones para phpMyAdmin (corrige los enlaces relativos)
            sub_filter_once off;
            sub_filter 'href="/' 'href="/phpmyadmin/';
            sub_filter 'src="/' 'src="/phpmyadmin/';

            # Deshabilitar proxy_redirect para evitar redireccionamientos incorrectos
            proxy_redirect off;

            # Manejo correcto de barras finales en las rutas
            rewrite ^/phpmyadmin/(.*)$ /$1 break;
        }
}
```

Figura 112. Configuración nginx heimdalsoundcontrol.com. Creación propia.

8.2 Despliegue a través de GitLab

El código del proyecto será gestionado y desplegado desde un servidor GitLab privado

del laboratorio de **GranaSat**. El acceso a este servidor estará autorizado para **Ándres Roldán**, tutor del proyecto y director del grupo. Los pasos a seguir son los siguientes:

1. **Acceso a GitLab:** Los archivos del proyecto se encontrarán en el repositorio privado de GitLab del laboratorio. Ándres Roldán tendrá acceso a este repositorio para revisar y gestionar el código.
2. **Descarga de los Archivos al Servidor:** Desde el servidor heimdalsoundcontrol.com, se utilizarán herramientas como git de GitLab para clonar el repositorio y descargar los archivos necesarios. Esto asegurará que el servidor esté siempre actualizado con los últimos cambios del proyecto.
3. **Configuración del Servidor de Producción:** Una vez descargado el código, se configurará Nginx para que actúe como proxy inverso, dirigiendo las peticiones del cliente hacia los puertos configurados. Esto incluye la configuración de SSL (si es necesario), y la correcta redirección entre la aplicación web, phpMyAdmin y MySQL.
4.

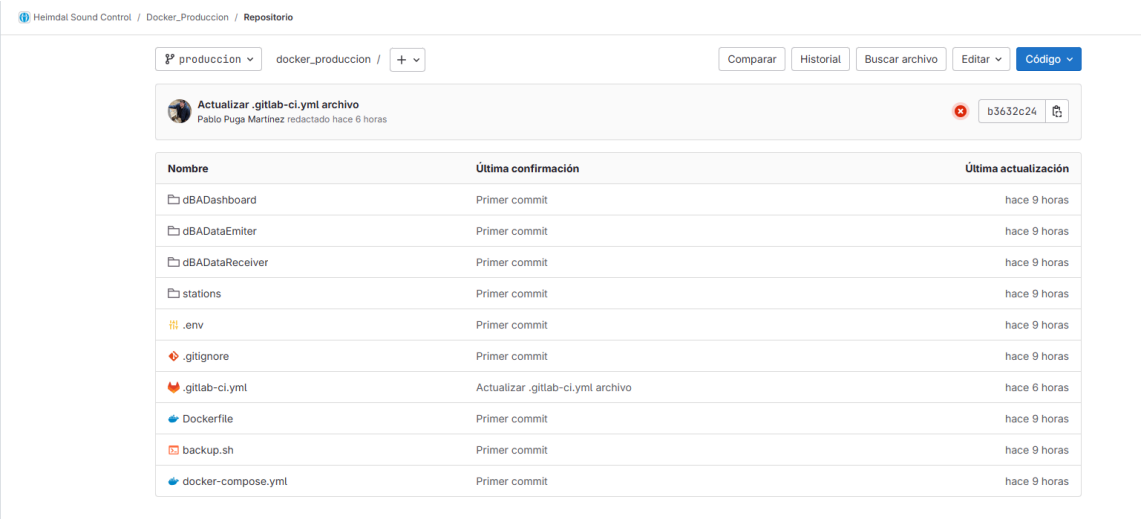


Figura 113. Producción de GitLab. GitLab GranaSat.

Resumen de Modificaciones y Configuración

1. **Puertos y IPs:**
   - **Aplicación Web:** Puerto 8000, IP 127.0.0.1.
   - **phpMyAdmin:** Puerto 8090, IP 127.0.0.1.
   - **MySQL:** Puerto 3306, IP 127.0.0.1.
2. **Servidor GitLab de GranaSat:**
   - El código será gestionado desde un repositorio privado en GitLab.
   - Ándres Roldán tendrá acceso al repositorio para revisar el proyecto y ayudar en el despliegue.
3. **Proxy inverso Nginx:**
   - Nginx manejará las peticiones y las redirigirá correctamente a los contenedores de la aplicación, phpMyAdmin y MySQL.

Con estos ajustes, el despliegue en el servidor será más eficiente y estará correctamente aislado, lo que facilita su gestión y mantenimiento a largo plazo.

9. Conclusiones y futuras mejoras

Este Trabajo de Fin de Grado ha sido fundamental para consolidar y aplicar los conocimientos adquiridos durante el grado, resultando en una mejora significativa de la plataforma de monitorización acústica. A través de la implementación de contenedores Docker y la configuración de un proxy inverso con Nginx y PM2, se ha logrado un sistema más estable y eficiente, que optimiza la gestión de servicios y facilita la escalabilidad del proyecto. Además, se ha desarrollado un simulador de dispositivos de ruido, que permitió analizar la comunicación de datos bajo distintas condiciones, aportando realismo y fiabilidad al entorno de pruebas.

El desarrollo del frontend y backend en Angular y Node.js ha permitido optimizar tanto la interfaz como el rendimiento de la aplicación, garantizando una experiencia de usuario más fluida y organizada. Durante la etapa de implementación, se abordaron distintos desafíos, especialmente en la recepción de paquetes erróneos, donde la solución mediante el manejo de promesas en Node.js ha mejorado considerablemente la estabilidad de la aplicación. No obstante, se identificaron algunos aspectos a mejorar en futuras versiones del sistema. Entre ellos destacan el perfeccionamiento del procesamiento de datos fallidos y la implementación de una limpieza automática de archivos e imágenes al eliminar dispositivos o grupos, para evitar la acumulación de datos innecesarios en el servidor.

La colaboración con el equipo del proyecto GranaSAT ha brindado un valioso contexto multidisciplinario, facilitando la aplicación del sistema a posibles usos en otros entornos de monitoreo y permitiendo la integración de conocimientos en el campo de la ingeniería. En conclusión, el proyecto ha cumplido con los objetivos planteados, logrando mejoras sustanciales en el sistema de monitorización y sentando una base sólida para futuras optimizaciones y ampliaciones en su funcionamiento.

10. Bibliografía

(1). (n.d.). Node.js — Run JavaScript Everywhere. Retrieved November 11, 2024, from https://nodejs.org/en/

(2). (n.d.). AngularJS — Superheroic JavaScript MVW Framework. Retrieved November 11, 2024, from https://angularjs.org/

(3). (n.d.). Trello: Manage Your Team's Projects From Anywhere. Retrieved November 11, 2024, from https://trello.com/

(4). (n.d.). Carbon | Create and share beautiful images of your source code. Retrieved November 11, 2024, from https://carbon.now.sh

(5). (n.d.). PlantUML. Retrieved November 11, 2024, from http://plantuml.com

(6). (n.d.). draw.io. Retrieved November 12, 2024, from http://draw.io

(7). *CAP21 Basic Sound Limiter*. (n.d.). dBelectronics. Retrieved November 11, 2024, from https://www.dbelectronics.es/en/producto/cap21-basic-sound-limiter/

(8). *Docker Desktop: The #1 Containerization Tool for Developers*. (n.d.). Docker. Retrieved November 11, 2024, from https://www.docker.com/products/docker-desktop/

(9). *Getting Started with Arduino products*. (2024, March 27). Arduino Uno. Retrieved November 11, 2024, from https://store.arduino.cc/en-es/products/arduino-uno-rev3?srsltid=AfmBOord9mWwUH6FKuvgaHPbF2FXn_pd5RKGTVELsjwQQc_9k_0A7dtd

(10). Roldan Miranda, A. (n.d.). *Heimdal Sound control*. heimdalsoundcontrol.com. heimdalsoundcontrol.com

(11). Vallejo, S. (2023, October 19). *Mapa de ruidos de Granada: la mitad de la población soporta altos niveles acústicos por tráfico*. Granada Hoy. Retrieved November 11, 2024, from https://www.granadahoy.com/granada/Mapa-ruidos-Granada-altos-niveles-acust

icos-trafico_0_1840316697.html

(12). *yEd - Download*. (n.d.). yWorks. Retrieved November 11, 2024, from https://www.yworks.com/products/yed/download#download

(13). (n.d.). dbdiagram.io - Database Relationship Diagrams Design Tool. Retrieved November 12, 2024, from http://dbdiagram.io

(14). GranaSAT – Electronics Aerospace Group. Retrieved November 12, 2024, from https://granasat.space/

Anexo 1. Manual de Usuario para la Aplicación de Monitoreo Acústico

Introducción

Este manual tiene como objetivo guiar a los usuarios (en este caso, los examinadores) a través del proceso de instalación y acceso a la aplicación de monitoreo acústico. El sistema está diseñado para controlar los niveles acústicos en locales con equipos de música, permitiendo el monitoreo en tiempo real a través de una interfaz web.

Requisitos previos

Antes de comenzar con la instalación de la aplicación, asegúrese de que su sistema cumpla con los siguientes requisitos:

1. **Node.js**: Asegúrese de tener instalada la versión más reciente de Node.js.
    - Puede descargar Node.js desde aquí.
2. **Docker y Docker-Compose**: Es necesario tener Docker y Docker-Compose instalados para el despliegue del proyecto.
    - Puede descargar Docker desde aquí.
    - Para instalar Docker-Compose, siga las instrucciones aquí.
3. **Bower**: La aplicación usa Bower para gestionar dependencias de frontend.
    - Para instalar Bower, ejecute: **npm install -g bower.**

Paso 1: Preparar el Proyecto

1. **Descomprima el archivo ZIP**:
    - Descomprima el archivo ZIP proporcionado en la entrega.
2. **Instalar dependencias**:
    - Una vez descomprimido el proyecto, abra una terminal en el directorio raíz del proyecto y ejecute los siguientes comandos para instalar las dependencias necesarias:
        i. npm install
        ii. bower install

Paso 2: Configuración de la Base de Datos

1. **Base de Datos**:
    - Use el archivo de base de datos alojado en el fichero de entrega.
2. **Ubicación del archivo de base de datos**:
    - Coloque el archivo de la base de datos descargada en el directorio principal del proyecto, es decir, en la misma carpeta donde se encuentra el archivo docker-compose.yml.
3. **Actualizar el archivo .env**:
    - En el archivo .env, encontrará una entrada correspondiente al archivo de la base de datos. Actualice el nombre del archivo de la base de datos en la siguiente línea:
        i. SQL_BACKUP = backup.sql
4. Reemplace backup.sql con el nombre del archivo de base de datos que acaba

de colocar en el directorio principal.
5. Si no se tiene la base de datos inicializada, aparte de tener el archivo se debe descomentar en el archivo docker-compose.yml la línea:
    - “- ./${SQL_BACKUP}:/docker-entrypoint-initdb.d/init.sql”
6. Nota: La copia de backup tarda hasta 20 minutos en inicializarse, recomiendo hacer uso de docker logs mysql para ver el logs que diga que el “temporal server is closed”.

Paso 3: Configuración de Docker

1. **Iniciar los contenedores de Docker**:
    - En el directorio principal del proyecto, donde se encuentra el archivo docker-compose.yml, ejecute el siguiente comando para iniciar los contenedores de Docker:
        i. docker-compose up -d
2. Esto iniciará los servicios necesarios para la aplicación (servidor web, base de datos, phpMyAdmin).

Paso 4: Acceder a la Aplicación

1. **Acceder a la aplicación web**:
    - Abra un navegador web y acceda a la siguiente URL para usar la aplicación de monitoreo acústico:
        i. http://0.0.0.0:8000
2. **Credenciales de acceso**:
    - Para acceder a la web, utilice las siguientes credenciales:

        i. **Usuario:** lpeinado

        ii. **Contraseña:** *Solicitar”

3. **Acceder a phpMyAdmin**:
    - Para gestionar la base de datos, puede acceder a phpMyAdmin usando la siguiente URL en su navegador:
        i. http://127.0.0.1:8090
4. **Credenciales de acceso**:
    - Para acceder a phpMyAdmin, utilice las siguientes credenciales:

        i. **Usuario:** dba

        ii. **Contraseña:** dbapassword

Paso 5: Finalización

Una vez realizados estos pasos, la aplicación debería estar operativa y lista para su uso. En caso de que desee detener los contenedores de Docker, ejecute el siguiente comando en el directorio principal del proyecto:

- docker-compose down

Solución de Problemas

- **Problema al ejecutar npm install**: Si se encuentran errores al ejecutar npm

install, asegúrese de tener la versión más reciente de Node.js y de haber ejecutado bower install para instalar las dependencias de frontend.

Problema con Docker: Si encuentra problemas al levantar los contenedores de docker, asegúrese de tener correctamente configurado Docker y Docker-Compose. Puede verificar que Docker esté corriendo usando el comando:

- docker ps

Conclusión

Una vez completados estos pasos, los examinadores podrán interactuar con la aplicación de monitoreo acústico, gestionar la base de datos y acceder a la plataforma a través de la interfaz web proporcionada.

Anexo 2 : Cumplimiento con la Legislación de Protección de Datos

A2.1. Cumplimiento con el Reglamento General de Protección de Datos (RGPD)

El sistema desarrollado en este proyecto maneja datos sensibles y personales, por lo que se ha tomado especial cuidado para garantizar el cumplimiento del **Reglamento General de Protección de Datos (RGPD)** de la Unión Europea, que establece las pautas para la protección y el manejo adecuado de la información personal.

El procesamiento de datos solo se lleva a cabo con el **consentimiento informado** de los usuarios, y estos tienen derechos sobre sus datos, como la **rectificación**, **supresión**, y **portabilidad**. Los usuarios pueden ejercer estos derechos en cualquier momento, contactando con los responsables del sistema.

A2.2. Política de Privacidad

Este proyecto sigue las directrices establecidas en la política de privacidad de GranaSat (grupo encargado del sistema), la cual establece cómo se recogen, almacenan, procesan y protegen los datos personales. Los usuarios son informados sobre el uso de sus datos y tienen la opción de revisar esta política antes de proporcionar cualquier dato.

A2.3. Medidas de Seguridad Implementadas

Se han implementado diversas **medidas de seguridad** para proteger los datos personales, tales como:

- **Cifrado de datos sensibles** para garantizar la confidencialidad de la información.
- **Autenticación de usuarios** para asegurar que solo los usuarios autorizados tengan acceso a sus propios datos.
- **Control de acceso** al sistema, asegurando que los datos sólo sean accesibles por personas y sistemas autorizados.

A2.4. Protección de Datos en el Código

El sistema ha sido diseñado para que todos los datos personales se almacenan y transmitan de manera segura. El uso de **SSL/TLS** para encriptar las comunicaciones es una de las medidas adoptadas, junto con otras prácticas de seguridad recomendadas en el desarrollo de software para evitar vulnerabilidades.