UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

MEDIDA DE PAR Y VELOCIDAD DEL BANCO DE ENSAYOS DE CONTROL DE MÁQUINAS ELÉCTRICAS BASADO EN UN SISTEMA EMBEBIDO DE BAJO COSTE

Trabajo Fin de Máster presentado por Aridane Álvarez Suárez

Máster en Ingeniería Mecatrónica, Curso académico 2019-2020

Dirigido por Rubén Puche Panadero

y Ángel Sapena Bañó

Medida de par y velocidad del banco de ensayos de control de máquinas eléctricas basado en un sistema embebido de bajo coste

Autor: Aridane Álvarez Suárez

Tutor: Rubén Puche Panadero / Ángel Sapena Bañó

Texto impreso en Valencia, diciembre 2020

Resumen

Este trabajo tiene como objetivo desarrollar un sistema embebido que realice la medida de velocidad y par de un motor asincrónico del banco de ensayos de control de máquinas eléctricas. Este sistema permite la monitorización en el banco de ensayos con una pantalla LCD y mediante la conexión Bluetooth a una aplicación móvil, realizada con el kit de desarrollo Flutter, que facilita la toma, visualización y exportación de mediciones.

Como software complementario, en el trabajo se desarrolla una plataforma para el internet de las cosas con una arquitectura de microservicios, donde los nodos registran sus mediciones y los usuarios de monitorización las visualizan desde una aplicación web SPA.

Abstract

The objectives of this project are the development of an embedded system that measures the speed and torque values of an induction motor at the electrical machines control test bench. These systems allow monitoring the test bench with an LCD screen and with a mobile application connected by the Bluetooth protocol to the embedded system. The application, made with Flutter SDK, eases the measurements, the visualization and the export of the data.

In addition to the embedded system and the mobile application, this project develops an Internet of Things platform built with microservices, where the IoT nodes register their measurements and the monitoring users observe them in an SPA web application.

Índice general

Índice de figuras

ÍNDICE DE FIGURAS

Índice de tablas

Acrónimos

ADC *Analog-to-Digital Converter*

BLE *Bluetooth Low Energy*

DAC *Digital-to-Analog Converter*

GPIO *General Purpose Input Output*

HTL *High Threshold Logic*

IoT *Internet of Things*

LCD Liquid-Crystal Display

MAC *Media Access Control*

PCB *Printed Circuit Board*

SoC *System on Chip*

SPI *Serial Peripheral Interface*

SRAM *Static Random-Access Memory*

TLS *Transport Layer Security*

UART *Universal Asynchronous Receiver-Transmitter*

CAPÍTULO

1

Introducción

En el laboratorio de control de máquinas y accionamientos eléctricos del departamento de Ingeniería Eléctrica de la Universitat Politècnica de València, Campus de Vera, se encuentran unos bancos de prácticas donde se obtienen las mediciones de velocidad y par de un motor asíncrono de inducción, mientras se controla con diferentes modelos de variador de frecuencia y simulaciones de cargas, para su posterior estudio. A continuación, se listan los componentes del banco de ensayo, esquematizado en la **Figura 1.1** y capturado en la **Figura 1.2**.

- **PM:** Motor Sincrónico de imanes permanentes.
- **RES:** Resolver.
- **IM:** Motor Asincrónico de jaula de ardilla (Inducción).
- **ENC:** Encoder incremental.
- **VSD:** Convertidor de frecuencia.
- **CA:** Pinza amperimétrica.
- **M:** Multímetro.

- **RB:** Resistencia frenado.

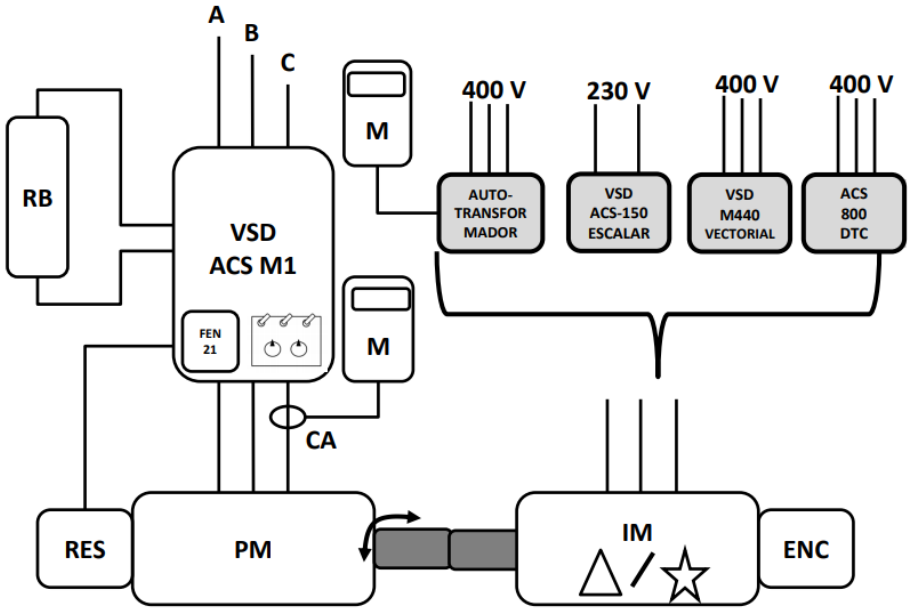

Figura 1.1: Esquema del banco de pruebas del laboratorio

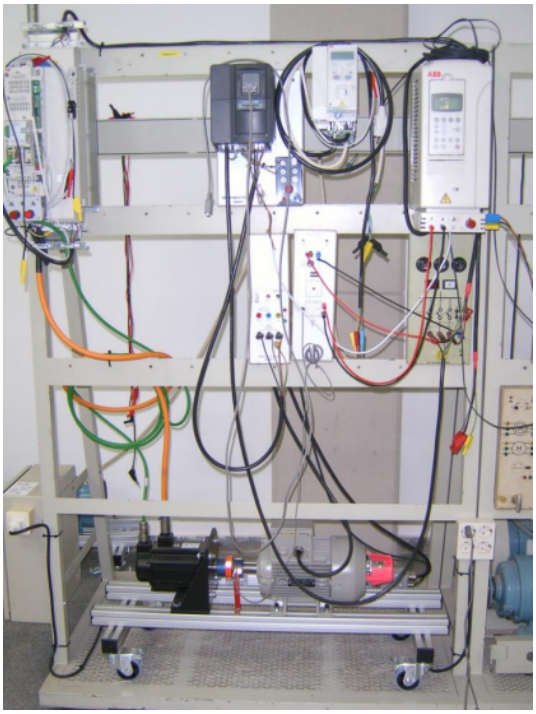

Figura 1.2: Banco de pruebas del laboratorio

El sistema de medición actual consiste en una PCB que integra un microcontrolador PIC de 16 bits, el cual captura la señal del encoder y calcula la velocidad, mostrándola por una pantalla *LCD* 16x2. Además, la relación de esta velocidad con respecto a una referencia se convierte a una señal de salida con rango $0 - 10V$, que es proporcionada con un *DAC* hacia el *ADC* del variador de frecuencia como señal de realimentación para el control en bucle cerrado. Finalmente, con la pinza amperimétrica se obtiene la corriente consumida por el motor de imanes permanentes como una tensión alterna. Esta tensión se lleva a un multímetro y con unos factores de conversión se obtiene el par resistente.

Este trabajo parte con la intención de remplazar el sistema de medición, integrando la lectura del par junto con la velocidad, y con ello prescindir del multímetro, la pinza externa y la posterior conversión manual. Además, se proporcionará el valor del par junto a la velocidad en una nueva pantalla de 20x4. Por otra parte, como alternativa a la toma de datos manual, se desarrollará una aplicación móvil que facilite la tarea de captura y exportación de medidas, recibiendo los datos a partir de una conexión *Bluetooth* con el microcontrolador.

Finalmente, como añadido a la medición local, se desarrollará una plataforma web para la industria 4.0 basada en microservicios.

1.1 Objetivos

El objetivo general del proyecto es el diseño de un sistema embebido para la monitorización del par y velocidad de un motor eléctrico asíncrono de los bancos de prácticas del departamento de Ingeniería Eléctrica, junto a utilidades de software para la monitorización local y remota. Para alcanzar este objetivo, el trabajo se divide en 3 objetivos específicos:

1. Diseño del circuito electrónico y desarrollo del software del microcontrolador para la captura de las medidas de par y velocidad del motor y su emisión a la aplicación móvil o plataforma *IoT*.

2. Diseño y desarrollo de la aplicación móvil para la recepción y exportación de las medidas realizadas por el sistema embebido.

3. Diseño y desarrollo de una plataforma *IoT*, incluyendo microservicios para la visualización de las medidas en tiempo real, muestra del histórico de medidas y creación de avisos personalizados, además de la administración de los dispositivos de la red.

1.2 Estructura

El presente trabajo se organiza en los siguientes capítulos:

1. **Introducción:** Corresponde al actual capítulo, en el que se presenta el trabajo y los objetivos fijados para abordarlo.

2. **Estudio de requisitos:** A partir del análisis del sistema de medición presentado en la introducción, se redacta una especificación de requisitos de manera que se cumplan los objetivos establecidos.

3. **Diseño de la solución:** A partir de los requisitos del anterior capítulo, se diseña el circuito electrónico y las arquitecturas de software de la solución.

4. **Tecnologías y Herramientas:** Antes de llevar a cabo el desarrollo, en este capítulo se introduce las tecnologías y herramientas que se emplearán en el mismo.

5. **Desarrollo de la solución e implantación**: Partiendo de las herramientas, tecnologías y diseño propuesto, se lleva a cabo el desarrollo en este capítulo. Por otra parte, los sistemas desarrollados se lleva a la práctica para su prueba.

6. **Pruebas:** A partir de la especificación de requisitos y la implementación del diseño, se comprueba que se cumplen los objetivos.

7. **Conclusiones:** En el último capítulo se recapitula el trabajo realizado y se obtiene las posibles futuras líneas de trabajo y mejoras.

CAPÍTULO

2

Estudio de requisitos

Como se ha descrito en el **Capitulo 1**, este trabajo tiene como objetivo la sustitución y mejora de un sistema de medida de par y velocidad de un motor eléctrico asincrónico, por lo que, además de los nuevos componentes y requisitos, hay que adaptarse al material ya existente en los bancos de prueba. A continuación, se detallan los diferentes elementos con los que hay que interactuar.

- Por una parte, para la medición de la velocidad del motor asincrónico se haya instalado un encoder incremental 1XP8001-1/1024 de Siemens, **Figura 2.1**.

Figura 2.1: Encoder instalado en el motor de ensayo

Entre sus características, se puede destacar [1]:

  - Alimentación: $+10\,V$ a $+30\,V$.
  - Salida: *Push-pull* (HTL).
  - Pulsos por revolución: 1024.

- Como el par resistente se simula con el motor de imanes permanentes, la medición de este par se obtiene mediante la conversión de la intensidad consumida por el motor de imanes permanentes a una señal de tensión alterna. Para ello, se decide sustituir la pinza amperimétrica que se encuentra en una fase del motor por un transductor de corriente **LTSR 6-NP** en la nueva instalación, **Figura 2.2**.

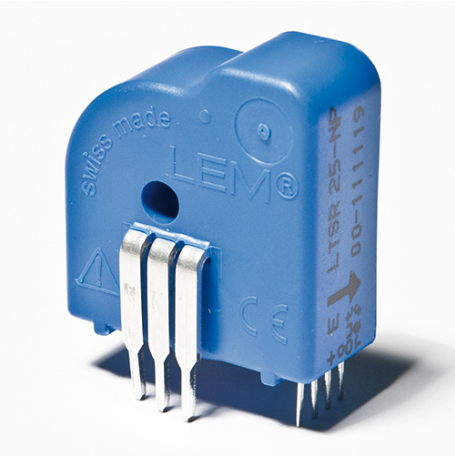

Figura 2.2: Transductor de corriente LTSR 6-NP

Sus características son [2]:

  - Intensidad máxima: $\pm 19{,}2\,A$.
  - Alimentación unipolar: $5\,V$.
  - Señal de salida: $2{,}5 \pm (0{,}625 \cdot I_P/I_{PN})\,V \quad (I_{PN} = 5\,A)$.

- Hasta este momento, en cuanto a la interacción con el usuario, para la realización de prácticas hay instalado un panel, **Figura 2.3**, donde se permite:

  - Realizar el conexionado del motor.
  - Visualizar en un *LCD* la medición de velocidad.

– Obtener la salida analógica de realimentación.
– Conectar la resistencia de frenado.

Figura 2.3: Panel para los ensayos en el banco de prácticas

Para usar junto con el sistema embebido de este trabajo, se ha diseñado un nuevo panel, **Figura 2.5**, donde se ha previsto un espacio para un pulsador y un nuevo *LCD* NHD-0420D3Z-NSW-BBW de Newhaven Display, **Figura 2.4**, con las siguientes características [3]:

– Pantalla con 4x20 caracteres.
– Interfaz *SPI* o *I2C*.

- Alimentación de 5V.

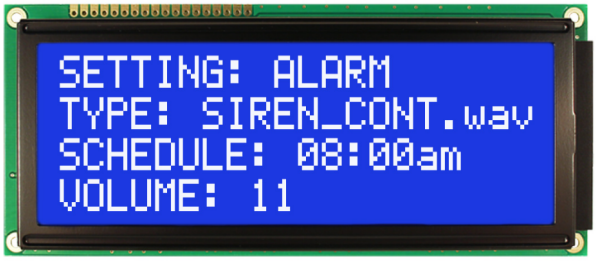

Figura 2.4: *LCD* de Newhaven Display

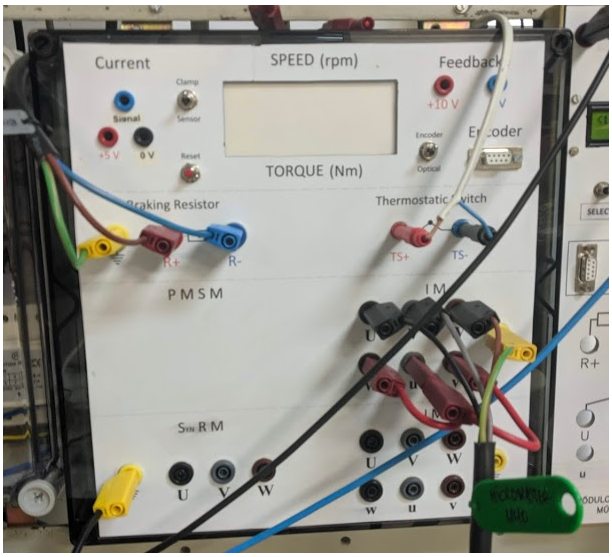

Figura 2.5: Nuevo panel para los ensayos en el banco de prácticas

- Finalmente, para el control en bucle cerrado se emplea el variador de frecuencia Micromaster 440 de Siemens, **Figura 2.6**. De toda su extensa hoja de características [4], el diseño de este trabajo debe tener en cuenta particularmente las de su *ADC*. Los 2 convertidores analógico-digital que incluye el variador de frecuencia pueden operar en los 2 siguientes modos:
  - Entrada unipolar de tensión con rango de $0 - 10\,V$.
  - Entrada unipolar de corriente con rango de $0 - 20\,mA$.

Figura 2.6: Variador de frecuencia Micromaster 440

A partir de los objetivos y teniendo en cuenta las restricciones, a continuación se realiza la especificación del sistema según el estándar IEEE 29148:2018 [5]. Esto permite identificar las necesidades y definir los requisitos detallados desde un punto de vista técnico. Una redacción correcta fija el alcance del proyecto y proporciona la base para el diseño de la arquitectura, desarrollo y final validación.

El trabajo se puede dividir en 3 sistemas suficientemente completos como para su propia especificación de requisitos. El principal abarca el sistema embebido de medición con un módulo software para el microcontrolador y los otros 2 lo conforman la aplicación móvil para la recepción y exportación de medidas y la plataforma IoT, siendo estas dos últimas especificaciones de software.

2.1 Especificación de requisitos del sistema de medición

2.1.1 Propósito

El propósito del desarrollo de este sistema es la sustitución y mejora de un sistema embebido de medición de un motor eléctrico asincrónico en un banco de ensayos.

2.1.2 Alcance

El proyecto tiene como meta obtener el sistema completamente funcional, incluyendo la programación y la electrónica complementaria. Los beneficios de esta mejora son:

- Integrar la medida y visualización del par, el cual es externo en la actualidad.
- Permitir la conexión *BLE* con una aplicación móvil externa.
- Añadir la capacidad de almacenar de manera persistente las mediciones en una plataforma *IoT*.
- Integrar una nueva y mayor pantalla *LCD* en la que se pueda visualizar más información.

2.1.3 Asunciones y dependencias

La mayor parte de los requisitos se obtienen partir del análisis de las necesidades, pero el material del que se parte en el banco de prácticas también afectan a los requisitos. Estos deben tener en cuenta el uso de:

1. *LCD* NHD-0420D3Z-NSW-BBW.
2. Encoder 1XP8001-1/1024.
3. Entrada del *ADC* para realimentación del variador de frecuencia M440.
4. Transductor de corriente LTSR 6-NP.

2.1.4 Vista general

En esta sección se proporciona una visión del sistema a alto nivel.

2.1.4.1 Perspectiva y funciones del sistema

El sistema, **Figura 2.7**, tiene como elemento principal un microcontrolador que, ya sea con componentes internos o externos, es capaz de comunicar con el resto de elementos del entorno con los protocolos de comunicación adecuados y, además, tiene una capacidad de computación suficiente para cumplir los diferentes requisitos funcionales y de rendimiento.

El motor de ensayo se trata del motor de inducción, solidario a este se encuentra un encoder, del cual el microcontrolador obtiene una lectura de velocidad. Unido al eje del motor se sitúa otro motor de imanes permanentes con un transductor de corriente en una de las fases. El microcontrolador calcula el par resistente a partir de la lectura de la onda de salida del transductor.

Con el objetivo de realizar ensayos mediante control de lazo cerrado, desde el microcontrolador se proporciona una señal analógica de realimentación al variador de frecuencia proporcional a la velocidad medida. Finalmente, se provee al usuario las mediciones con una pantalla *LCD* que emplea una comunicación serie.

Los límites del sistema son las conexiones con la aplicación móvil y la plataforma *IoT*, los cuales se definen como 2 sistemas externos.

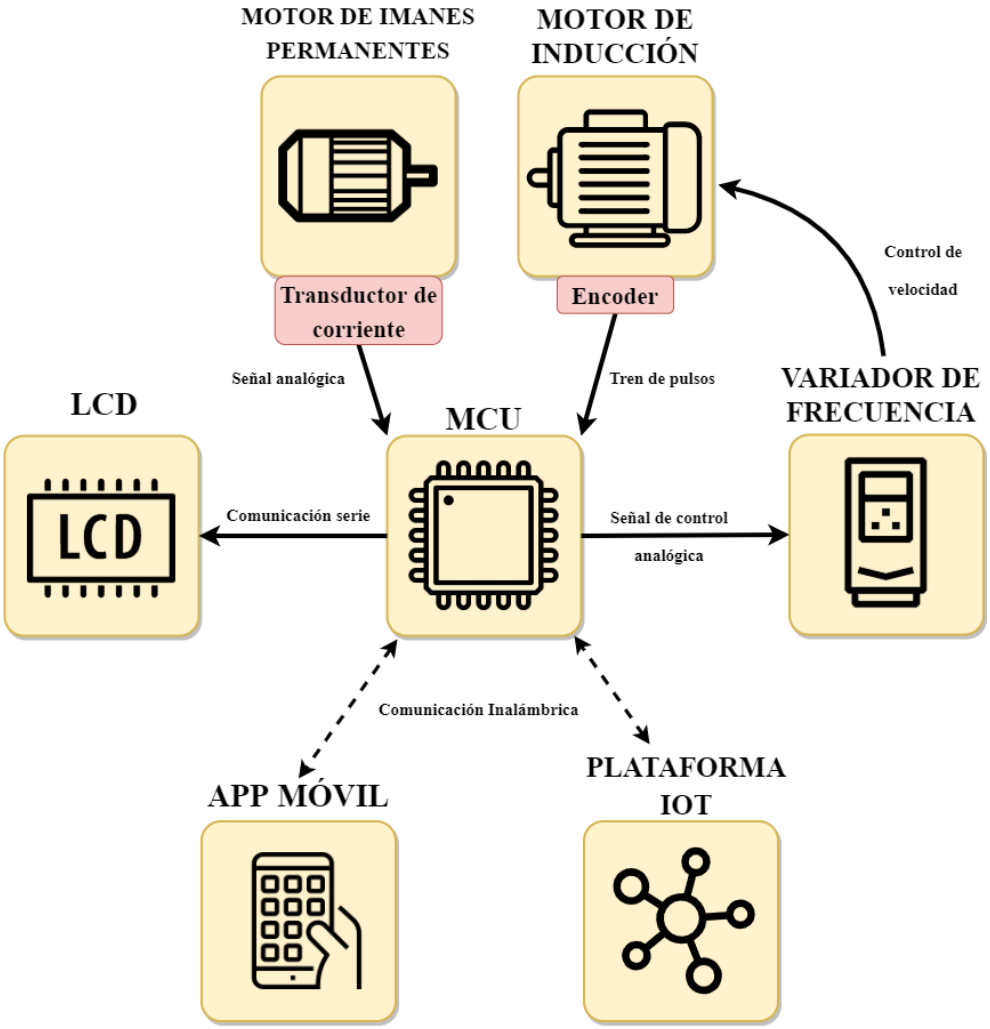

Figura 2.7: Vista general del sistema

2.1.4.2 **Características del usuario**

El sistema está destinado a los alumnos de la UPV que realizan ensayos en el laboratorio de máquinas eléctricas. Los usuarios deben haber adquirido un conocimiento básico sobre el funcionamiento de la instalación y la conexión de sus componentes.

2.1.5 Requisitos del sistema

En esta sección se detallan los diferentes requisitos necesarios para cumplir los objetivos en cuanto a su funcionalidad y la interacción con los componentes externos al sistema, incluyendo los propios usuarios.

2.1.5.1 Requisitos funcionales

Estos requisitos describen salidas que debe tener el sistema a partir de una acción sobre las entradas.

Identificador	Descripción
RF-01	Con la entrada de la señal del encoder, el sistema debe capturar y contar el número de pulsos durante un tiempo determinado y obtener un valor de velocidad.
RF-02	Con la entrada de la señal del transductor de corriente, el sistema debe leer valores instantáneos durante un tiempo determinado, calcular su valor eficaz y convertirlo a un valor de par resistente.
RF-03	A partir de la velocidad calculada, el sistema debe escalar el valor al intervalo de lectura del *ADC* del variador de frecuencia y proporcionar una señal analógica.
RF-04	El sistema debe alimentar el *LCD*, el *MCU* y el encoder adaptando una entrada de 12V.

Tabla 2.2: Requisitos funcionales del sistema

2.1.5.2 Requisitos de usabilidad

Los requisitos de usabilidad describen las necesidades en cuanto a la interacción del usuario con el sistema.

Identificador	Descripción
RU-01	El sistema debe enviar los valores de par y velocidad calculados a la pantalla *LCD* con un periodo máximo de $500\,ms$ para su correcta visualización.
RU-02	El sistema debe enviar los valores de par y velocidad calculados a la aplicación móvil con un periodo máximo de $500\,ms$.
RU-03	El sistema debe proporcionar al usuario la capacidad de habilitar o deshabilitar la conexión inalámbrica.
RU-04	El sistema sólo debe permitir un usuario conectado a este con la aplicación móvil simultáneamente.

Tabla 2.4: Requisitos de usabilidad del sistema

2.1.5.3 **Requisitos de interfaz**

Con estos requisitos se definen las capacidades que debe tener el sistema para permitir la interoperabilidad con los otros elementos fuera de los límites de este.

Identificador	Descripción
RI-01	El microcontrolador del sistema debe tener compatibilidad con comunicaciones *Wi-Fi* y *Bluetooth Low Energy* para la conexión con la plataforma *IoT* y la aplicación móvil respectivamente.
RI-02	Mientras el sistema esté conectado a la aplicación móvil, debe poder recibir la configuración de parámetros desde ella.
RI-03	Mientras el sistema esté conectado a la plataforma *IoT*, debe enviar los valores de par y velocidad con un periodo máximo de $500\,ms$.
RI-04	Mientras el sistema esté conectado a la aplicación móvil, debe enviar los valores de par y velocidad con un periodo máximo de $500\,ms$.

Tabla 2.6: Requisitos de interfaz del sistema

2.1.5.4 Estados y modos del sistema

Teniendo en cuenta los posibles modos del sistema surgen otros requisitos del mismo.

Identificador	Descripción
REM-01	El sistema tendrá 3 modos que puedan funcionar en cualquier combinación: Muestra de parámetros en *LCD*, envío de lecturas a la aplicación móvil y envío de lecturas a la plataforma *IoT*.
REM-02	El sistema debe mostrar el estado de la conexión con la aplicación móvil.

Tabla 2.8: Requisitos de estados y modos del sistema

2.2 Especificación de requisitos de software de la aplicación móvil

2.2.1 Propósito

El propósito del desarrollo de este software es proporcionar un complemento al sistema embebido de medición para facilitar la toma de mediciones en los ensayos.

2.2.2 Alcance

El software tiene como objetivo facilitar y reducir el tiempo de la toma de medidas al realizar prácticas en el laboratorio de máquinas eléctricas. Los beneficios de esta aplicación son:

- Lectura en tiempo real de los valores de velocidad y par del motor desde un dispositivo móvil.
- Permitir la configuración de parámetros del sistema de medición sin cambiar el *firmware* de este.
- Dibujar los valores tomados en gráficas.
- Exportar mediciones en una hoja de cálculo.

2.2.3 Vista general

En esta sección se proporciona una visión del sistema a alto nivel.

2.2.3.1 Contexto del producto

Este software se relaciona con el sistema embebido de medición, el hardware donde funciona y el usuario. La relación con estos límites se realiza de la siguiente manera:

1. Interfaces del sistema: En cuanto al sistema embebido, el software debe transferir datos mediante comunicación inalámbrica con este de manera bidireccional. En la **Figura 2.7** ya se ha mostrado la situación de este software en el sistema global.

2. Interfaces de usuario: La interfaz gráfica de usuario debe tener mínimo dos pantallas en las que se muestre los valores recibidos por el sistema de medición en tiempo real y visualizarlos en gráficas con respecto al tiempo.

3. Interfaces de hardware: Como aplicación móvil, el software debe ser compatible con dispositivos ARM64 y x86-64 con el sistema operativo Android.

2.2.3.2 Funciones del producto

Las funcionalidades del software se describen en el siguiente diagrama de casos de uso:

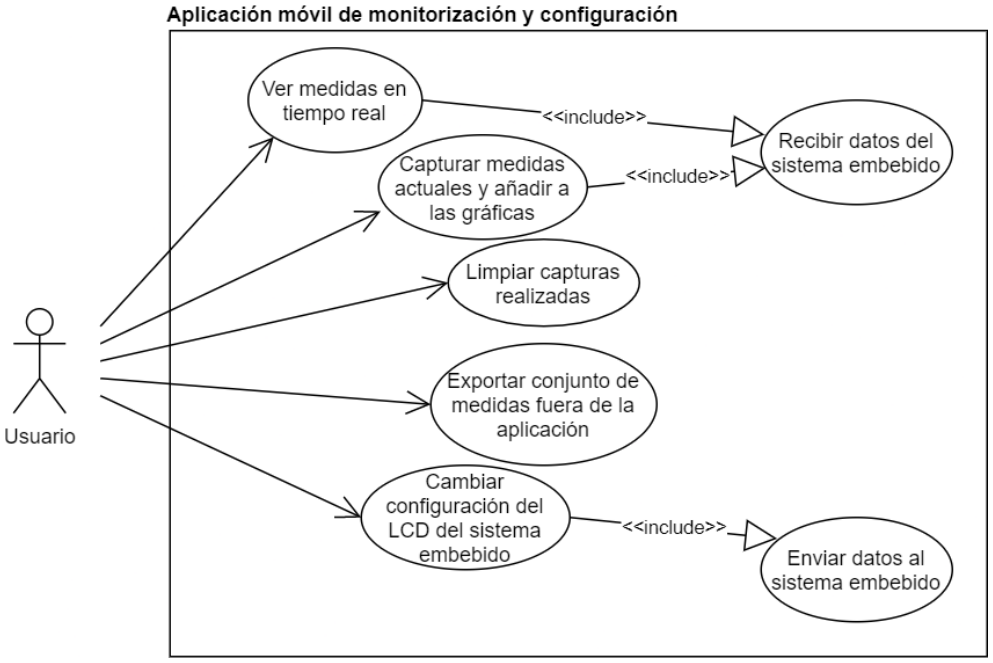

Figura 2.8: Diagrama UML de casos de uso de la aplicación móvil

2.2.3.3 **Características del usuario**

El software está destinado a los alumnos de la UPV que realizan ensayos en el laboratorio de máquinas eléctricas, la aplicación deberá tener un funcionamiento sencillo que no requiera formación para su uso.

2.2.4 **Requisitos del sistema**

En esta sección se detallan los diferentes requisitos necesarios para cumplir los objetivos en cuanto a su funcionalidad y la interacción con los componentes externos al sistema, incluyendo los propios usuarios.

2.2.4.1 **Requisitos funcionales**

Estos requisitos describen salidas que debe tener el sistema a partir de una acción sobre las entradas.

Identificador	Requisito	Descripción
RF-01	Medidas en tiempo real	Cuando el usuario seleccione las medidas en tiempo real desde el menú, la aplicación deberá recibir los datos del sistema embebido y mostrar contadores en tiempo real.
RF-02	Capturar medidas actuales	Cuando el usuario seleccione la captura de datos, la aplicación debe guardar las medidas de ese instante y mostrarlas en una gráfica temporal.
RF-03	Eliminar capturas	Cuando el usuario seleccione eliminar las capturas, la aplicación deberá borrar las capturas de las gráficas.
RF-04	Exportar medidas capturadas	Cuando el usuario seleccione exportar las medidas, la aplicación deberá recoger las medidas tomada, reunirlas en un archivo con valores separados por coma (*CSV*) y permitir exportar a la aplicación deseada.
RF-05	Configurar parámetros del sistema	Cuando el usuario modifique la posición de *sliders* que controlen el brillo y contraste del *LCD* del sistema embebido, la aplicación deberá tomar los valores y enviarlos al sistema embebido.

Tabla 2.10: Requisitos funcionales del software

2.3 Especificación de requisitos de software de la plataforma *IoT*

2.3.1 Propósito

El propósito del desarrollo de este software es obtener una plataforma basada en tecnologías web que permita registrar y visualizar en tiempo real mediciones del sistema embebido desarrollado, además de otros sistemas de la industria con la misma capacidad de comunicación.

2.3.2 Alcance

Este software complementa el sistema embebido de medición, sin ser necesario para su normal funcionamiento. Tiene como objetivo obtener una plataforma fácilmente ampliable a nuevos nodos de medición con la capacidad de administrar los nodos y usuarios. El software tiene una componente *front end* que proporciona una capa de presentación al usuario y otra *back end* que realiza la parte lógica y de almacenamiento de datos.

2.3.3 Vista general

En esta sección se proporciona una visión del sistema a alto nivel:

2.3.3.1 Contexto del producto

Este software se relaciona con el sistema embebido de medición, el hardware donde funciona y el usuario. La relación con estos límites se realiza de la siguiente manera:

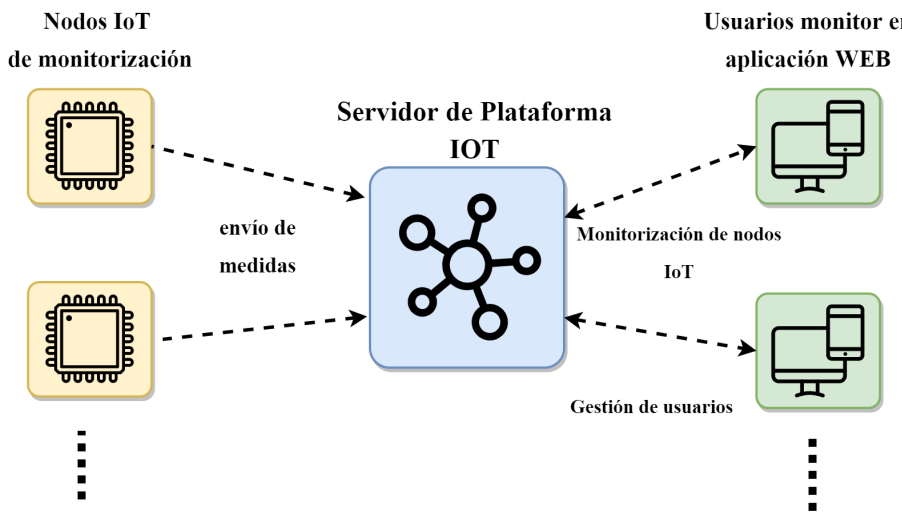

Figura 2.9: Contexto del sistema

1. Interfaces del sistema: En cuanto al sistema embebido, el software debe recibir las medidas tomadas por los nodos *IoT* mediante comunicación inalámbrica.

2. Interfaces de usuario: La interfaz gráfica del sistema la proporciona el componente *front end* mediante una aplicación web a la que accede el usuario de monitorización desde un ordenador con conexión a Internet.

3. Interfaces de hardware: El componente *back end* del sistema se ejecutará en un equipo x86-64 con sistema operativo Linux.

2.3.3.2 **Funciones del producto**

Las funcionalidades del software se describen en el siguiente diagrama de casos de uso:

Figura 2.10: Diagrama UML de casos de uso de la plataforma IoT

Por una parte, los nodos *IoT* se dedican a enviar medidas a la plataforma, mientras que los usuarios monitor acuden a servicios para tratar con estos datos. Un usuario administrador tiene permisos sobre todos los demás usuarios, además de tener la capacidad de registrar y eliminarlos, asignando permisos de los usuarios monitor sobre los usuarios nodo *IoT*. De esta manera, los usuarios monitor comunes podrán utilizar los servicios sobre los nodos *IoT* de los cuales tengan permisos.

2.3.3.3 **Características del usuario**

El usuario del software puede ser un administrador que se encargue de la gestión de la plataforma o un usuario al que se le asigna unos nodos *IoT* concretos para mo-

nitorizar. El software debe ser intuitivo para una persona sin formación específica.

2.3.4 Requisitos del sistema

En esta sección se detallan los diferentes requisitos necesarios para cumplir los objetivos en cuanto a su funcionalidad y la interacción con los componentes externos al sistema, incluyendo los propios usuarios.

2.3.4.1 Requisitos funcionales

Estos requisitos describen salidas que debe tener el sistema a partir de una acción sobre las entradas.

Tabla 2.11: Requisitos funcionales del software

Identificador	Requisito	Descripción
RF-01	*Log in*	Cuando el usuario monitor introduzca sus credenciales en una pantalla de *log in*, el sistema deberá validar las entradas de texto y comprobar si la combinación usuario-contraseña es correcta, dando acceso a la pantalla principal o avisando del fallo.
RF-02	*Log out*	Cuando el usuario monitor seleccione salir de la cuenta, la interfaz deberá eliminar la sesión y mostrar la pantalla de *log in*.
RF-03	Recibir medidas	Cuando un nodo *IoT* registrado envíe medidas a la plataforma, el sistema debe almacenarlas con su identificador, valor y tiempo.

(Continua)

Identificador	Requisito	Descripción
RF-04	Visualizar nodos *IoT* registrados	Cuando un usuario monitor acceda en el menú a los usuarios registrados, el sistema deberá obtener los usuarios sobre los que tiene permiso, comprobar la información de estos y mostrarlos.
RF-05	Registrar nuevo usuario	Cuando un usuario administrador introduzca las credenciales y tipo de un nuevo usuario, el sistema deberá comprobar la validez y crear una nueva entrada en la base de datos o devolver error.
RF-06	Eliminar usuario	Cuando un usuario administrador seleccione eliminar un usuario, el sistema deberá comprobar la validez y eliminar las entradas de la base de datos o devolver error.
RF-07	Visualizar medidas en tiempo real	Cuando el usuario monitor acceda en el menú a las medidas de tiempo real, el sistema deberá comprobar los permisos y mostrar las medidas que se reciben de los nodos *IoT* correspondientes.
RF-08	Visualizar medidas en un intervalo de tiempo	Cuando el usuario monitor acceda en el menú a las medidas introduciendo 2 fechas, el sistema deberá comprobar los permisos y mostrar, en una gráfica temporal, las medidas enviadas entre las 2 fechas por el nodo *IoT* correspondiente.

(Continua)

Identificador	Requisito	Descripción
RF-09	Visualizar historial de alertas	Cuando el usuario monitor acceda en el menú a las alertas de nodos *IoT*, el sistema deberá comprobar los permisos y mostrar las alertas de los nodos *IoT* correspondientes.
RF-10	Crear aviso	Cuando el usuario monitor rellene y envíe los campos de entrada, el sistema deberá comprobar los campos, los permisos y registrar un nuevo aviso o devolver error.
RF-11	Visualizar avisos	Cuando el usuario monitor acceda en el menú de avisos, el sistema deberá comprobar los permisos y mostrar los avisos registrados.
RF-12	Eliminar aviso	Cuando el usuario monitor seleccione uno de los avisos en la vista de avisos, el sistema deberá comprobar los permisos y eliminar los campos de la base de datos correspondientes.

2.3.4.2 **Requisitos de base de datos**

En este software se identifican dos tipos de datos a almacenar. Debido a esto, los datos comunes de información de usuarios se deben almacenar en bases de datos relacionales, mientras que las medidas de los nodos *IoT* en una base de datos de series temporales.

2.3.4.3 **Atributos del sistema software**

En cuanto a seguridad del software, todas las transferencia por red deberán utilizar el protocolo criptográfico *Transport Layer Security* (*TLS*) para proporcionar

seguridad a estas.

CAPÍTULO

3

Diseño de la solución

Tras enumerar los requisitos derivados de los objetivos del proyecto, en este capítulo se obtiene la arquitectura de cada sistema. Este diseño servirá de base para la elección de las tecnologías en el **Capítulo 4** y su desarrollo en el **Capítulo 5**. A cada uno de los 3 sistemas por desarrollar se le dedica una sección por separado.

3.1 Sistema embebido de medición

A este sistema, teniendo en cuenta que se trata de un sistema embebido, además de la arquitectura y desarrollo del software, se le dedicará una sección al diseño del circuito electrónico que permite enlazar los elementos externos con el microcontrolador.

3.1.1 Diseño del circuito electrónico

Como elemento principal del sistema tenemos un *SoC* ESP32, el cual viene integrado en una placa de desarrollo *ESP32 devkitc v4*.

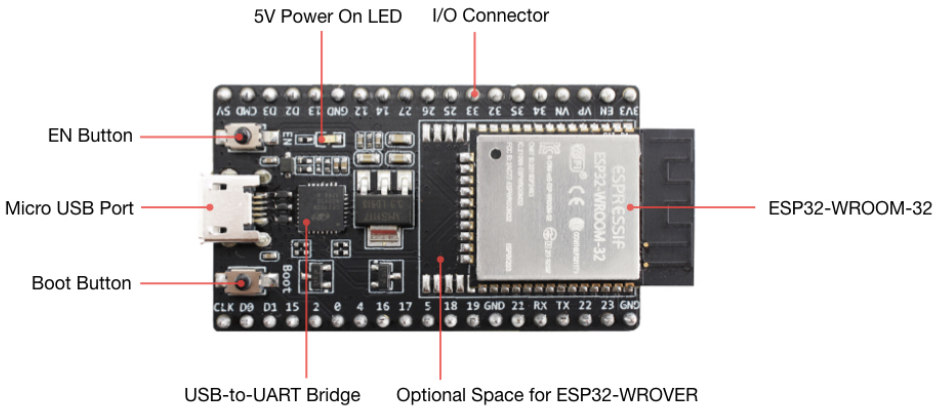

Figura 3.1: ESP32 devkitc v4

El ESP32 integra en un *SoC* de bajo coste y consumo dos núcleos *Xtensa LX6* (set de instrucciones *Xtensa* de *Tensilica*), un conjunto de periféricos con posibilidad de multiplexado a cualquier pin, memoria volatil *SRAM* y no volatil *FLASH*. Las características diferenciadoras de este microcontrolador son la inclusión de conectividad inalámbrica *Wi-Fi* 2.4Gz y *Bluetooth Low Energy*. Además, el fabricante proporciona las librerías ESP-IDF (*IoT Development Framework*) junto con una abstracción de esta a la plataforma Arduino.

La lista de caracteríscas se puede ver a continuación. [6]

- Alimentación de 3.3V.
- 2 nucleos Xtensa de 32 bit hasta 240MHz.
- *Wi-Fi* 2.4GHz.
- *Bluetooth Low Energy*.
- Ethernet *MAC*.
- 512 KB de *SRAM*.
- 4MB de *FLASH*.

- 34 *GPIO*.
- 4 x *SPI*.
- 2 x *I2C*.
- 3 x *UART*.
- 2 x *DAC* 8 bit.
- *ADC* de 12 bit hasta 18 canales.
- Contador de pulsos.

En cuanto al kit de desarrollo, al microcontrolador se le une un encapsulado que efectúa un puente *UART-USB* para la programación y *debugging*, un regulador de tensión lineal de 5V a 3.3V y un puerto microUSB.

A continuación, se describe cada parte del circuito necesaria para la conexión del microcontrolador con el resto de componentes mostrados en la **Figura 2.7**.

3.1.1.1 **Microcontrolador ESP32**

En la siguiente imagen se pueden ver los pines seleccionados que serán posteriormente utilizados por los diferentes subcircuitos. Además, el ESP-32 DevKit recibe una alimentación de $5\,V$, que es adaptada a $3,3\,V$ con un regulador *Low-dropout* (*LDO*).

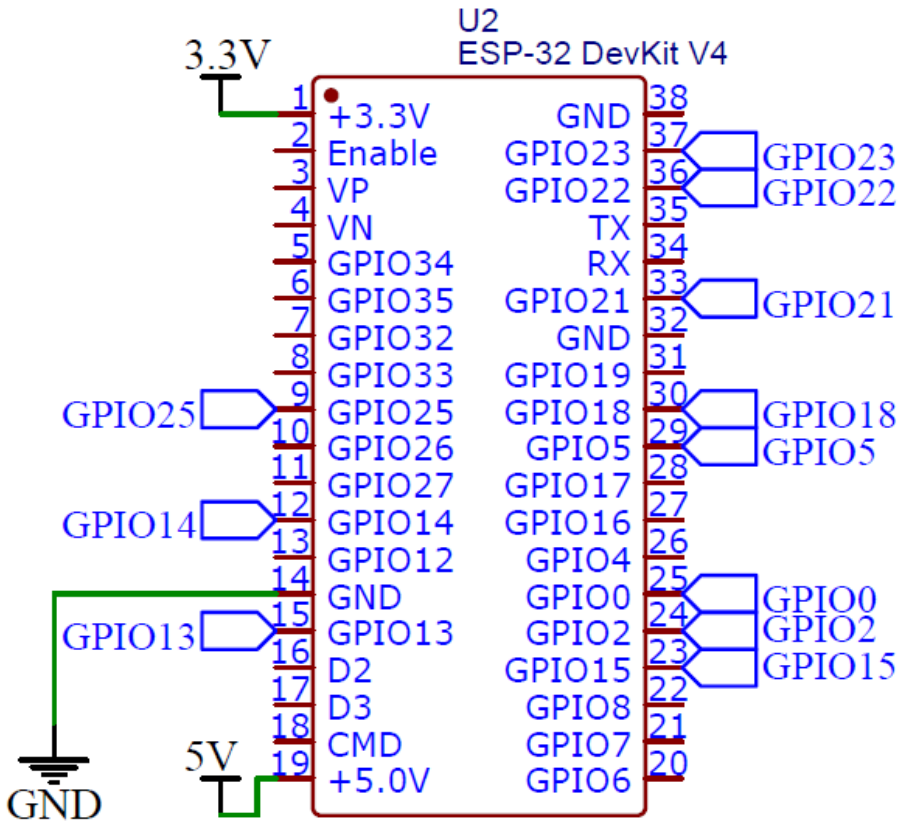

Figura 3.2: Conexiones de ESP32

3.1.1.2 *ADC* del variador de frecuencia

Desde el microcontrolador se proporciona una señal analógica al variador de frecuencia M440 de Siemens, el cual tiene un *ADC* con un rango de tensión de $0 - 10V$. [4]

A pesar de que el microcontrolador incluye *DAC* integrado, se ha decidido emplear un *DAC* externo de MICROCHIP, modelo MCP4922 con interfaz *SPI*. Con este encapsulado pasamos de los 8 bits del ESP32, 256 valores posibles, a un *DAC* de 12 bits con 4096, con el que, mediante una alimentación de $5V$, obtenemos una resolución de $1,22\,mV$ antes de la amplificación y $2,44\,mV$ después de la misma. [7]

El *DAC* MCP4922 tiene una conexión *SPI* con el microcontrolador y utiliza

una alimentación de $5V$ con 2 condensadores en paralelo de *bypass* para filtrar ruido de alta frecuencia. Estos condensadores son cerámico $(0,1\,\mu F)$ y de tántalo $(10\,\mu F)$, valores obtenidos de la recomendación del fabricante [7]. Por otra parte, el pin $\overline{SHDN}$ se mantiene en alto para activar la conversión y el pin $\overline{LDAC}$ en bajo para que se realice la conversión de manera inmediata desde que se escribe en el registro. Finalmente, la salida de uno de los *DAC* del MCP4922 se lleva a un amplificador operacional TLC272 de Texas Instruments. **Figura 3.3**.

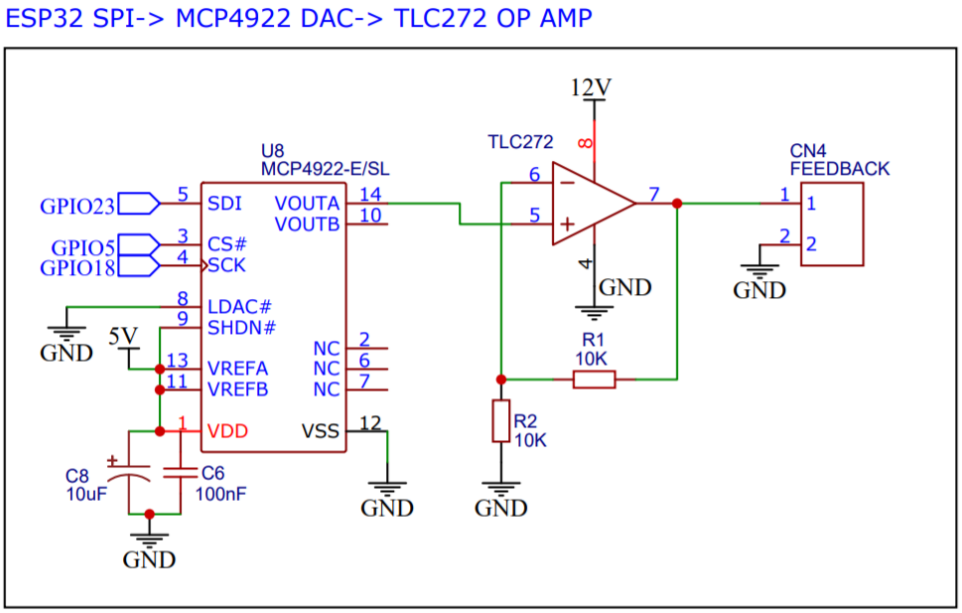

Figura 3.3: Subcircuito del *DAC* y amplificador operacional

El amplificador operacional tiene una configuración simple de amplificador no inversor, que lleva el rango de tensión de $0-5V$ a $0-10V$ necesario para el *ADC* del variador de frecuencia. Con estos valores, se obtiene la siguiente ganancia:

- A_v: Ganancia del amplificador.
- V_i: Tensión de entrada.
- V_o: Tensión de salida.
- $R1, R2$: Resistencias de realimentación.

$$V_o = A_v \cdot V_i \tag{3.1}$$

$$A_v = \frac{10}{5} = 2 \tag{3.2}$$

Para obtener los valores de las resistencias de realimentación se utiliza la siguiente expresión:

$$A_v = 1 + \frac{R_2}{R_1} \tag{3.3}$$

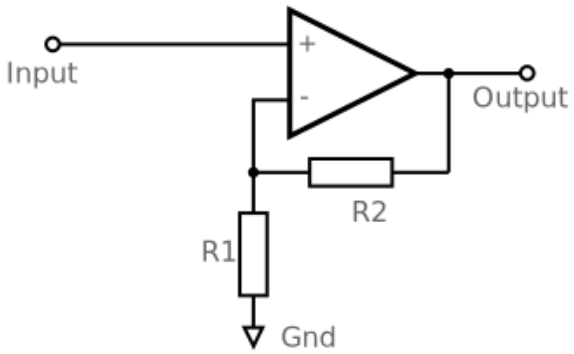

Figura 3.4: Amplificador operacional en configuración amplificador no inversor

Con una pareja de resistencias de $10K\Omega$ obtenemos el valor correcto de ganancia:

$$A_v = 1 + \frac{10}{10} = 2 \tag{3.4}$$

En cuanto a la alimentación, el amplificador TLC272 tiene una tensión máxima de $16\,V$ y se le aplicará $12\,V$, teniendo margen de $2\,V$ para la tensión de salida de $10\,V$. [8]

3.1.1.3 **Encoder**

La señal de salida que proporciona el encoder de Siemens, con una tensión de alimentación de $12\,V$, corresponde a una señal cuadrada de $12\,V$ con una corriente máxima de carga de $200\,mA$ [1].

El microcontrolador trabaja con una tensión de $3{,}3\,V$, por lo que se ha implementado un circuito para adaptar el nivel la señal del encoder a la entrada digital del ESP32.

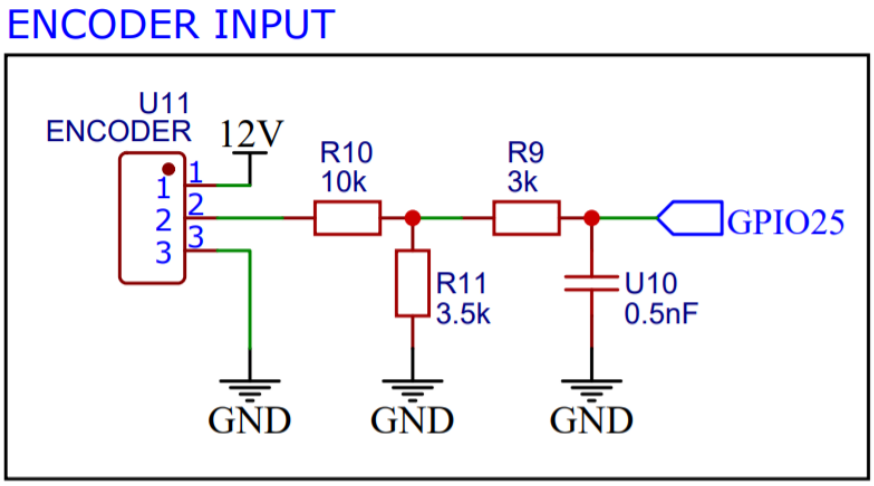

Figura 3.5: Subcircuito de la entrada del encoder

El circuito consta de dos etapas, un divisor de tensión y un filtro pasivo:

- **Divisor de tensión:** La tensión de $12\,V$ se debe adaptar a la entrada del microcontrolador, para ello se acude al manual del ESP32 [6], de donde se obtiene que, con una alimentación de $V_{DD} = 3{,}3\,V$, el nivel mínimo de entrada para un valor lógico alto es de $V_{IH} = 0{,}75 \cdot V_{DD}\,V = 2,475\,V$. Esto nos da un intervalo deseado para el valor alto de la señal digital de entrada $V_{IH} - V_{DD}$.

  El siguiente paso es elegir los valores de la resistencias que forman el divisor de todos los que cumplen la relación entrada-salida. Para ello hay que tener cuenta que el divisor de tensión, por si sólo, no tiene ninguna utilidad, sino que necesita una carga conectada. En este caso, un pin de un microcontrolador llevado a modo entrada digital con alta impedancia con su multiplexor interno, tiene una resistencia en el intervalo de $100k\Omega - 1M\Omega$. En base a esto, la pareja de resistencias deben tener unos valores que eviten los dos extremos:

  1. Valores muy bajos de las resistencias del divisor consumirán demasiada corriente, disipando esa energía en forma de calor.

2. Por el contrario, valores muy altos que se acerquen a la resistencia de carga harán que la resistencia formada en paralelo con R_2, **Figura 3.6**, empiece afectar a la caída de tensión, obteniendo un valor menor del calculado en el divisor. Este valor puede llegar a ser menor que V_{IH}, impidiendo que el microcontrolador detecte el nivel alto de señal del encoder.

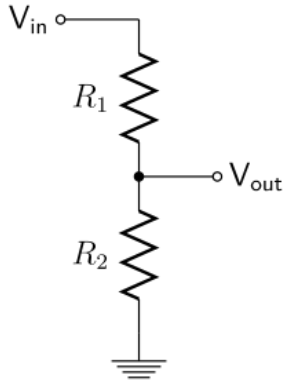

Figura 3.6: Circuito del divisor de tensión

Tras la prueba experimental de diferentes valores, se han fijado dos resistencias de $R_1 = 10k\Omega$ y $R_2 = 3{,}5k\Omega$

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in} = 3{,}11\,V \tag{3.5}$$

Teniendo como peor caso una resistencia de entrada del pin del microcontrolador de $R_L = 100k\Omega$, la resistencia $R_2//R_L$ tendría un valor de $3{,}38k\Omega$, lo que sólo bajaría la tensión de salida del divisor a $3{,}03\,V$. Este valor de señal alta se considera correcto para el intervalo ya mencionado, $2,475 - 3{,}3\,V$.

Por otro lado, la potencia disipada por cada resistencia se obtiene a continuación:

– Corriente consumida por todo el circuito:

$$V_{in} = V_{R1} + V_{R2-R_L} = I_T \cdot (R_1 + R_2//R_L) \tag{3.6}$$

$$I_T = \frac{12}{10000 + 3380} = 0,896\,mA \tag{3.7}$$

– Potencia disipada por la primera resistencia:

$$P_{R_1} = I_{R_1}^2 \cdot R_1 = 8\,mW \tag{3.8}$$

– Potencia disipada por la segunda resistencia:

$$V_{R_1} = I_T \cdot R_1 = 8{,}96\,V \qquad V_{R_2} = V_{in} - V_{R_1} = 12 - 8{,}96 = 3{,}04\,V \tag{3.9}$$

$$P_{R_2} = \frac{V_{R_2}^2}{R_2} = \frac{3{,}04^2}{3500} = 2{,}62\,mW \tag{3.10}$$

Se puede concluir que, con estos valores de resistencias, se cumplen las dos condiciones de bajo consumo y nivel de tensión de salida correcto sin que afecte la resistencia de entrada del microcontrolador.

- **Filtro pasivo:** Una vez convertida la onda cuadrada a una tensión adecuada para el microcontrolador, nos encontramos con un ruido que es lo suficientemente alto para producir picos de tensión confundibles con pulsos para el microcontrolador. Debido a esto, después del divisor de tensión se sitúa un filtro pasivo paso bajo de primer orden.

  A través de pruebas experimentales, se ha obtenido que con una frecuencia de corte de $106{,}1\,kHz$ se reduce el ruido de la onda sin llegar a ralentizar el flanco de subida en las frecuencias más altas de trabajo ($26\,kHz$).

  Para elegir los valores de los componentes pasivos, primero se fija la resistencia para mantener la integridad de la señal según la resistencia de entrada y salida del circuito.

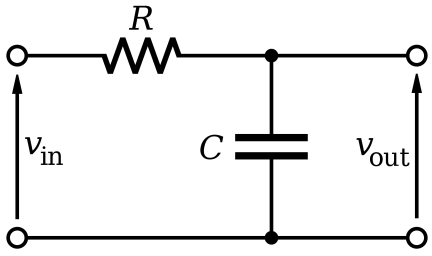

Figura 3.7: Circuito de filtro paso bajo de primer orden

El circuito del filtro se encuentra entre el divisor de tensión y la entrada digital del microcontrolador. Considerando la salida de señal del encoder como una fuente de tensión ideal, la resistencia de entrada corresponde a las dos resistencias del divisor en paralelo, $2{,}592k\Omega$. Por otra parte, hay que tener en cuenta que la resistencia del filtro se une en serie a la resistencia de entrada del microcontrolador, la cual ya se ha visto que tiene un valor suficiente para no alterar la señal. Debido a esto vemos que, en cuanto a la resistencia de entrada, hay libertad en elegir el valor para el filtro.

En cuanto a la resistencia de salida, considerando el peor caso de resistencia de la entrada del microcontrolador de $R_L = 100k\Omega$, la resistencia del filtro tiene que ser varias magnitudes inferior para no generar un error de divisor de tensión.

Finalmente, para el filtro que se observa en la **Figura 3.7** se han seleccionado los valores de $R = 3\,k\Omega$ y $C = 0{,}5\,nF$.

$$f_c = \frac{1}{2 \cdot \pi R \cdot C} = 106{,}1\,kHz \tag{3.11}$$

3.1.1.4 Transductor de corriente

La señal parte del motor de imanes permanentes, del cual se obtiene una intensidad de pico $I_p = 43,3\,A$ y una intensidad nominal de $I_n = 14,4\,A$. Por otro lado, el transductor de corriente LTSR 6-NP tiene una intensidad de sobrecarga de $I_p = 250\,A$, lo que hace innecesario protección externa para el pico de corriente del motor. Además, su rango de medición $I_{pm} = \pm 19{,}2\,A$ también cubre los valores que dará el motor en los ensayos [2].

En sensor se alimenta con una tensión de $5\,V$ y en la salida se obtiene una tensión de $2{,}5 \pm (0{,}625 \cdot I_p/I_{pn})\,V$, con $I_{pn} = 6\,A$. Esta señal entra en el intervalo de medición $0 - 5\,V$ del *ADC* externo, ADS1015 de Texas Instruments.

El *ADC* seleccionado tiene una tensión de alimentación de $5\,V$. Además, este *ADC* tiene una resolución de 12-bit y una frecuencia de muestreo de 3300 *SPS* [9].

El motor de inducción de ensayo se utiliza a una velocidad de $0 - N_n$ ($0 - 1420\,rpm$) y el motor de imanes permanentes se encuentra solidario con una frecuencia nominal de $F_n = 200\,Hz$ a su velocidad nominal $N_n = 3000\,rpm$. Esto

resulta en una velocidad máxima del motor de imanes permanentes de $1500\,rpm$ y frecuencia máxima de $100\,Hz$. La frecuencia de muestreo del *ADC* es más de 33 veces el valor de esta, por lo que se supera la frecuencia de Nyquist.

El ADS1015 se utiliza en una placa de desarrollo. la cual incluye el circuito recomendado por el fabricante con las resistencias de las entradas y el condensador de *bypass* junto a la alimentación del circuito integrado.

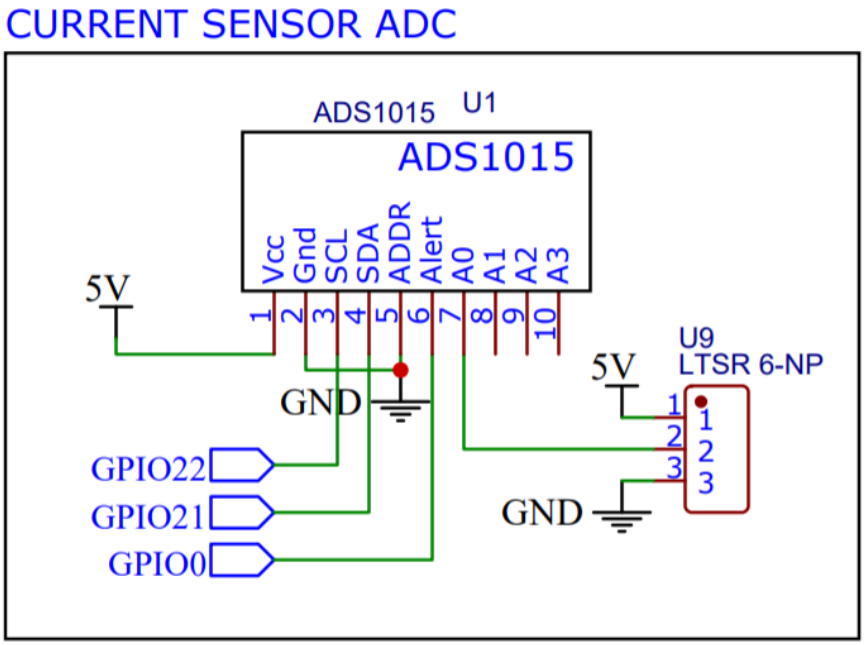

Figura 3.8: Subcircuito de la entrada de la pinza amperimétrica

3.1.1.5 **LCD**

El *LCD* de Newhaven Display tiene una alimentación de $5\,V$ y permite la comunicación por *SPI* ($100\,KHz$) o *I2C* ($50\,KHz$). Por la mayor velocidad se ha decidido emplear comunicación *SPI*, para la cual hay que realizar un corto en R2 y dejar abierto R1 en el *PCB* del *LCD*. [3]

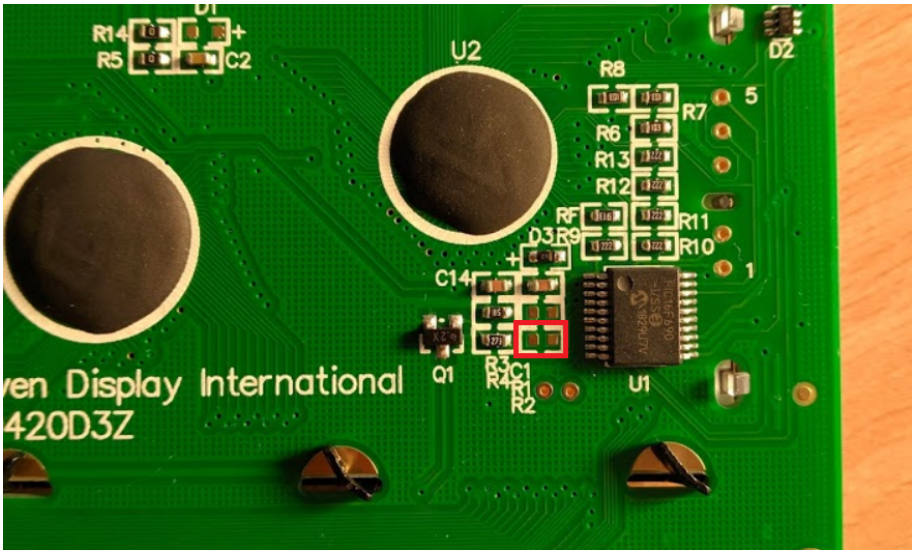

Figura 3.9: Configuración del *LCD* para protocolo *SPI*

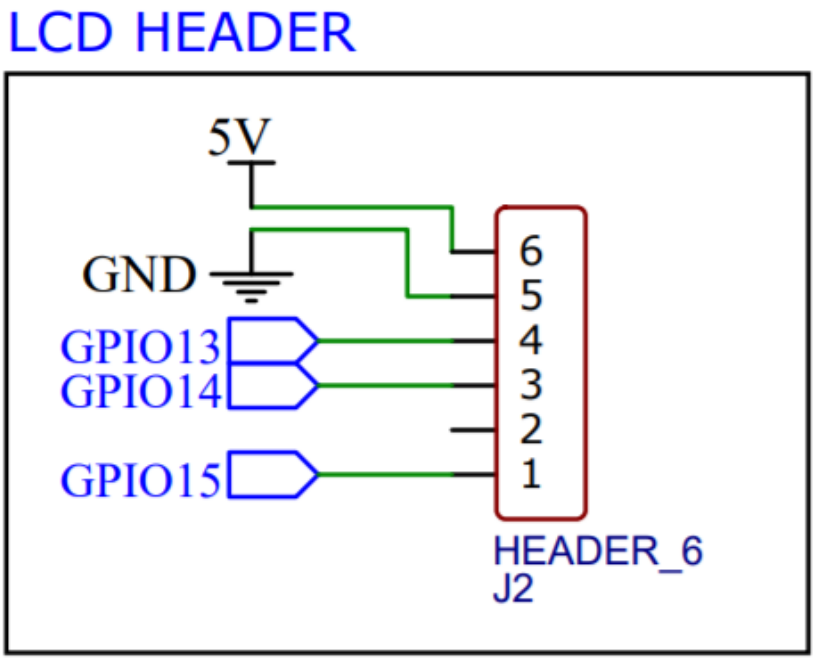

Figura 3.10: Subcircuito del conector del *LCD*

3.1.1.6 Pulsador para el control de la conexión *Bluetooth*

Para activar y desactivar la conexión *Bluetooth* del microcontrolador ESP32, en la caja de instalación se situará un pulsador. Esta conexión lleva uno de los terminales del botón a tierra y el otro a un pin del microcontrolador, el cual activará una resistencia de *pull-up* interna vía Software.

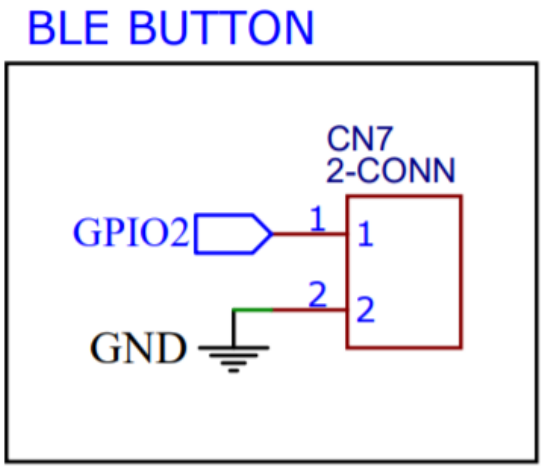

Figura 3.11: Subcircuito del pulsador

3.1.1.7 Alimentación

En los anteriores circuitos se han podido ver dos alimentaciones, una de $12\,V$ y otra de $5\,V$. Para obtener estas tensiones se parte de una alimentación continua externa de $12\,V$ como entrada al circuito. El amplificador operacional recibe esta tensión, mientras que el resto de módulos la obtienen de un convertidor reductor DC-DC, modelo MP1584EN con las siguientes características [10]:

- Rango de tensión de entrada: $4,5 - 28\,V$.
- Rango de tensión de salida: $0,8 - 25\,V$.
- Corriente de salida: $3\,A$.

Los componentes con mayor consumo son el microcontrolador y la pantalla *LCD*, con unos valores de:

- ESP32: $240\,mA$ como máximo en la transmisión con el protocolo *Wi-Fi* 802.11b [6].
- *LCD* NHD: Máximo de $44\,mA$.

Por otra parte, el resto de componentes tienen un consumo insignificante como los $0,3\,mA$ del ADS1015 [9] o los $0,7\,mA$ del MCP4922 [7] . En definitiva, el convertidor *DC-DC* proporciona holgadamente la suficiente corriente para la alimentación de todo el circuito.

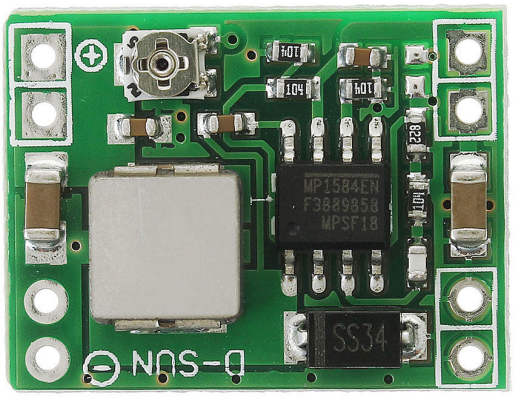

Figura 3.12: Módulo del convertidor *DC-DC* MP1584

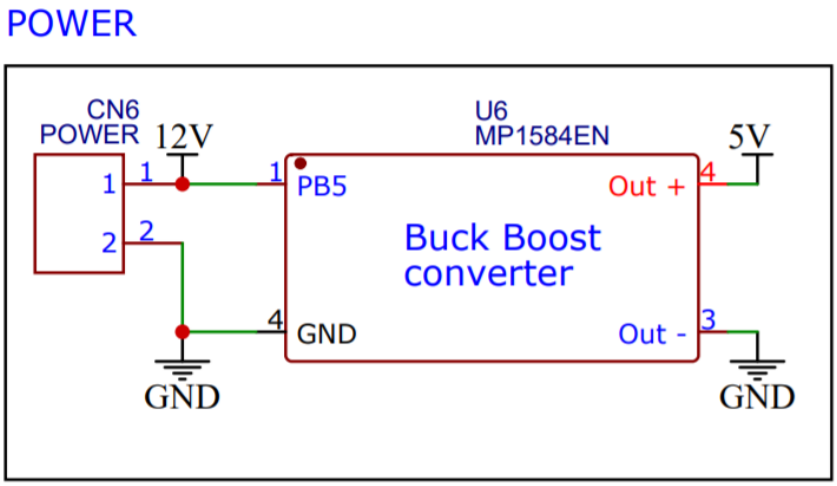

Figura 3.13: Subcircuito de la alimentación

3.1.2 Software embebido

Tras describir el diseño de la electrónica que rodea al microcontrolador, en esta sección se diseña una arquitectura del software embebido del mismo. Este firmware tiene que cumplir los requisitos del sistema, controlando los diferentes dispositivos adicionales del circuito y realizando la conexión con la plataforma *IoT* a través de una conexión *Wi-Fi* y protocolo *MQTT* y con la aplicación móvil mediante *Bluetooth Low Energy*.

3.1.2.1 Arquitectura del Sistema

La arquitectura de un software trata la organización de sus componentes y la relación entre ellos. El software embebido es un caso especial, donde se tiene que interactuar con un entorno hardware de sensores y actuadores y unas restricciones de funcionamiento de tiempo real [11].

En el software se ha realizado una descomposición por funcionalidad y en clases. Este tipo de diseño permite la portabilidad y fácil mantenimiento del software embebido [12]. En este caso, como funcionalidades se ha tenido en cuenta los dispositivos hardware a controlar, **Figura 3.8**.

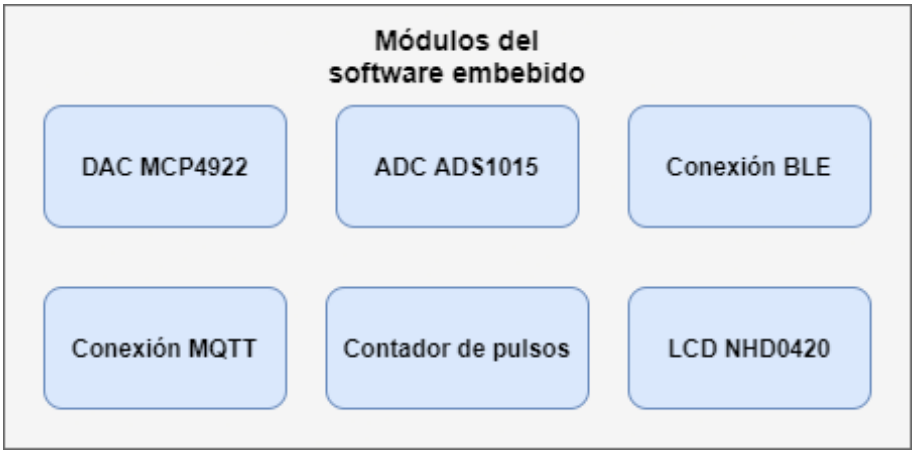

Figura 3.14: Módulos del software embebido

Como *middleware* se utilzará un sistema operativo de tiempo real (*RTOS*), que proporciona múltiples ventajas frente al desarrollo *bare-metal*, como permitir la ejecución de diferentes tareas con prioridades y capacidad de *preemption* sin tener que desarrollar un algoritmo de planificación. De esta manera, los módulos que abstraen el hardware son empleados por un conjunto de tareas que se reparten entre los dos núcleos disponibles. Asimismo, mecanismos como semáforos son necesarios para actuar de manera atómica cuando se utiliza un módulo hardware desde 2 tareas concurrentes.

3.1.2.2 **Diseño detallado**

Una vez descompuesto el software en diferentes funciones, estas se encapsulan en clases para su desarrollo. A continuación, se proporciona la declaración de las clases que definen como se utilizarán en las diferentes tareas.

- **Clase del contador de pulsos**: Se trata de una clase estática que permite inicializar el módulo de contador de pulsos del ESP32 a partir un pin elegido, obtener el número del pulsos detectados hasta el momento y limpiar el contador.

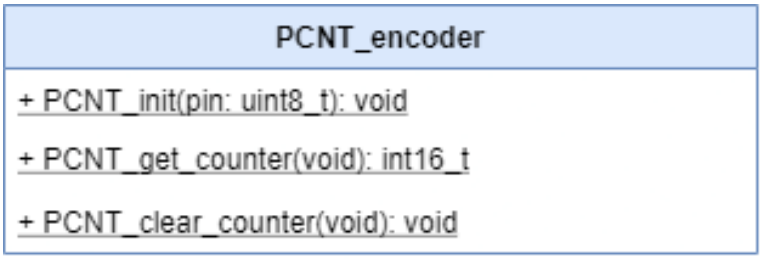

Figura 3.15: Diagrama de Clase PCNT_encoder

- **Clase del *ADC* ADS1015**: A partir del constructor de esta clase, indicamos con que pines se realizará la conexión *I2C*. Podemos inicializar el dispositivo y obtener una lectura instantánea en valor digital y convertida a mV.

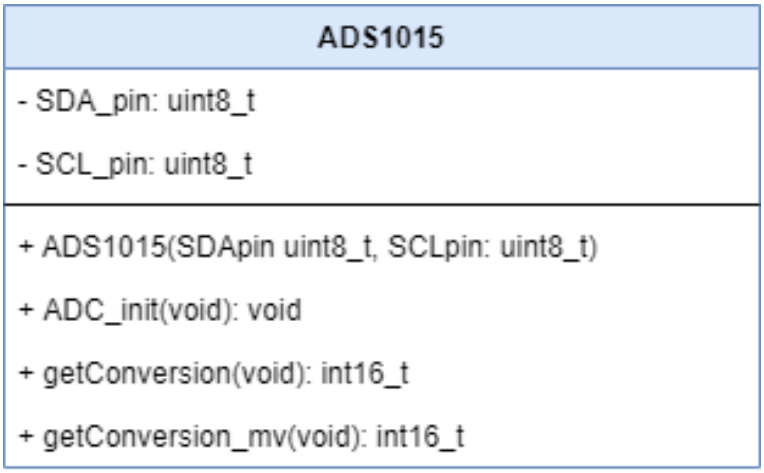

Figura 3.16: Diagrama de Clase ADS1015

- **Clase de la conexión *BLE***: Con el constructor indicamos el nombre que tendrá el microcontrolador al ser buscado por un dispositivo *Bluetooth*. En cuanto al estado del controlador *Bluetooth*, se permite conectar, desconectar y comprobar el valor del mismo. Relacionado con los valores transferidos entre aplicación móvil y microcontrolador, podemos obtener los valores configurados del *LCD* y conexión *Wi-Fi*, además de enviar los valores de medición a la aplicación.

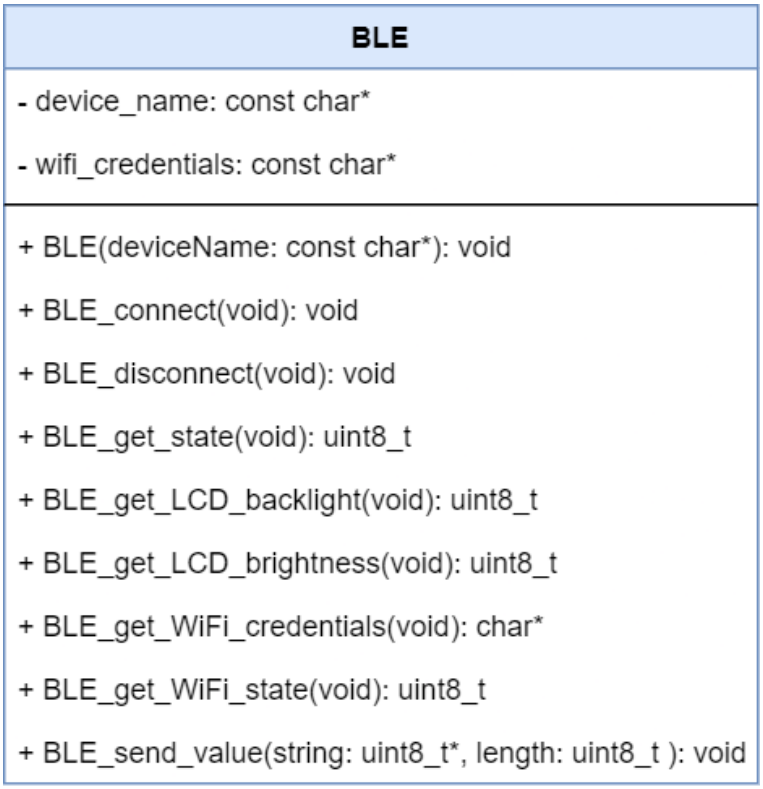

Figura 3.17: Diagrama de Clase *BLE*

- **Clase del *DAC* MCP4922**: A partir del constructor de esta clase, indicamos con que pines se realizará la conexión *SPI*. La definición de la clase da la posibilidad de inicializar el *DAC* y escribir un valor a partir del valor digital o en unidad de mV.

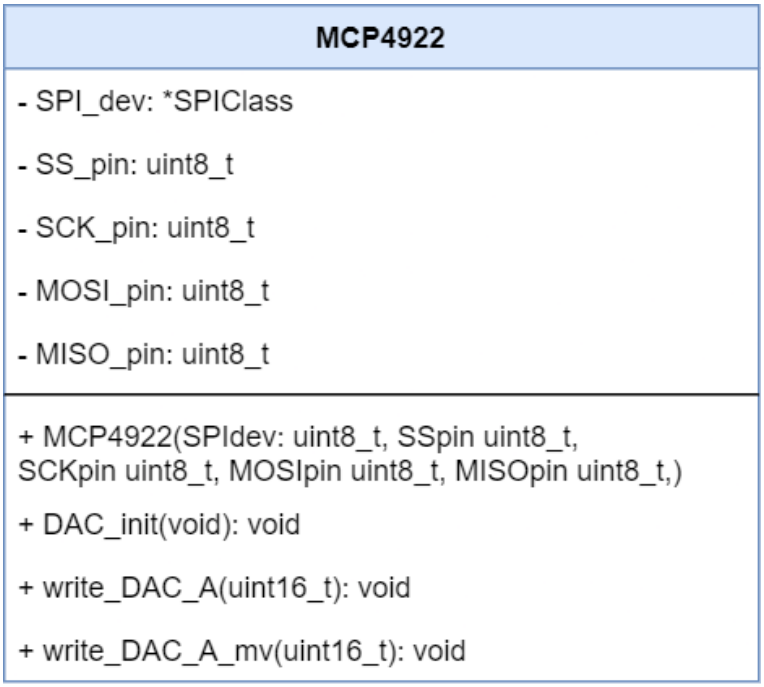

Figura 3.18: Diagrama de Clase MCP4922

- **Clase de la pantalla *LCD* NHD0420**: A partir del constructor de esta clase, indicamos con que pines se realizará la conexión *SPI*. Con los métodos proporcionados, se da la posibilidad de inicializar la pantalla, controlar el cursor de escritura, limpiar la pantalla, ajustar parámetros de visibilidad e imprimir valores en ella.

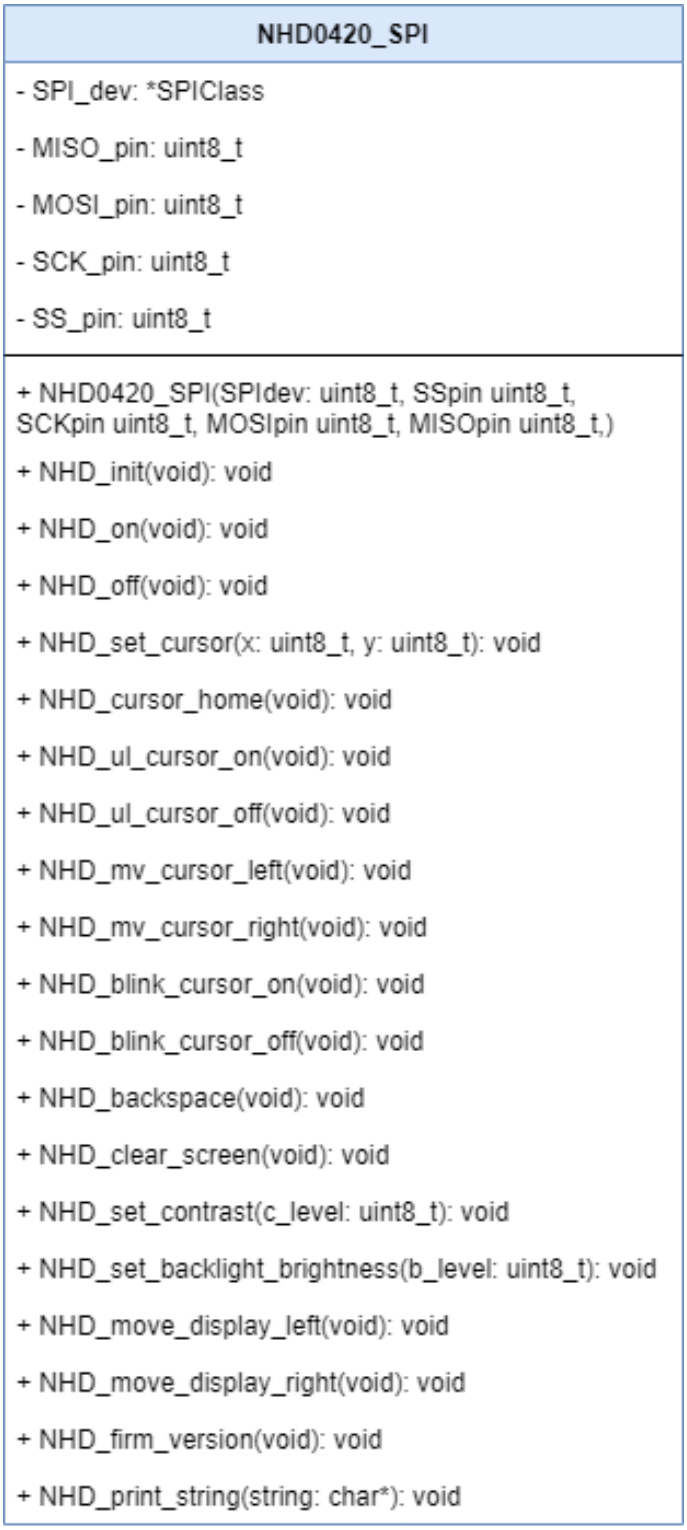

Figura 3.19: Diagrama de Clase NHD0420_SPI

- **Clase de la conexión *Wi-Fi* y *MQTT***: Se proporcionan métodos para conectar y desconectar a la plataforma *IoT*, comprobar el estado de esta conexión y publicar mensajes al mismo.

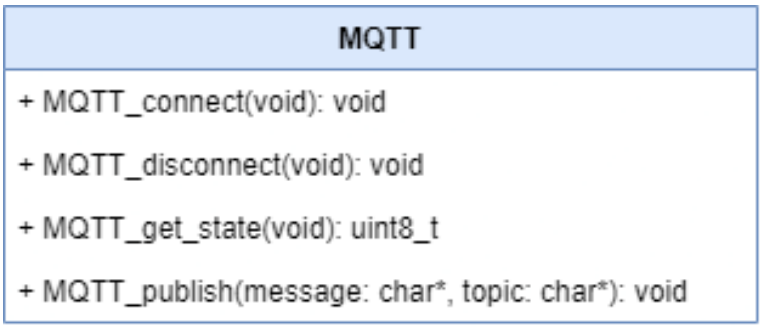

Figura 3.20: Diagrama de Clase MQTT

3.2 Aplicación móvil

La aplicación móvil tiene un diseño sencillo, ya que no realiza comunicaciones con un servidor que contenga un *back end* y bases de datos. La aplicación se limita a recibir y enviar datos *Bluetooth* con el sistema embebido y representarlos en la aplicación de diferentes maneras, además de otras funcionalidades básicas. Por otra parte, para elegir el puesto de prácticas al que conectar, se proporcionará un código QR con su identificación en cada uno, de esta manera, mediante su escaneo, la aplicación intentará la conexión con el puesto deseado.

3.2.1 Navegación

Para el diseño de la aplicación, partiendo de los casos de uso, se esquematizan las diferentes pantallas con *wire-frames*. La aplicación se divide en 2 pantallas, una inicial y otra que se divide en 4 vistas:

- Pantalla inicial: En la primera pantalla de la aplicación se sitúa un botón para realizar el escáner del código QR. Para proceder al resto de vistas se debe escanear un código que indique un puesto de prácticas existente.

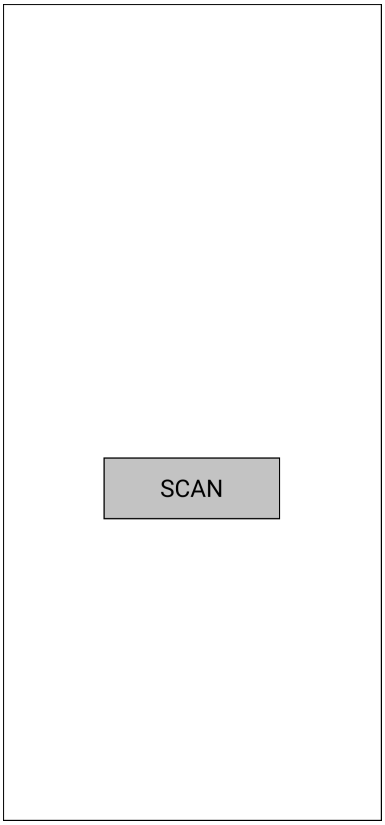

Figura 3.21: Diagrama de la pantalla inicial de escáner QR

- Pantalla principal: Una vez realizada la conexión *Bluetooth*, en la parte superior de la aplicación se mostrará el nombre del puesto de prácticas conectado y, además, un menú inferior con las diferentes opciones:
  - Pantalla de medidas en tiempo real: Los valores de velocidad y par recibos desde el motor se mostrarán en tiempo real.

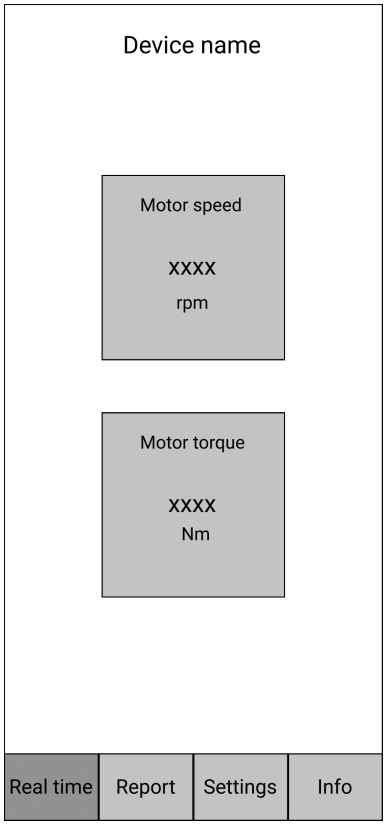

Figura 3.22: Diagrama de la pantalla de tiempo real

– Pantalla de generación de reporte de medidas: Además de mostrar estos valores de tiempo real en un menor tamaño, esta pantalla permite capturar los valores actuales y añadirse en las gráficas para terminar exportándolo a una aplicación externa (ej: de mensajería, almacenamiento, etc).

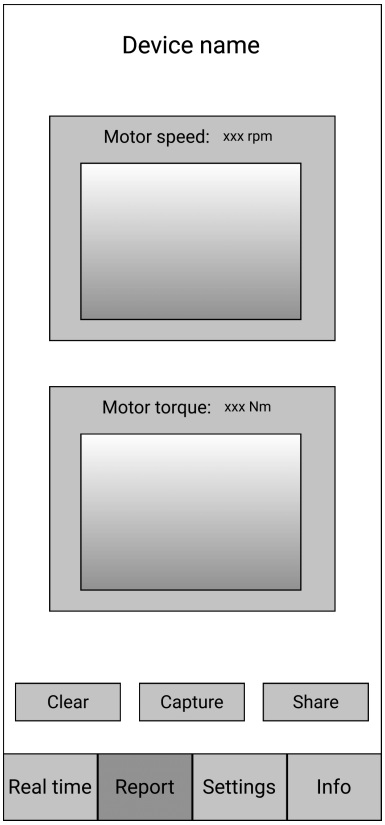

Figura 3.23: Diagrama de la pantalla de reporte

– Pantalla de ajustes: En esta vista de la aplicación se realizan ajustes que se envían al sistema embebido, como los ajustes de la pantalla *LCD* o datos de la red *Wi-Fi*. Además, se permite el cambio de aspecto de la aplicación entre modo claro y oscuro.

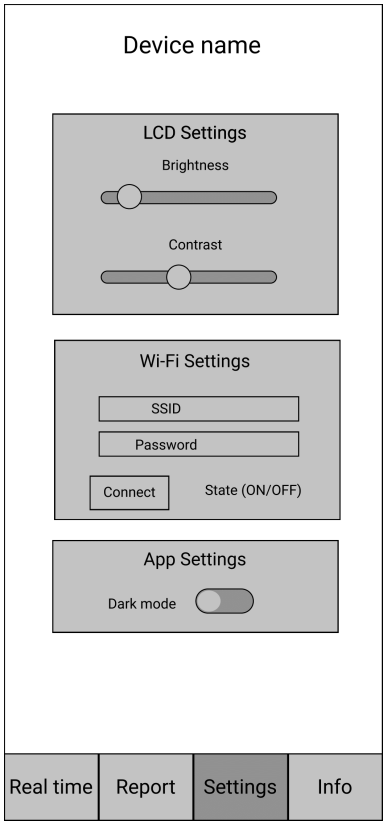

Figura 3.24: Diagrama de la pantalla de ajustes

– Pantalla de información: Finalmente, en esta pantalla se muestra información sobre la aplicación.

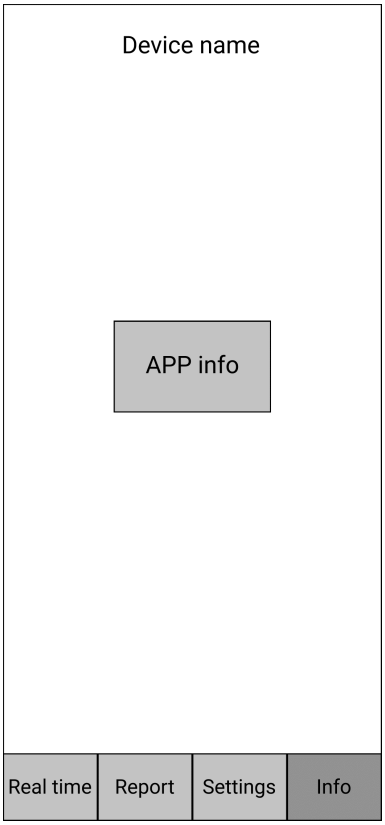

Figura 3.25: Diagrama de la pantalla de información

3.2.2 Manejo del estado

El estado de la aplicación, que contiene datos que se pueden acceder y modificar desde diferentes puntos de la aplicación, se clasifica en 4 grupos:

- Tema visual seleccionado.
- Conexión *Bluetooth*.
- Valores almacenados en la pantalla de captura.
- Pantalla seleccionada en el menú de navegación.

Estos 4 grupos se encapsularán en las siguientes clases:

- **Clase BluetoothModel:** En esta clase se almacena la información del dispositivo conectado, los últimos valores recibidos, el estado de la característica

donde se escribirá los valores enviados al sistema embebido y estos mismos valores de configuración del *LCD* y Wi-Fi. Además, otra variable privada contiene la fecha y hora del último envío.

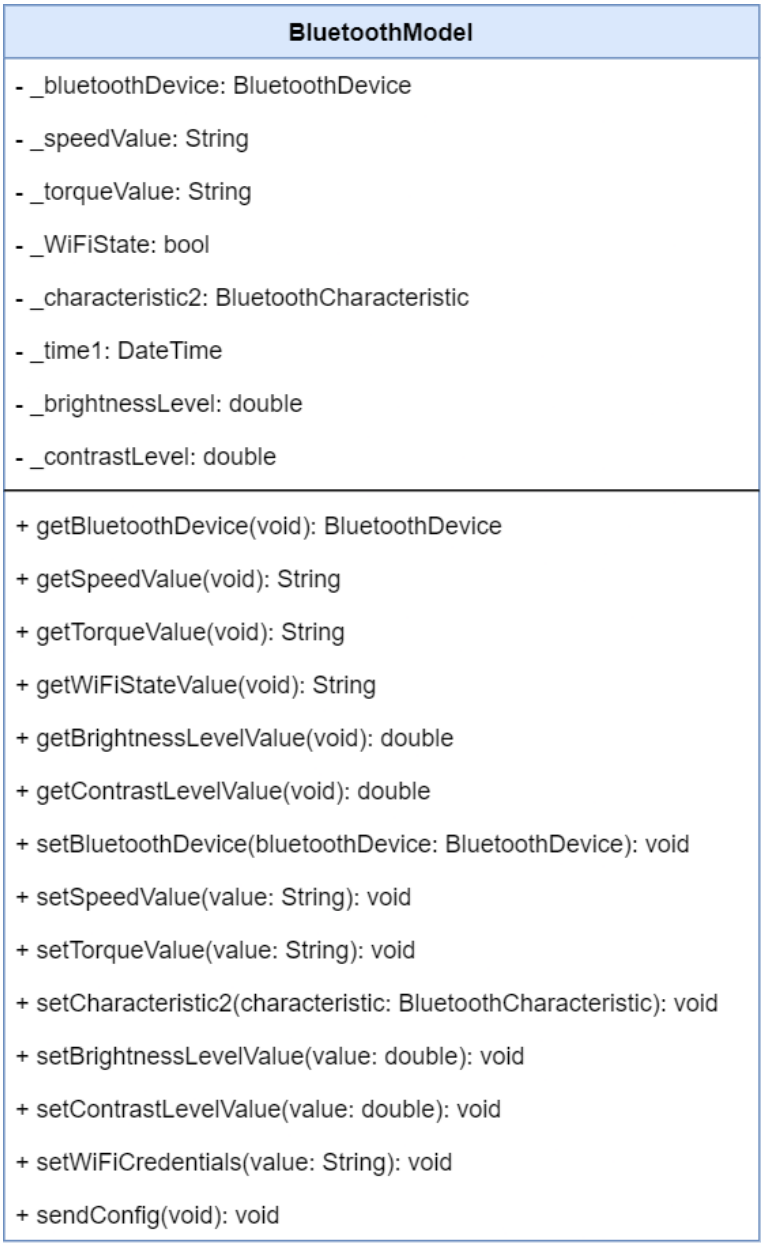

Figura 3.26: Diagrama de la clase BluetoothModel

- **Clase ChartDataModel:** Esta clase contiene 2 listas con los valores de par

y velocidad capturados, permitiendo añadir valores, obtener las listas y vaciarlas.

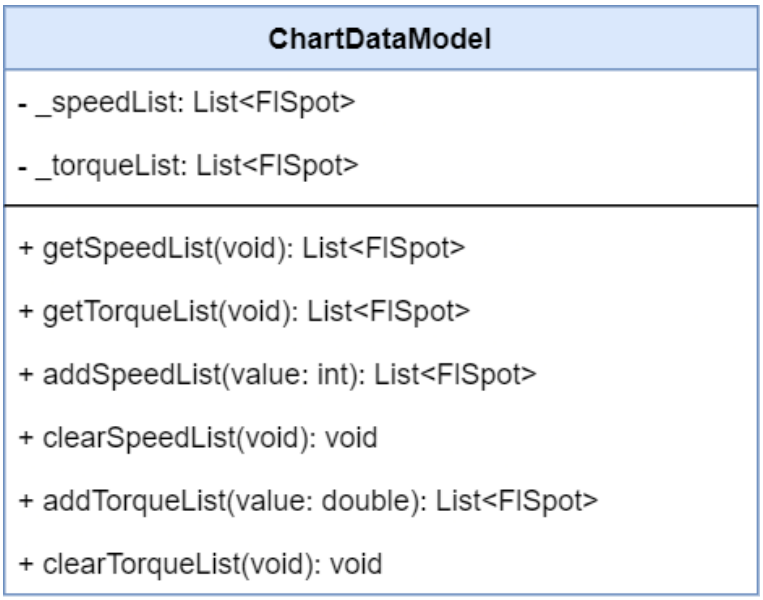

Figura 3.27: Diagrama de la clase ChartDataModel

- **Clase ThemeDataModel:** Los valores de 2 temas, oscuro y claro, se almacenan en esta clase, junto con el tema seleccionado y el estado de un interruptor que lo gestiona en la interfaz gráfica.

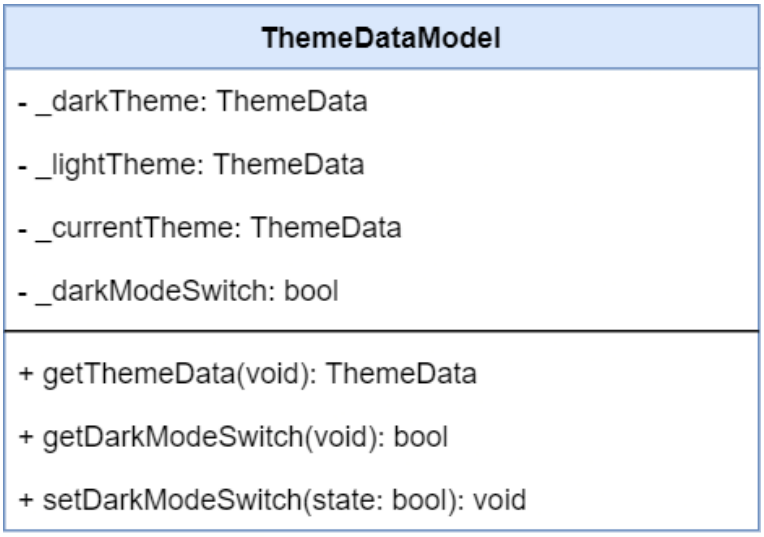

Figura 3.28: Diagrama de la clase ThemeDataModel

- **Clase NavigationBarModel:** Con esta clase se gestiona el estado de la navegación entre las pantallas descritas anteriormente.

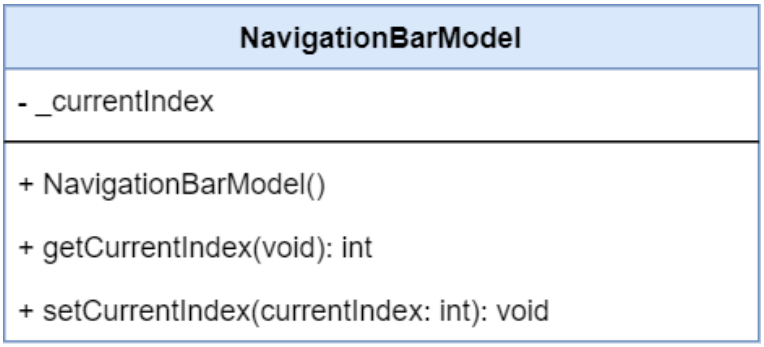

Figura 3.29: Diagrama de la clase NavigationBarModel

Para almacenar este estado se utiliza el patrón de diseño *Provider*. Con este patrón, viendo la aplicación como una jerarquía de árbol, el estado se encuentra en un nivel superior del mismo, permitiendo a los hijos del árbol acceder como *Consumers* a las notificaciones que genera el *Provider* cuando se ha modificado algún valor del estado. Además, estos pueden modificar los valores y que el *Provider* se encargue de notificar al resto de *Consumers*. De esta manera, el estado de la aplica-

ción queda gestionado en un nivel superior de manera centralizada y accesible en cualquier pantalla de la aplicación.

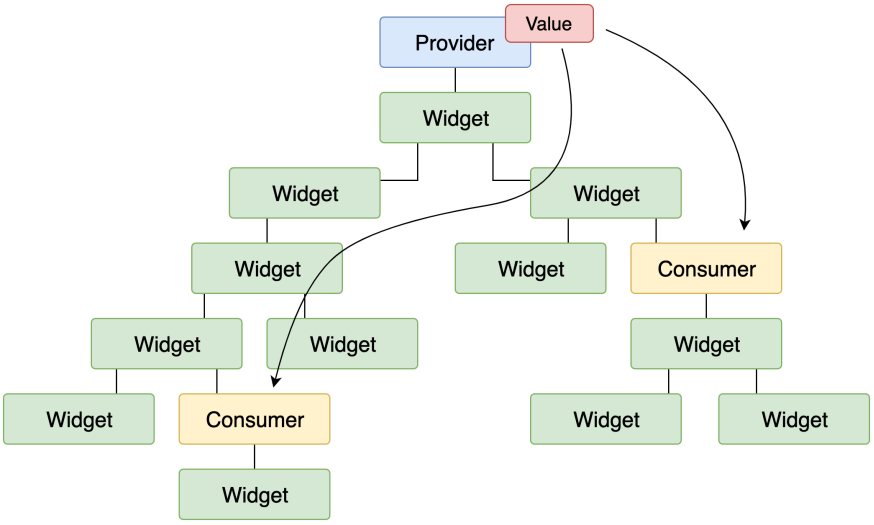

Figura 3.30: Diagrama del patrón *Provider*

3.3 Plataforma *IoT*

Los casos de uso de la plataforma *IoT* descritos en el capítulo anterior se pueden dividir en grupos de casos de usos relacionados. En vez de realizar una arquitectura monolítica que incluya todos los servicios en un único código de la misma tecnología, la arquitectura de la plataforma se organiza en microservicios y cada uno se encargará de realizar una función específica. Esto permite usar tecnologías y lenguajes diferentes en cada servicio, pudiendo elegir el más adecuado o destinar diferentes equipos de desarrollo. Además, cada microservicio se puede escalar individualmente cuando su demanda lo indique, así como realizar su mantenimiento sin afectar al resto de módulos.

3.3.1 Arquitectura del Sistema

Para cubrir los casos de uso de la plataforma, se ha diseñado la arquitectura de la **Figura 3.31**.

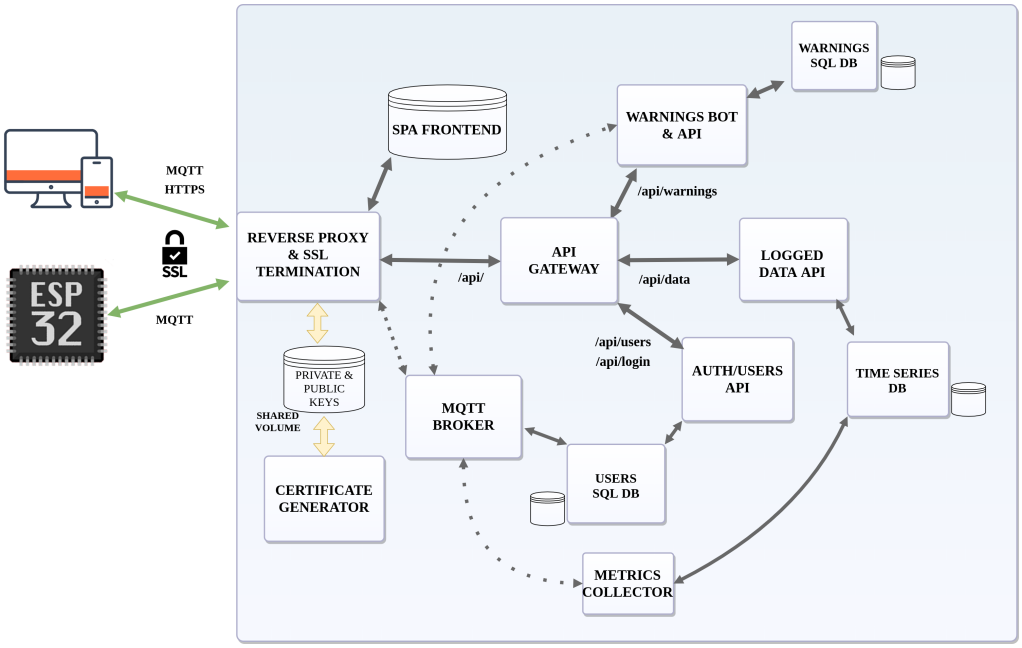

Figura 3.31: Diagrama de la arquitectura de microservicios

Se puede ver en la arquitectura que existen dos tipos de usuarios que interactúan con la red, los nodos *IoT* que toman las medidas y publican en determinadas rutas dependiendo de su tipo de nodo y nombre y los usuarios que monitorizan a los primeros. Los usuarios de monitorización pueden tener permisos de administrador como se indica en la **Figura 3.32**. Además, aunque en este trabajo se desarrolle un nodo *IoT* de un motor de inducción, la plataforma es fácilmente ampliable a otros tipos de nodos que monitoricen otro tipo de máquina con su identificación.

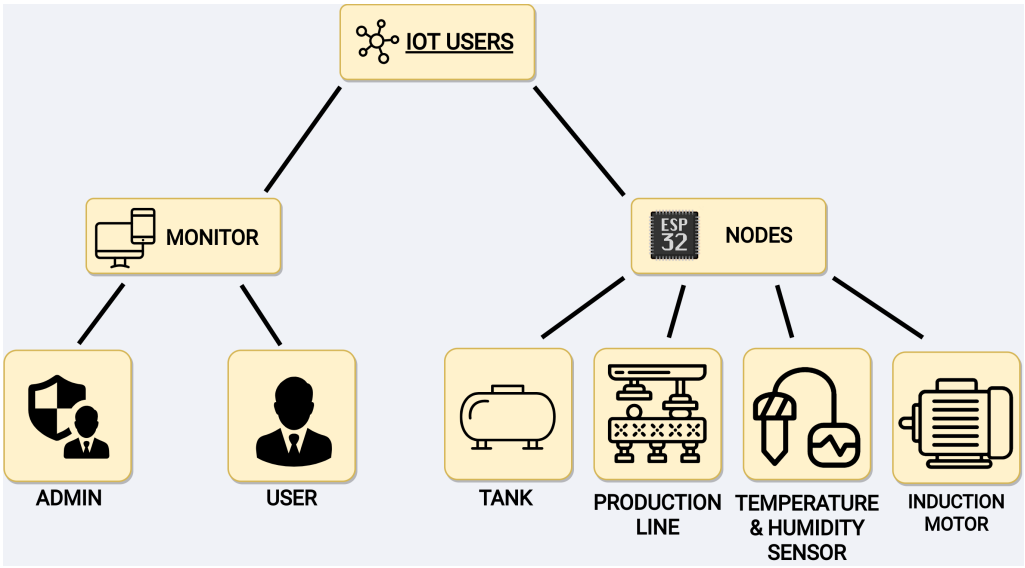

Figura 3.32: Diagrama usuarios de la plataforma IoT

Como descripción general de la plataforma, los nodos *IoT* se limitan a publicar sus medidas con su nombre de usuario en los campos correspondientes a su tipo, como pueden ser *speed* y *torque* en un nodo de tipo monitorización de motor. Para poder realizar esta acción, los nodos tienen unas credenciales para que la plataforma le permita la escritura. Por otra parte, los usuarios de monitorización acceden a una aplicación web con sus credenciales y tienen la posibilidad de interactuar con 3 servicios:

1. Autenticación: Con este servicio, los usuarios monitor pueden autenticarse en la plataforma para acceder al resto de servicios sobre los nodos a los que tiene permisos.

2. Obtención de medidas y alertas: Además de ver medidas de un *nodo IoT* en tiempo real, el usuario monitor puede obtener el histórico de medidas expresadas en gráficas entre 2 fechas indicadas. Además, en una lista se muestra las alertas publicadas por los nodos hasta la fecha actual.

3. Administración de avisos sobre nodos *IoT*: Además de las alertas fijas de los nodos *IoT*, se permite crear unos avisos personalizados sobre cada nodo, proporcionando el campo de de medida, el límite y el usuario de una aplicación de mensajería al que debe avisar.

Nota: Un usuario administrador tiene más opciones en los anteriores servicios en cuanto a la gestión de usuarios de la plataforma, además de tener permisos sobre todos los nodos *IoT* creados.

A continuación se enumeran los componentes que forman la plataforma *IoT*:

- **Reverse Proxy & TLS Termination:** Este componente se encarga de distribuir las conexiones a la plataforma según su protocolo y ruta a otros microservicios. Las comunicaciones *HTTP* en la raíz se dirigen a la aplicación web que se encuentra en el volumen de datos *SPA Frontend*, mientras que las que lleven la ruta */api/* se redirigen al *API Gateway*. Por otra parte, cuando el protocolo se trata de MQTT, las conexiones se dirigen al broker del componente *MQTT Broker*.

  Finalmente, además de redirigir las conexiones a la plataforma, este componente se encarga de centralizar los certificados para una conexión segura mediante *TLS*, recogiéndolos de una unidad de almacenamiento. Como terminación *TLS*, las peticiones MQTT con *TLS* y *HTTPS* se convierten a MQTT y *HTTP*, que luego son gestinados por los otros microservicios. Con esto se evita que cada uno de los componentes gestione por su parte los certificados, con los recursos que esto conlleva.

- **Certificate Generator:** Los certificados que utiliza la terminación *TLS* tienen que ser generados y mantenidos, de esto se encarga este componente.

- **Users SQL DB:** En esta base de datos se almacenan las credenciales de los usuarios e información adicional como su tipo de usuario.

- **MQTT Broker:** Se trata del *broker MQTT* que gestiona a los clientes conectados a la plataforma.

- **API Gateway:** Un *API Gateway* se encarga de recoger las llamadas que recibe a la *API* y distribuirlas por los diferentes microservicios según su ruta.

- **Warning BOT & API:** El microservicio de avisos tiene la función de crear, devolver y eliminar los avisos sobre un nodo *IoT* que tiene un usuario de monitorización. Además, contiene un *bot* que realiza los avisos a los usuarios mediante una aplicación de mensajería cuando estos ocurren.

- **Warnings SQL DB:** En esta base de datos se almacenan los avisos creados por *Warning API*.
- **Logged Data API:** Este microservicio se encarga de devolver las medidas de un nodo *IoT* seleccionado durante un periodo determinado de la base de datos de series temporales.
- **Auth/Users API:** La aplicación web utiliza este microservicio para autenticar a los usuarios, obtener sus datos y crear nuevos en la base de datos *Users SQL DB*.
- **Metrics Collector:** Este último componente tiene la capacidad de recoger las medidas que envían los nodos *IoT* al broker MQTT y almacenarlas en la base de datos de series temporales para luego poder ser obtenidas por el microservicio.

3.3.2 Diseño detallado

Cada uno de los microservicios (*WARNING, LOGGED DATA, AUTH/USERS*) proporciona una *API REST*, donde las llamadas a la *API* son independientes y no existe un estado o sesión compartido entre ellas. Debido a esto, el modelo de autenticación de usuarios de la plataforma es mediante el uso de *tokens*. El flujo que se realiza en esta autenticación es el siguiente:

1. El usuario utiliza la *API* del microservicio de autenticación con sus credenciales, el cual es libre para cualquier usuario sin autenticar.
2. En microservicio comprueba las credenciales, en caso de coincidir con una pareja de usuario-contraseña existente, encripta un *token* que le es enviado al usuario junto con información del mismo (tipo de cuenta, permisos, etc). Este *token* lo almacena el usuario en su almacenamiento local.
3. Al hacer uso de uno de los microservicios con rutas protegidas a usuarios autenticados, las peticiones llevan adjuntas el *token* en su cabecera y, cuando llega al *API Gateway*, se comprueba su autenticidad con la clave privada.

4. En caso de ser un *token* correctamente encriptado por la plataforma y sin modificar, la petición se deja pasar al microservicio de la ruta correspondiente.

Para que la aplicación web pueda acceder a las *API* de los microservicios, en las siguientes tablas se detallan los métodos que se debe implementar en cada uno.

- **Auth/Users API:** El único método que se proporciona en la ruta *login* espera una pareja de usuario-contraseña, el cual comprueba su validez para devolver el *token* encriptado o un error de autorización cuando no coincidan o del propio servidor para errores internos en el acceso a la base de datos.

Method	URL	Body type	Request Body	Response Status Code	Response Body
POST	api/login	json	{username, password}	200: OK 401: Unauthorized 500: Internal server error	{token} {message: 'Invalid login'} {message: 'Database error'}

Figura 3.33: Métodos de la *API* de *login* del microservicio *AUTH/USERS*

Por otra parte, en cuanto a los datos de usuarios, en este microservicio se permiten 4 métodos:

1. Obtener datos del usuario actual: Con esta petición, a partir del id de usuario de nuestro *token* obtenemos el nombre de usuario, los nodos a los que tenemos permisos y el estado de administrador.
2. Crear nuevo usuario: A partir de los datos de un nombre, contraseña, tipo de usuario y permisos, creamos un nuevo usuario si no existe uno con el mismo nombre. Nota: Esta acción sólo la puede realizar un administrador.
3. Borrar un usuario: Permite pedir la eliminación de un usuario a partir de su nombre. Nota: Esta acción sólo la puede realizar un administrador.
4. Obtener tipos de usuarios: Con este último método, pedimos una lista con los tipos de usuarios y campos a los que publican datos en caso de tratarse de un nodo.

Method	URL	Body type	Request Body	Response Status Code	Response Body
GET	api/users/userData	json		200: OK 500: Internal server error	{username, permissions: [{username, user_class},], admin} {message: 'Database error'}
POST	api/users	json	{username, password, user_class, permissions}	201: Created 403: Forbidden 500: Internal server error	{message: 'User ${username} created successfully'} {message: 'User ${username} already exist'} {message: 'Database error'}
DELETE	api/users/:username	json		200: OK 404: Not found 500: Internal server error	{message: 'User ${username} deleted successfully'} {message: 'User ${username} does not exist'} {message: 'Database error'}
GET	api/users/userClasses	json		200: OK	[{user_class, topics: [ ]}]

Figura 3.34: Métodos de la *API* de *usuarios* del microservicio *AUTH/USERS*

- **Warnings API:** En este microservicio tenemos 3 métodos:

  1. Obtener avisos de nuestro usuario: A partir del *id* de usuario almacenado en el *token*, se devuelven los datos de los avisos creados sobre los nodos *IoT* con permisos.
  2. Crear un nuevo aviso: Aportando el nombre del nodo *IoT*, campo de medición, valor límite, signo mayor/menor y nombre de usuario de la aplicación de mensajería para avisar, se crea un nuevo aviso.
  3. Borrar un aviso existente: Mediante el *id* del aviso en la *URL*, se pide eliminar un aviso anteriormente creado.

Method	URL	Body type	Request Body	Response Status Code	Response Body
GET	api/warnings	json		200: OK 500: Internal server error	[{username, topic, data_value, data_sign, tel_username, id},] {message: 'Database error'}
POST	api/warnings	json	{device_username, topic, value, sign, tel_username}	201: OK 403: Forbidden 500: Internal server error	{message: 'Warning created successfully'} {message: 'User does not have permissions'} {message: 'Database error'}
DELETE	api/warnings:id	json		200: OK 403: Forbidden 500: Internal server error	{message: 'Warning deleted'} {message: 'User does not have permissions'} {message: 'Database error'}

Figura 3.35: Métodos de la *API* del microservicio *WARNINGS*

- **Logged Data API:** Finalmente, este microservicio proporciona 2 métodos:

  1. Obtener medidas de un nodo *IoT* entre dos tiempos determinados: El microservicio devuelve una lista con las parejas de medida-tiempo de un nodo *IoT* entre los 2 tiempos indicados.
  2. Obtener alertas de los nodos *IoT*: Se recibe una lista con las alertas de los nodos *IoT* con permisos, donde se incluye la fecha, el nombre del nodo, tipo de nodo y valor de la alerta.

Method	URL	Body type	URL parameters	Response Status Code	Response Body
GET	api/data	json	device date1 date2	200: OK 403: Forbidden 500: Internal server error	[[{time, value},],] {message: 'Device not authorized'} {message: 'Database error'}
GET	api/data/alerts	json		200: OK 403: Forbidden 500: Internal server error	[{time, username, user_class, value},] {message: 'User has not authorized devices'} {message: 'Database error'}

Figura 3.36: Métodos de la *API* del microservicio *LOGGED DATA*

CAPÍTULO

4

Tecnologías y herramientas

En este capítulo se describen las diferentes tecnologías y herramientas empleadas para el desarrollo del software del diseño de los 3 sistemas.

4.1 Software embebido

4.1.1 Tecnologías

En el desarrollo del firmware de un microcontrolador ESP32 se tiene la posibilidad de utizar el *Framework* Arduino o el propio ESP-IDF de Espressif. Debido a que la adaptación a Arduino se realiza como envolvente al ESP-IDF y que no impide seguir haciendo uso de las librerías de este, se ha decidido emplear el *Framework* de Arduino proporcionado por Espressif. De esta manera, permite el desarrollo con un alto nivel de abstracción al mismo tiempo que llegar a un nivel inferior en dispositivos hardware que no cubran la plataforma Arduino o en partes donde se necesite mayor control del que proporciona esta plataforma. Esta adaptación de ESP-IDF a Arduino utiliza el lenguaje C++ y tiene como *middleware* un sistema operativo de tiempo real, *FreeRTOS*.

Por otra parte, para la comunicación de este sistema con los otros 2 desarrollados, aplicación móvil y Plataforma *IoT*, se ha empleado la tecnología *Bluetooth Low Energy* y el protocolo de comunicación *MQTT*.

4.1.1.1 FreeRTOS

FreeRTOS es un sistema operativo de tiempo real. Este tipo de sistema operativo destaca por tener un planificador que permite la multitarea de una manera de determinista, lo cual lo hace ideal para sistemas embebidos con requisitos de tiempo real [13]. Un caso de requisito de tiempo real en este sistema es la necesidad de tomar la medida de pulsos del tacómetro cada cierto intervalo fijo de tiempo.

Cabe destacar que las tareas que se introducen el planificador tienen unas prioridades y, además, en el caso del ESP32, estas pueden ser distribuidas por los dos núcleos disponibles en el microcontrolador y permitir que en un núcleo una tarea crítica tenga todo el tiempo disponible, mientras que el resto de tareas se reparten el tiempo del segundo núcleo.

FreeRTOS está diseñado expresamente para ser usados en microcontroladores de bajos recursos de memoria y procesamiento. Sin embargo, los típicos mecanismos de los sistemas operativos de tiempo real como la comunicación entre tareas o los diferentes métodos de sincronización siguen estando presentes.

4.1.1.2 BLE

Bluetooth Low Energy es una versión más optimizada del protocolo de comunicación Bluetooth diseñado para dispositivos de bajo consumo, el cual se ha convertido en el estándar en la comunicación inalámbrica entre los teléfonos inteligentes y sus accesorios[14].

4.1.1.2.1 Topología de RED

Los dispositivos *BLE* se pueden comunicar mediante *broadcasting* o *connections*.

- *Broadcasting*: El dispositivo envía datos a cualquier receptor que se encuentre en el rango de escucha. Estos datos se envían en un solo sentido.

- *Connections*: Mediante esta conexión, dos dispositivos *BLE* pueden intercambiar paquetes de datos en los 2 sentidos. Cabe de destacar que la comunicación es sólo entre dos dispositivos.

Como se exige en los requisitos, la conexión debe ser entre dos dispositivos (aplicación móvil y sistema de medición) y de manera bidireccional, lo que hace necesaria la la comunicación como *connection*.

Con esta topología, los dispositivos pueden tener dos roles:

- Periférico (esclavo): Este dispositivo envía paquetes periódicamente para avisar de su disponibilidad y aceptar una conexión. Una vez aceptada, el periférico deja de emitir paquetes de aviso.
- Central (maestro): El dispositivo escanea las diferentes frecuencias buscando los paquetes enviados por un periférico. Una vez conectado, el maestro lleva el mando del intercambio de datos.

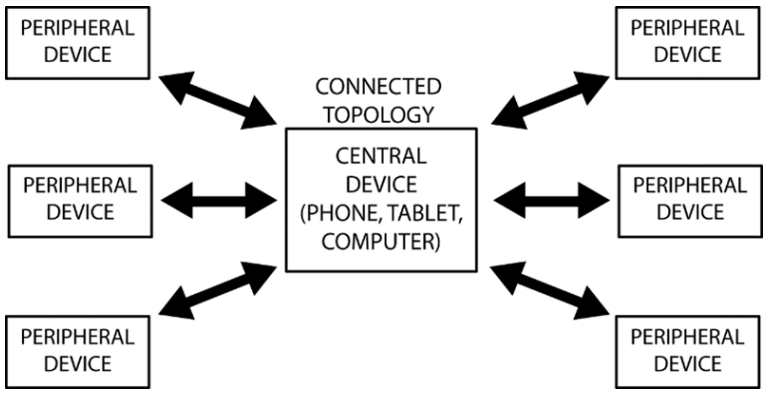

Figura 4.1: Diagrama de la topología *connections*

Una ventaja de la topología *connections* es la organización de los datos utilizando capas adicionales al protocolo, *Generic Attribute Profile* (*GATT*). Estos datos se organizan en servicios y características.

4.1.1.2.2 *Attribute Protocol* (*ATT*)

En *BLE*, es un protocolo simple cliente/servidor basado en atributos presentes en un dispositivo. Los atributos son la unidad mínima de datos definido por *ATT* y

GATT. Cada dispositivo es un cliente, servidor o los dos, independientemente de ser maestro o esclavo. El servidor organiza los datos en forma de atributos, incluyendo un identificador (*UUID*), permisos y el valor, a los que accede el cliente.

4.1.1.2.3 *GATT*

Añade una jerarquía y capa de abstracción a la organización de datos del protocolo de atributos (*ATT*). Define como se organizan y se intercambias los datos entre dispositivos. A partir de este perfil genérico, se pueden diseñar otros perfiles específicos (*GATT-based profiles*).

Los atributos son encapsulados en servicios, los cuales contienen una o más características. Cada característica se trata de un dato unido a otros meta-datos que describen propiedades como nombre o unidad de medida.

- Servicios: Reúne atributos relacionados en una sección.
- Características: Las características son contenedores de datos. Incluye al menos dos atributos:
  - Declaración de característica: Meta-datos del valor almacenado.
  - Valor de la característica.

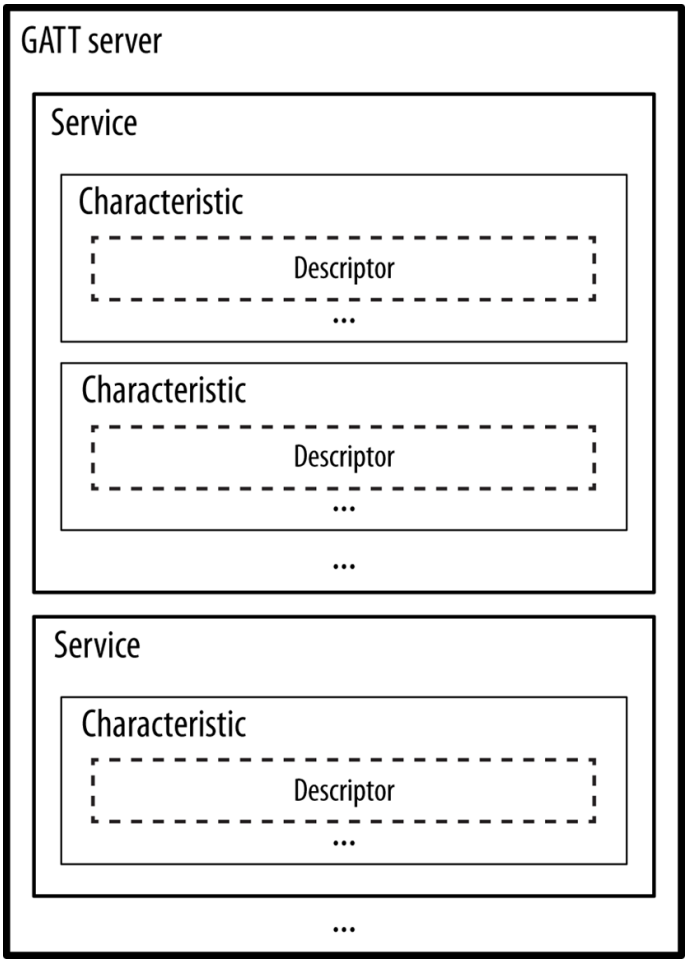

Figura 4.2: Organización de datos en *GATT*

4.1.1.3 **MQTT**

MQTT es un protocolo de comunicación de tipo *M2M* (*machine-to-machine*) que se basa en la pila *TCP/IP*. Su diseño permite un correcto funcionamiento en redes inestables, a la vez que consumir un bajo ancho de banda. Esto hace que su uso sea muy común en dispositivos *IoT*, donde muchos de estos dispositivos se basan en microcontroladores sencillos. Otros protocolos de la capa aplicación de la pila *TCP/IP* como *HTTP* necesitan mayor gasto computacional y consumo de energía, además de estar diseñado para la transferencia de documentos, en vez de datos como *MQTT* [15].

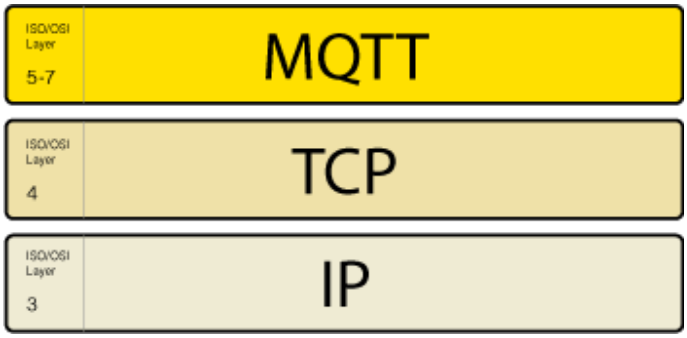

Figura 4.3: Pila del protocolo *MQTT* sobre *TCP/IP*

En una comunicación *MQTT* existen dos tipos de dispositivos, el servidor o ***Broker*** y el cliente. La comunicación se basa en el patrón de publicación-suscripción, en la que los clientes no tienen conciencia de la existencia de otros clientes, por lo que sus mensajes los publican a un *topic* determinado o se suscriben al mismo para recibir los mensajes publicados por otro cliente. Para que este comportamiento sea posible, el *broker* se encarga de recibir los mensajes enviados por los clientes y distribuirlos a los otros que estén interesados.

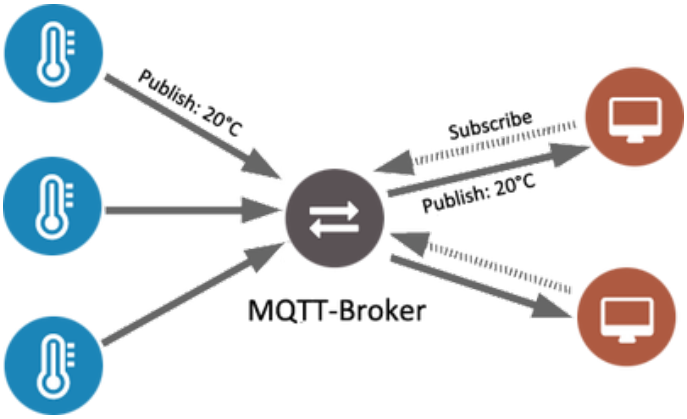

Figura 4.4: Diagrama de dispositivos en una comunicación MQTT

Los *topics* son una manera de clasificar los mensajes como si de una jerarquía de directorios se tratase. Los clientes pueden suscribirse específicamente a un *topic* o a un nivel superior que englobe varios de estos.

Una ventaja del protocolo *MQTT* es la inclusión de 3 tipos de calidad de servicio (*QoS*) y conseguir un intercambio de datos más robusto. Los niveles corres-

ponden a:

- **QoS 0** (Como mucho una vez): Se mantiene la misma garantía que el protocolo *TCP* subyacente. El mensaje no es comprobado por el receptor, por lo que es el que tiene mayor velocidad de recepción y menos exigencia de computación.
- **QoS 1** (Al menos una vez): Con este nivel de *QoS* se garantiza que el mensaje ha llegado al destinatario, pero no si ha sido de manera duplicada.
- **QoS 2** (Exactamente una vez): El mayor nivel de calidad de servicio consigue garantizar que el mensaje ha llegado exactamente una vez al destinatario.

La elección de uno de los niveles de calidad de servicio viene dado por la exigencia de rapidez de recepción, gasto de banda ancha y nivel de seguridad de recepción.

4.1.2 Herramientas

Para llevar a cabo el desarrollo en un ESP32, Espressif no dispone de ningún entorno de desarrollo (*IDE*) oficial, sino que proporciona unas herramientas de *cross-compilado* y una serie de scripts para compilar aplicaciones C/C++ y *linkear* con sus librerías.

Ante el aumento de fabricantes de microcontroladores y entornos de desarrollo propios surgió la plataforma de código libre **PlatformIO**. Esta herramienta permite desarrollar para múltiples arquitecturas de los fabricantes más destacados con diferentes *frameworks* disponibles. Esta herramienta se proporciona como una extensión para uno de los editores de textos más utilizados en el momento, **VS Code** de Microsoft.

Figura 4.5: VS Code y PlatformIO

VS Code es un editor de código de código abierto multiplataforma que incluye soporte para *debugging*, correción de sintaxis y compleción de código, entre otra muchas funcionalidades en un entorno moderno y sencillo. Destaca por su gran capacidad de ampliación mediante *plugins* y su gran comunidad. Por medio de estos, en un mismo editor se puede desarrollar cualquier tipo de aplicación como software embebido, aplicaciones de escritorio, web o móvil.

En el caso del ESP32, tras instalar el editor de texto VS Code y la extensión PlatformIO, el comienzo de un proyecto se convierte en la sencilla tarea de elegir una placa de desarrollo y un *framework* (Arduino o ESP-IDF).

Con la creación del proyecto se nos proporciona un fichero *main* y una carpeta *lib* donde incluir nuestras librerías. Además, en el fichero de configuración *platformio.ini* se permite cambiar parámetros como la velocidad del puerto de serie que se muestra por el terminal.

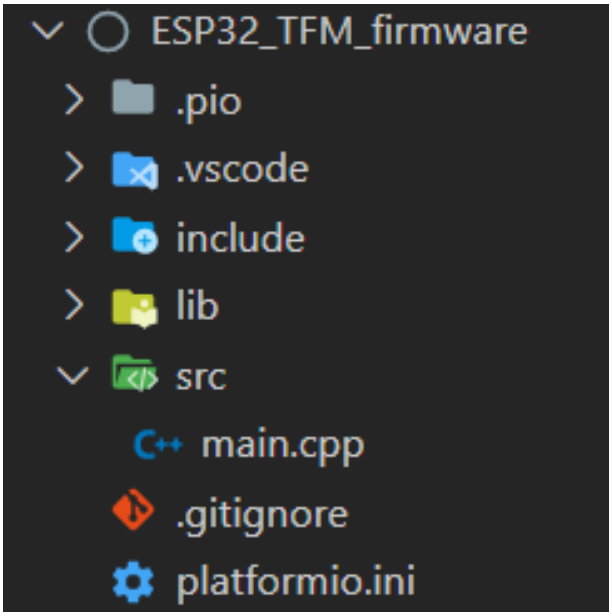

Figura 4.6: Directorios de un proyecto ESP32 en PlatformIO

4.2 Aplicación móvil

4.2.1 Tecnologías

Para desarrollar la aplicación móvil se utiliza el kit de desarrollo de software libre (*SDK*) **Flutter** de Google, el cual emplea el lenguaje de programación **Dart**.

4.2.1.1 Dart

Dart es un lenguaje de programación de propósito general de código abierto desarrollado por Google. Está orientado al desarrollo de aplicaciones cliente con una sintaxis similar a otros lenguajes orientados a objetos como C# o Java. Su transición es muy sencilla con conocimientos de programación previos, ya que evita el código *boilerplate* propio del lenguaje y permite tanto un tipado estático como dinámico.

Una de las mayores ventajas de Dart es la inclusión de dos tipos de compilaciones, *Ahead Of Time* (*AOT*) y *Just In Time* (*JIT*). Con la compilación *AOT*, el código de Dart se convierte a código máquina de la arquitectura correspondiente (ARM64/x86-64), permitiendo una ejecución predecible y el mayor rendimiento para producción. Por otra parte, con la compilación *JIT*, el código se compila en

el momento justo antes de la ejecución del mismo, lo que permite un proceso de desarrollo muy rápido en cuanto a la transición entre la modificación del código y la ejecución en el dispositivo objetivo a costa del rendimiento. En este último modo de compilación, la máquina virtual de Dart se encarga de realizar el proceso y proporcionar el entorno de ejecución.

4.2.1.2 **Flutter**

Flutter es un kit de desarrollo de aplicaciones móviles que permite obtener ejecutables para iOS y Android con un único código base. A diferencia de otros kits de desarrollo multiplataforma, las aplicaciones se compilan para la arquitectura objetivo, obteniendo un gran rendimiento sin puentes entre otro lenguaje interpretado como JavaScript.

Una aplicación Flutter se organiza en *widgets* que forman toda la interfaz de usuario. Se puede decir que todo en Flutter es un *widget*, tanto elementos visibles (un texto o una imagen) como de organización (Contenedor, *padding*, filas, columnas). Todos estos *widgets* se encuentran en una estructura de árbol como se observa en la siguiente imagen.

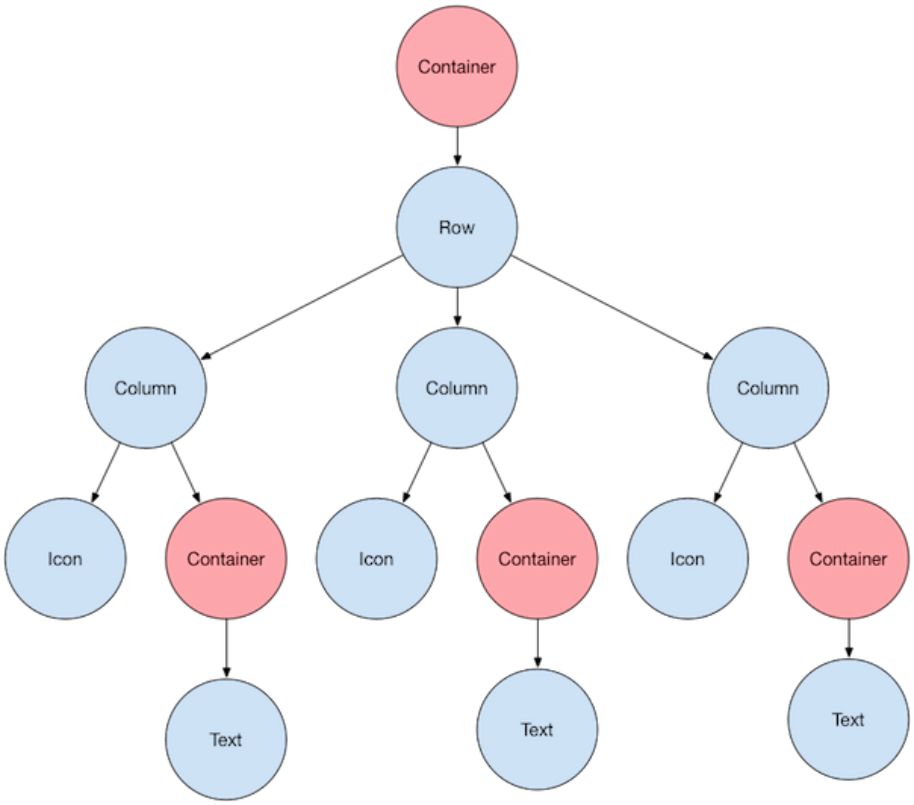

Figura 4.7: Diagrama de árbol de *widgets*

Hay que destacar que Flutter no utiliza los *widgets* originales de las plataformas Android y iOS, sino que proporciona otros propios con el mismo estilo y comportamiento, a partir de los cuales, el desarrollador puede derivar a los suyos propios. Esta manera de mostrar la interfaz, junto con la utilización de Dart y sus dos modos de compilación cambian totalmente el panorama de desarrollo multiplataforma de aplicaciones móviles.

Al compilar una aplicación Flutter como *AOT* se utiliza la librería de gráficos 2D **Skia**. Este motor gráfico es el que se encarga de dibujar los *widgets* en la pantalla del dispositivo como si de un videojuego se tratase, consiguiendo interfaces muy fluidas compatible con tasas de refresco de 90 Hz o 120 Hz que se incluyen en nuevos dispositivos móviles.

Flutter también aprovecha la capacidad de compilación *JIT* de Dart utilizando lo que denomina *hot reload*. Con esta funcionalidad, un cambio en el código fuente se ve reflejado de manera instantánea en un emulador del dispositivo objetivo o

en uno real en modo de desarrollo, sin necesidad de reiniciar y compilar toda la aplicación en cada cambio.

La arquitectura de Flutter se divide en varias capas, donde ninguna tiene privilegios para acceder a otra de nivel inferior. [16]

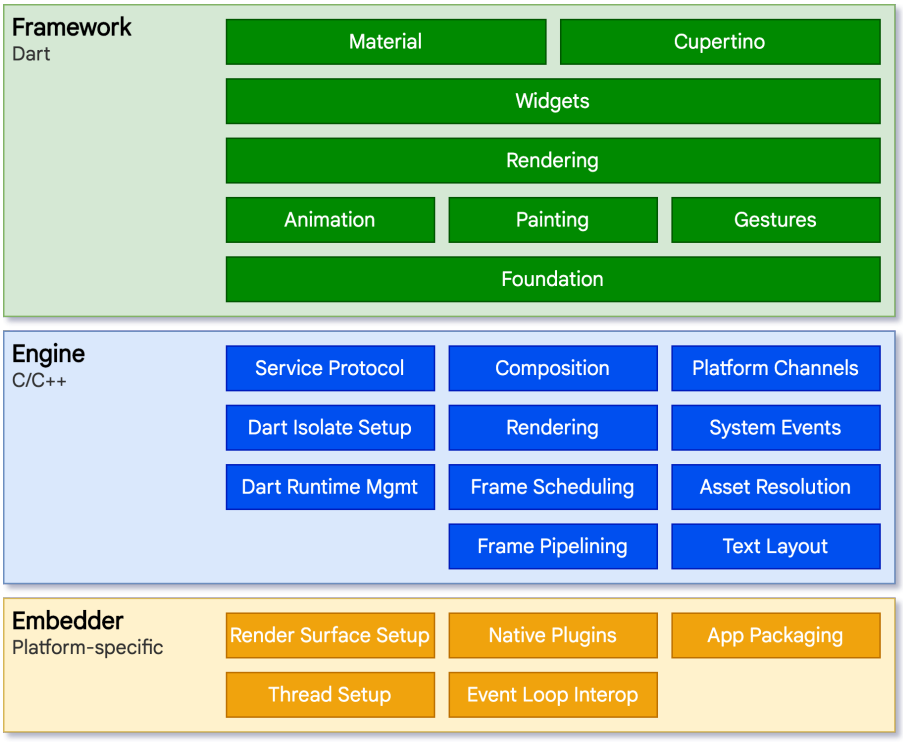

Figura 4.8: Diagrama de la arquitectura de Flutter

En la capa más cercana al hardware se encuentra el *Embedder* de la plataforma, que proporciona un *entry-point* para la ejecución y permite las llamadas de sistema.

Por otra parte, el motor de Flutter incluye la implementación de la *API* de más bajo nivel con los gráficos de Skia, control de entradas y salidas, comunicaciones y el entorno de ejecución de Dart. Esta capa se conecta a la superior con una interfaz en Dart que envuelve el código C++ del motor.

Finalmente, el *framework* con el que interactúa el desarrollador contiene las librerías escritas en Dart, que abstraen el funcionamiento del motor de Flutter para facilitar el desarrollo de las interfaces y proporciona todos los *widgets* que reproduce los originales de las plataformas iOS y Android junto con los de organización

y reconocimiento de gestos.

4.2.2 Herramientas

En el desarrollo de una aplicación con el kit de desarrollo Flutter tenemos la eleccion de elegir el entorno de desarrollo de Android Studio o VS Code. Se elegido para el desarrollo el editor de código VS Code, para el cual Google proporciona una extensión oficial, por ser más liviano y permitir instalar las múltiples extensiones que desarrolla la comunidad de VS Code para facilitar este desarrollo. En la siguiente captura se muestra un proyecto de Flutter en este editor.

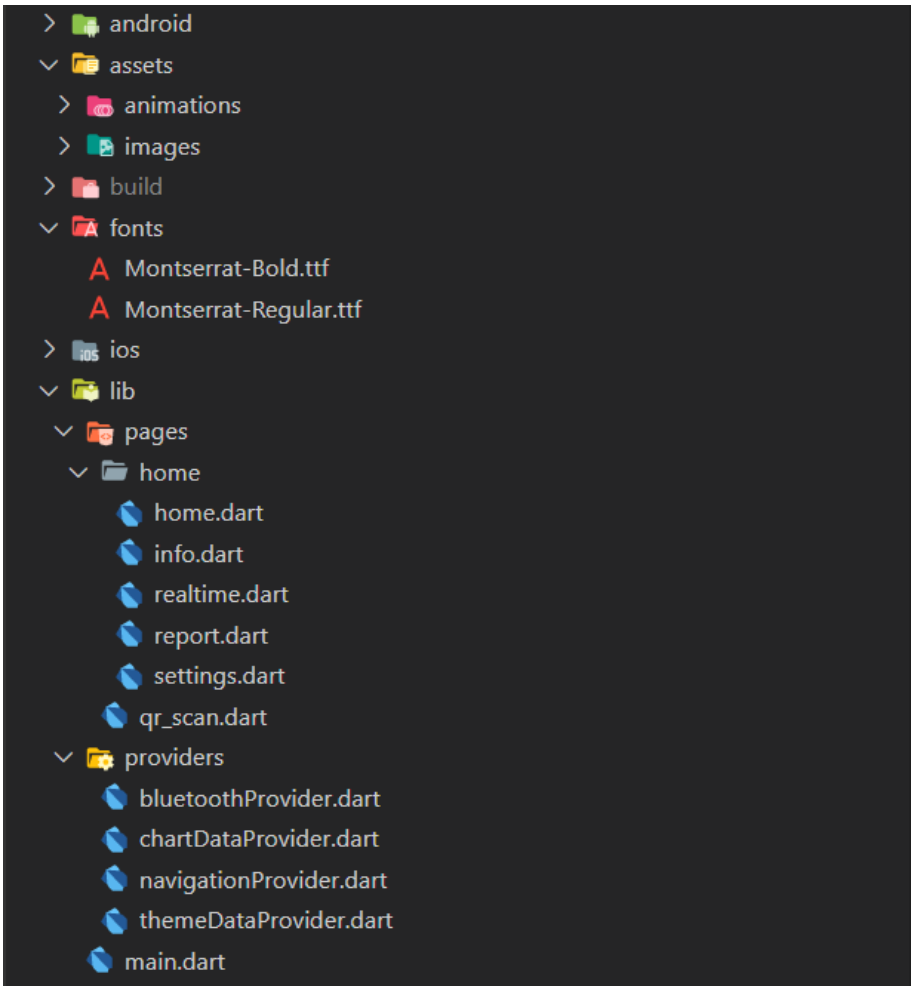

Figura 4.9: Directorios de una aplicación Flutter en VS Code

En la raíz del proyecto nos encontramos 5 carpetas:

- **android:** Contiene la base de la aplicación Android nativa a la que se unirá la aplicación desarrollada en flutter.
- **ios:** Contiene la base de la aplicación iOS nativa a la que se unirá la aplicación desarrollada en flutter.
- **assets:** En esta carpeta se introducen los recursos gráficos como imágenes o animaciones que se utilizarán en el código.
- **fonts:** En esta carpeta se incluyen las fuentes externas que se emplearán al renderizar la aplicación.
- **lib:** Se trata de la carpeta principal de desarrollo. Contiene el fichero principal *main.dart* y los que se deseen incluir en un diseño modular.

4.3 Plataforma IoT

Una vez se ha diseñado la arquitectura del software, el siguiente paso es elegir la tecnología adecuada para cada componente de la plataforma.

4.3.1 Tecnologías

Tratándose de un diseño con múltiples componentes y microservicios, la tecnología **Docker** es idónea para contener cada una de las aplicaciones que forman el conjunto del software. Docker permite empaquetar una aplicación software junto con sus librerías y dependencias en un contenedor. Estos contenedores se pueden desarrollar y ejecutar con independencia de unos con otros, lo que evita el conflicto de dependencias. Además, se permite que cada componente del software tenga su propio escalado. Una vez desarrollado el conjunto del software, los contenedores se pueden ejecutar en cualquier máquina soportada, sin necesidad de gestionar algo más que el motor de Docker. Añadido a Docker, para organizar los múltiples contenedores se utiliza **Docker-compose**. Esta tecnología permite describir la

configuración de contenedores, la comunicación y las dependencia entre ellos mediante un fichero. A partir de este, el conjunto del software se puede manejar sin necesidad de ejecutar cada contenedor por separado.

A continuación, se describe la tecnología que se usará en el desarrollo de cada componente de la plataforma *IoT*:

- **Reverse Proxy & TLS Termination:** En este componente se emplea **nginx**. Este software comenzó como un servidor web de código libre con un alto rendimiento, pero con el tiempo ha ido añadiendo múltiples funcionalidades como las que nos concierne, terminación *TLS* y proxy inverso. Nginx utiliza una arquitectura asíncrona controlada por eventos, lo que le permite manejar múltiples conexiones simultáneas con gran rendimiento.

- **SPA web:** Dentro de los tipos de aplicaciones web, las *single-page application* (*SPA*) tienen cada vez más presencia. Una *SPA* es una aplicación web que proporciona una interfaz dinámica al usuario sin tener que recargar toda la web con nuevos datos del servidor. De esta manera se consigue la percepción de utilizar una aplicación nativa de escritorio o móvil.

  Los *frameworks* más conocidos para realizar *SPA* son Angular, React y Vuejs. En este trabajo se utilizará **Vuejs**, que además proporciona los módulos **Vuex** y **Vuejs Router**. Vuejs es un *framework* progresivo de código libre para realizar interfaces de usuario. Por otra parte, Vuex es una librería para Vuejs que facilita el control del estado de la aplicación. Finalmente, con Vuejs Router se añade la posibilidad de navegar entre vistas anidadas.

- **Certificate Generator:** La función terminación *TLS* de nginx necesita una pareja de claves para su criptografía asimétrica, clave pública y clave privada, las cuales tienen un tiempo de expiración desde que se generan. Como entidad de certificación se emplea **Let's Encrypt**, la cual proporciona certificados gratuitos y es apoyada por la Fundación Linux. Por otra parte, para automatizar el proceso de generación de claves se utiliza la herramienta de codigo libre **cerbot**.

- **Users and Warnings SQL DB:** Los datos de usuarios y avisos de la plataforma *IoT* se organizan en bases de datos relacionales con **PostgreSQL**.

PostgreSQL es una base de datos relacional orientado a objetos de código libre.

- **MQTT Broker:** Para gestionar los mensajes que se reciben y envían en una red MQTT se utiliza **Mosquitto**. Se trata de un software de código libre implementando en C por la fundación Eclipse. Este *broker* tiene la capacidad de gestionar el protocolo MQTT convencional por *TCP* y utilizando la capa de *websockets*, por lo que permite la conexión de los nodos *IoT* y a los usuarios monitor mediante la aplicación web. Además, el software tiene una gran comunidad que desarrolla expansiones, como puede ser añadir la capacidad de autorización de los usuarios de la red.

- **API Gateway:** En la implementación del patrón de diseño *API Gateway* se emplea el software **Express Gateway**. Este software libre basado en Node.js y Express permite orquestar las *API* de diferentes microservicios con un fichero de configuración. Tiene la capacidad de dirigir las rutas de las peticiones entrantes a cada *API* con reglas globales e individuales, como la comprobación de *tokens*, filtración de métodos y la modificación del cuerpo de la petición.

- **Warning BOT & API:** Para implementar esta *API* se emplea el software libre **Flask**. Se trata de un *framework* web para el lenguaje de programación Python. Proporciona herramientas y librerías para desarrollar fácilmente una aplicación web. Lo caracteriza ser un *micro-framework*, que indica tener pocas dependencia y ser un software liviano.

- **Logged Data y Auth/Users API:** En estas otras *API* el software utilizado es **Nodejs** y **express**. Nodejs es un entorno de ejecución que permite llevar el lenguaje javascript al servidor, mientras que express se utiliza como *framework* para crear aplicaciones webs en Nodejs. Este entorno de ejecución destaca por su gran capacidad de manejar un alto número de peticiones simultáneas.

- **Time Series DB:** A diferencia de las bases de datos SQL, las medidas de los nodos *IoT* se almacenan en base de datos de series temporales, **influxdb**. Es

una base de datos de código libre escrita en el lenguaje de programación Go y está optimizado especialmente para el uso en el internet de las cosas. Sin ser una base de datos relacional, proporciona un lenguaje parecido a SQL para facilitar su uso.

- **Metrics Collector:** Como software de recolección de medidas se utiliza **Telegraf**. Este software libre permite hacer de puente entre diferentes tipos de fuentes de datos, como dispositivos *IoT* o bases de datos y la base de datos de series temporales.

4.3.2 Herramientas

Para el desarrollo de esta plataforma es ha utilizado el editor **VSCode** por las numerosas ventajas ya mencionadas, como la compatibilidad con cualquier lenguaje de programación existente mediante extensiones.

Por otra parte, para facilitar las pruebas durante el desarrollo y al final de este, se han empleado las herramientas **MQTT explorer** e **Insomnia**.

4.3.2.1 MQTT explorer

Esta aplicación es un cliente MQTT que permite obtener la actividad de todo el *Broker MQTT* y simular un cliente. Proporciona filtro de *topics* y la capacidad de generar gráficas de mensajes publicados.

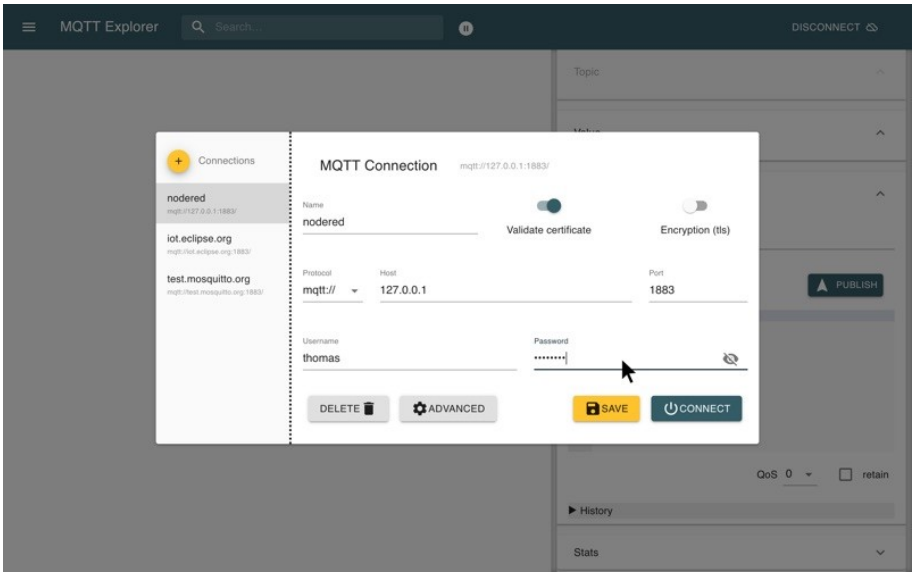

Figura 4.10: Interfaz de la aplicación MQTT explorer

4.3.2.2 **Insomnia**

La herramienta Insomnia da la posibilidad de interactuar con una *API* sin un navegador web. Se puede crear peticiones con cuerpos *json* y cabeceras personalizados, incluido los *tokens* de autenticación, y obtener la respuesta junto con métricas de rendimiento.

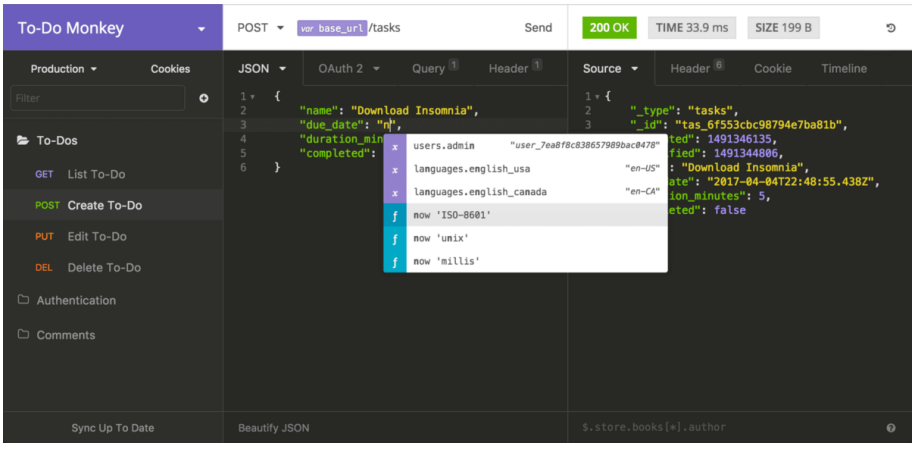

Figura 4.11: Interfaz de la aplicación Insomnia

CAPÍTULO

5

Desarrollo de la solución e implantación

En este capítulo se detalla el desarrollo de los 3 sistemas, comentando las decisiones tomadas y las particularidades de la solución. Finalmente, se comenta su implantación para realizar las pruebas.

5.1 Software embebido

Una vez definidas las interfaces de las clases necesarias para cumplir las distintas funcionalidades, en este apartado se describen las consideraciones a destacar en cada implementación, además de las tareas de *RTOS* realizadas en la aplicación principal que utiliza estos módulos.

5.1.1 Clases de módulos

- **Clase PCNT_encoder**: En la implementación de esta funcionalidad se acude a la librería $< driver/pcnt.h >$ propia de ESP-IDF. Durante la inicialización del contador de pulsos, se establece el pin indicado como entrada digital para realizar la cuenta en los flancos de subida.

- **Clase ADS1015**: Para realizar el driver del *ADC*, en la comunicación *I2C* se ha empleado la librería $< Wire.h >$. Este *ADC* presenta 4 registros de 16 bits [9]:

  - *Conversion register*: Contiene el valor de la última conversión en un entero con signo de 12 bits.
  - *Config register*: Permite la configuración del *ADC*.
  - *Lo_thresh* y *Hi_thresh*: Fijan los valores límites para la función de comparador.

  Para poder acceder a estos 4 registros, primero se escribe en un registro especial, *pointer register*, donde se indica en cual de los 4 se realizará la escritura a posterior. Debido a esto, el procedimiento para modificar los registros de 16 bits del *ADC* sigue el siguiente procedimiento:

  1. Escribir en el *pointer register* cual de los 4 registros se pretende escribir.
  2. Escribir el byte más significativo.
  3. Escribir el byte menos significativo.

  Durante la inicialización del *ADC* se realizan las siguientes configuraciones:

  - Generar un pulso de interrupción cuando una conversión es completada y puede ser leída.
  - Desactivar el *latching* del comparador.
  - Polaridad del comparador en nivel bajo cuando se activa.
  - Frecuencia de muestreo a 860 *SPS*.
  - Conversión en modo continuo. Con ello, el *ADC* realiza conversiones en todo momento a la frecuencia configurada, enviando un pulso por el pin de alerta/interrupción por cada conversión completada.
  - Ajustar la ganancia programable.
  - Entradas en modo diferencial.
  - Habilitar Pin de *Conversion ready interrupt*.

Finalmente, para obtener el valor de una conversión, bajo demanda, se escribe en el *pointer register* la dirección del registro *conversion register* y se espera la recepción de dos bytes para formar el valor con signo de 12 bits convertido a 16 bits.

- **Clase MCP4922**: El driver realizado para el *DAC* MCP4922 utiliza el protocolo de comunicación *SPI* con librería $< SPI.h >$. El funcionamiento de este encapsulado se basa en un único registro de 16 bits, de los cuales, los 12 bits menos significativos conforman el valor a escribir en el *DAC*, mientras que los 4 bits más significativos son bits de control [7]. Los bits de control tienen las siguientes funciones:

  - Bit 15: Selección entre el DAC-A o DAC-B en la escritura.
  - Bit 14: Control de entrada $Vref$ *buffered* o *unbuffered*.
  - Bit 13: Selección de ganancia.
  - Bit 12: Apagado del *DAC*.

  Durante la inicialización del *DAC* se configura la comunicación *SPI* a $1\,MHz$ en modo SPI0. Por otra parte, al tratarse de un único registro, en una misma escritura por *SPI* se proporciona el valor de salida del *DAC* y la configuración.

  En el método de escritura del driver, donde se proporciona como argumento el valor de salida del *DAC*, se le aplica a este una máscara con los 4 bits de configuración:

  - DAC-A seleccionado.
  - *Buffer control* como *unbuffered*, de manera que el rango de salida sea de $0 - V_{dd}$.
  - Ganancia de 1x.
  - Apagado desactivado.

  En la escritura se proporciona primero el byte menos significativo y, finalmente, el byte más significativo.

- **Clase NHD0420_SPI**: Como se ha indicado en el diseño del circuito, el *LCD* se ha configurado con el *jumper* para comunicación *SPI*. Esta comunicación se realiza en modo SPI3 y hasta una frecuencia de $100\,kHz$ [3]. La metodología para enviar instrucciones al *LCD* se diferencia en si se trata de un carácter *ASCII* a mostrar o un comando propio.

  En el caso de los comandos, el manual proporciona una tabla con el valor del comando en 8 bits, el número de parámetros necesarios tras este y el tiempo de ejecución del mismo. Para que el *LCD* tenga constancia de que se le envía un comando, también es necesario enviar primero un prefijo de valor $0xFE$. Un ejemplo de configuración del brillo del *LCD* podría ser:

  1. Enviar prefijo $0xFE$.
  2. Enviar el valor del comando de configuración del brillo, $0x53$.
  3. Enviar un byte con el valor del brillo a configurar.
  4. Esperar $100\,\mu S$ antes de enviar otro comando.

  Por otra parte, cuando se envía un carácter *ASCII*, simplemente se envía un valor entre $0x20 - 0x7F$. Por lo que para enviar una cadena se realiza un bucle hasta el símbolo '\0'que indique la final de la cadena.

- **Clase BLE**: Esta clase utiliza las librerías de ESP-IDF $< BLEDevice.h >$, $BLEServer.h$, $BLEUtils.h$ y $BLE2902.h$. Cabe destacar que, en la clase de conexión *Bluetooth*, se ha seguido un patrón de diseño ***facade***. [17] Mediante este patrón de diseño, se proporciona una interfaz unificada de alto nivel para manejar un subsistema de manera sencilla. De esta manera, se abstrae de otro conjunto de clases necesarias para realizar la funcionalidad que se pretende. En la siguiente figura se encuentra un esquema de las diferentes clases que debe gestionar la clase ***BLE***:

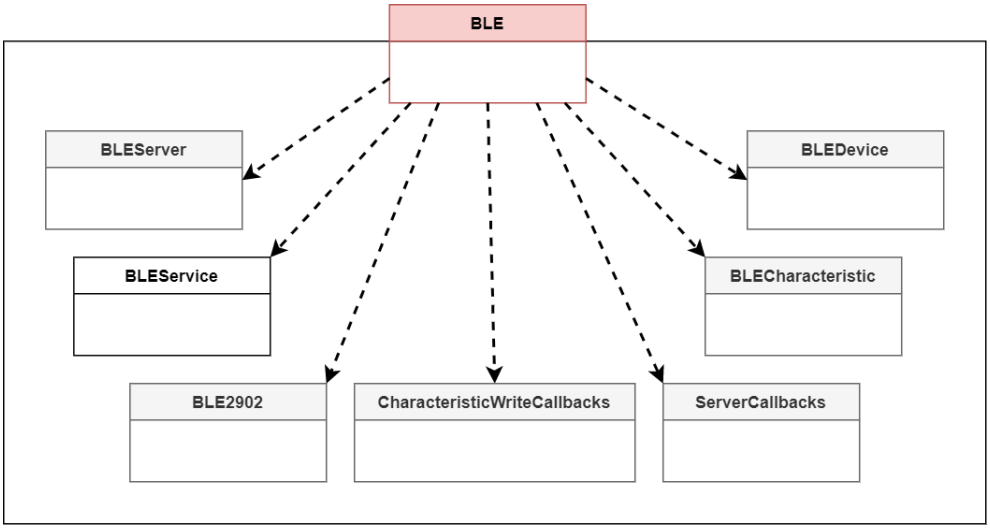

Figura 5.1: Clase *BLE* con patrón *Facade*

El microcontrolador ESP32 actúa como periférico en cuanto a topología de red al advertir de su presencia para que un dispositivo central (teléfono inteligente) se conecte. En referencia al protocolo *GATT*, el dispositivo se comporta como servidor para organizar los datos.

El procedimiento de inicialización es el siguiente:

1. Inicialización del dispositivo con el nombre definido.
2. Creación del servidor *BLE*.
3. Registro de dos funciones *callback* para el servidor, las cuales se ejecutan cuando se efectúa una conexión o desconexión en este servidor generado.
4. Creación de un servicio con dos características, formando una comunicación bidireccional con el cliente, la aplicación móvil.
   - *pCharacteristicNotify*: Esta característica es de tipo notificación, de manera que se avisa al cliente del servidor cuando ocurre una escritura en el campo de datos. Esta característica se emplea para enviar los valores de las mediciones de Par y velocidad al cliente.

- *pCharacteristicWrite*: Con esta característica de tipo escritura, se reciben los datos enviados desde el cliente. Mediante el registro de una función *callback*, se procesa la cadena de caracteres recibida y actualiza las variables correspondientes.

5. Actualización de la variable de estado si la inicialización se realiza correctamente. Nota: Esta variable y las que almacenan los valores enviados por el cliente, como la configuración de la pantalla, se pueden retornar desde las tareas de la aplicación principal con los métodos *get* listados anteriormente.

Finalmente, cuando se pide la desconexión, se rompe la comunicación que hay en el momento o se deja de notificar la disponibilidad del dispositivo en caso de no existir ninguna.

- **Clase MQTT**: Esta clase realiza la conexión Wi-Fi con la red configurada mediante la librería $< WiFi.h >$ y se inicia un cliente MQTT con la librería $< PubSubClient.h >$ cuando se le llama a inicializar. El estado de la conexión se guarda en una variable que permite informar a la aplicación móvil del mismo. Finalmente, cuando se le indica enviar un valor a la plataforma *IoT*, como *topic* se utiliza el propio nombre del dispositivo que lo identifica.

5.1.2 Programa principal

Una vez implementados los diferentes módulos que abstraen el hardware que interacciona con el medio físico, en el programa principal estos se emplean para realizar la funcionalidad deseada.

Tras importar las interfaces de los módulos, se definen los pines y datos que se les pasan a estos módulos para instanciarlos. De esta manera, se pueden cambiar los pines de los dispositivos hardware u otros datos como el identificador *Bluetooth* del dispositivo desde una configuración centralizada.

Al utilizar un sistema operativo de tiempo real, se definen varias tareas que se ejecutan de manera concurrente según la prioridad de cada una. Además, a estas tareas se les asigna una identificación, un tamaño de *stack* y un puntero hacia la función que ejecutará.

Identificador	Prioridad	Núcleo
ADCTask	Alta	0
mainTask	Alta	1
BLEStateTask	Baja	1

Tabla 5.2: Tareas del RTOS

Por otra parte, dos dispositivos realizan interrupciones de hardware, las cuales suspenden la tarea en ejecución para dar paso a su rutina. Para mayor optimización, la función que maneja la interrupción (*ISR*) se define con un atributo especial de *GCC* y así cambiar las secciones de este código, que se almacenarían en las secciones *.text* y *.data* de la memoria externa *FLASH*, a una sección especial, *.iram1*, que luego el *linker script* utlizado por las herramientas de *cross-compilación* del ESP32 lo almacenan en la memoria interna *SRAM*, permitiendo un acceso rápido y alterar lo menos posible la ejecución de las tareas.

Identificador	Configuración de interrupción
button_ISR	Flanco de subida del botón de activar/desactivar conexión *BLE*
ADC_ISR	Flanco de bajada del pin ALERT del *ADC* ADS1015, el cual informa que se ha realizado una conversión analógico-digital.

Tabla 5.4: *ISRs* del *RTOS*

Como se ha mencionado, las rutinas de las interrupciones deben ser muy rápidas para no alterar la ejecución de las tareas, por lo que se deben ejecutar un código muy liviano, que se limite a pequeñas comprobaciones y avisar a las tareas de su ejecución. Estos avisos se realizan con el mecanismo *Event groups* que proporciona *RTOS* y dispone de funciones para avisar y esperar por un evento ya sea desde un *ISR* o una tarea. Dicho esto, el siguiente paso del programa principal es registrar los dos *Event group*, uno por cada interrupción.

Además, una situación que se puede dar en el funcionamiento del programa es que una tarea o *ISR* esté utilizando un módulo de hardware y, en ese instante, otra tarea o *ISR* le interrumpa y para usar el mismo módulo. Esto provocaría un fallo de configuración y podría inutilizar el sistema completo. Para solucionar este escenario, se utiliza el mecanismo de sincronización *Mutex*, también proporcionado por el sistema operativo de tiempo real. La interfaz del módulo *BLE* se utilizará por las tareas **mainTask** y **BLEStateHandle**, por lo que se se define un *Mutex* para el módulo de conexión *Bluetooth*.

Cabe destacar que, para almacenar los valores tomados por el *ADC* se define un *buffer* y un puntero que contiene el índice del siguiente espacio libre para escribir.

Finalmente, se describirá las acciones realizadas por las tareas y los manejadores de interrupciones.

- **button_ISR:**
  1. Comprobar el tiempo que ha pasado entre la actual interrupción y la anterior para evitar el rebote del interruptor.
  2. Si ha pasado más de *800 ms* desde la anterior pulsación, enviar un aviso con el *Event Group* a la tarea **BLEStateTask**.
  3. Actualizar tiempo de última interrupción.
- **ADC_ISR:**
  1. Enviar un aviso con el *Event Group* a la tarea **ADCTask**.
- **BLEStateTask:** Se realizan en bucle las siguientes acciones:
  1. Esperar por el aviso de **button_ISR**.
  2. Almacenar el estado del módulo *Bluetooth* con el *mutex*.
  3. Según el estado de la conexión, activar o desactivar utilizando el *mutex*.
- **ADCTask:** Se realizan en bucle las siguientes acciones:
  1. Esperar por el aviso de **ADC_ISR**.

2. Si el el puntero del *buffer* no ha llegado al final, almacenar el valor leído por el *ADC* en este e incrementar el índice del puntero.

- **mainTask:** En la inicialización, las acciones son:

  1. Inicializar el contador de pulsos, el *DAC* MCP4922, el *ADC* ASD1115 y el *LCD*.
  2. Imprimir en el *LCD* los textos fijos "Velocidad", "Par", etc.

  Finalmente, en un bucle que se repite cada $500\,ms$ se ejecutan las siguientes acciones:

  1. Obtener el número de pulsos del encoder desde el último bucle mediante el módulo del encoder.
  2. Convertir los pulsos en este intervalo de $500\,ms$ a rpm.
  3. Enmascarar la interrupción del ADC para que deje de almacenar lecturas.
  4. Detectar 3 cruces con cero de la onda almacenada en el *buffer*.
  5. Si se encuentran 3 cruces con cero, calcular la tensión *RMS* y convertirlo a valor de par.
  6. Limpiar el *buffer* a ceros y poner el puntero del mismo en la primera posición.
  7. Desenmascarar la interrupción del ADC para que empiece a almacenar lecturas en el *buffer* hasta el próximo bucle.
  8. Tomar los valores de configuración de la pantalla y estado de la conexión *Bluetooth* desde el módulo *BLE* utilizando el *mutex*.
  9. Imprimir en el *LCD* los valores de velocidad, par y estado de la conexión *Bluetooth*, además de cambiar la configuración de la pantalla según los valores obtenidos.
  10. Si hay un cliente conectado por *Bluetooth*, enviar las lecturas de velocidad y par con el *mutex*.

11. Si hay una conexión con el *broker* MQTT, se envían las lecturas de velocidad y par.
12. Limpiar la cuenta de número de pulsos.
13. Suspender la tarea durante $500\,ms$.

5.2 Aplicación móvil

En esta sección se explica la implementación del diseño de la aplicación móvil.

5.2.1 Dependencias

Para el desarrollo de la aplicación se han acudido a los siguientes paquetes de Dart/Flutter:

- **provider:** Paquete para el manejo del estado de la aplicación recomendado por Flutter.
- **barcode_scan:** Proporciona una una pantalla de escáner para códigos de barras y QR, devolviendo la lectura obtenida y permitiendo la activación del *flash* de la cámara.
- **flutter_blue:** Librería para realizar conexiones y transferir datos de Bluetooth Low Energy con dispositivos externos.
- **fl_chart:** Librería para dibujar gráficas reactivas a datos de la aplicación.
- **sleek_circular_slider:** Paquete para generar animaciones circulares personalizables.
- **url_launcher:** Permite lanzar enlaces URL en el navegador web predeterminado del usuario.
- **font_awesome_flutter:** Permite utilizar los iconos *font awesome* en una aplicación Flutter.
- **google_nav_bar:** Proporciona una barra de navegación animada y personalizable.

- **flare_flutter:** Genera un entorno de desarrollo para animaciones desarrolladas con flare.
- **share:** Permite compartir ficheros generados en la aplicación con las apps instaladas en el dispositivo móvil.
- **csv:** Incluye la capacidad de decodificar y codificar listas en o desde ficheros csv.
- **path_provider:** Proporciona la capacidad de trabajar con rutas locales en el dispositivo móvil.
- **intl:** Facilita la localización de la aplicación y tratar con formatos de texto, fechas y números.

5.2.2 Pantallas

Como se ha explicado, una aplicación de Flutter se basa en un árbol de *widgets* que describe toda la interfaz gráfica. En la **Figura 5.2** se muestra los principales *widgets* hasta el primer nivel de cada pantalla. La aplicación empieza con la función main que lo primero que carga es un *widget MultiProvider*, el cual instancia cada uno de los *Provider* que mantienen el estado de la aplicación y son accesibles a todos los hijos de este. Como único hijo de *MultiProvider* se encuentra el *widget MaterialApp* que proporciona la plantilla para una aplicación con aspecto *Material Design* de Google. Esta plantilla permite tener varias rutas, que en esta aplicación son la pantalla de escaneo de código de QR *QRScanPage* (ruta por defecto) y la pantalla principal *HomePage*. Finalmente, la pantalla principal puede mostrar las 4 vistas con sus *widgets RealTimeTab*, *ReportTab*, *SettingsTab* e *InfoTab*.

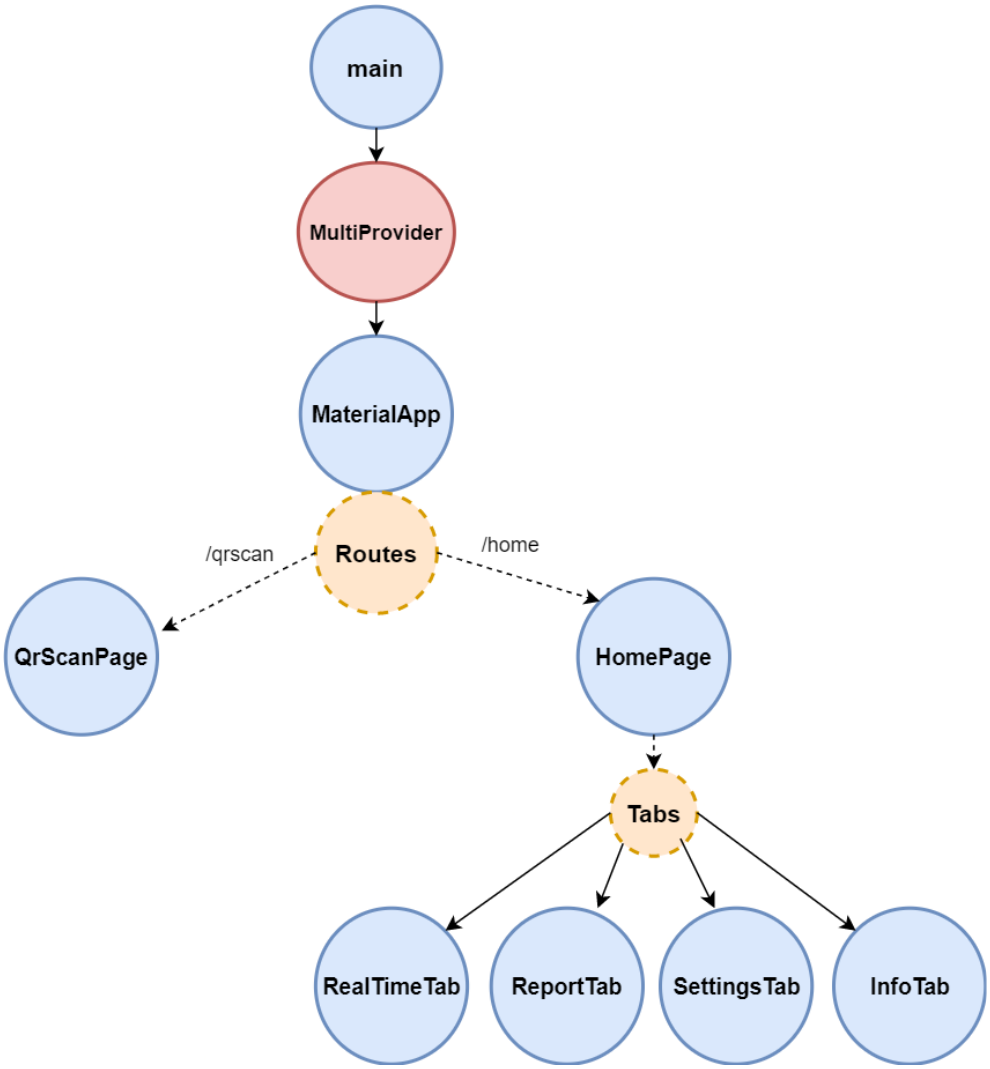

Figura 5.2: Vista general de la estructura de los *widgets*

A continuación, se comentarán detalles a destacar de cada una de las 5 pantallas:

- **QrScanPage:** La primera pantalla que ve el usuario es la que le permite conectarse al sistema embebido, la cual muestra un botón si la conexión Bluetooth del dispositivo móvil se encuentra activa, **Figura 5.3**. Para realizar la conexión se accede a la cámara con el botón *Scan* y se enfoca al código QR. Una vez la aplicación detecte el código, cierra la vista de la cámara e intenta la conexión con el dispositivo que lleve el nombre del código leído. Durante este proceso se muestra la animación de la **Figura 5.4 a)** y, en el caso de

no tener éxito en la conexión, se obtiene la alerta de la **Figura 5.4 b)**. En el caso deseado de realizar la conexión, la aplicación almacena los datos del dispositivo conectado en el *Provider* y pasa a la pantalla de *Home*.

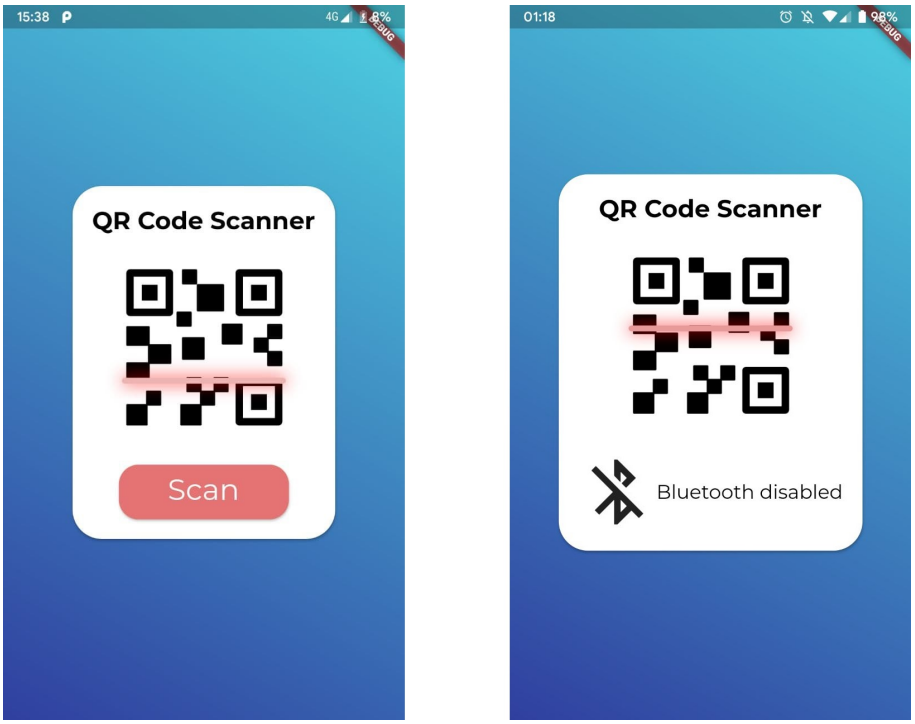

Figura 5.3: Pantalla de escáner inicial

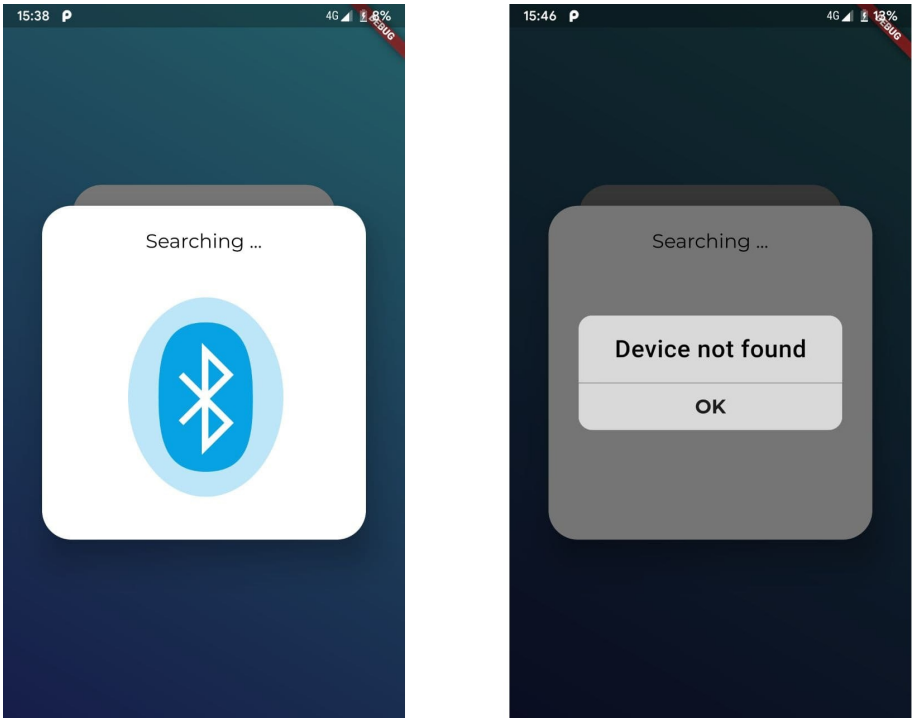

Figura 5.4: Pantalla de conexión tras el escáner

- **HomePage:** Tras realizar una conexión, la aplicación pasa a la pantalla de *Home*. La primera acción de esta pantalla es buscar en los servicios y características del dispositivo Bluetooth para suscribirse a las notificaciones de los valores de velocidad y par recibidos de forma asíncrona, que son escritos en el *Provider* de Bluetooth. Además, se inicia un *watchdog* que, ante una desconexión, lanza un mensaje de error, **Figura 5.5**, y vuelve a la pantalla de escáner. Esta pantalla se divide en 3 *widgets*, una barra superior con el nombre del dispositivo conectado, una zona principal donde se muestra la vista seleccionada y una barra de navegación inferior donde seleccionar otra vista.

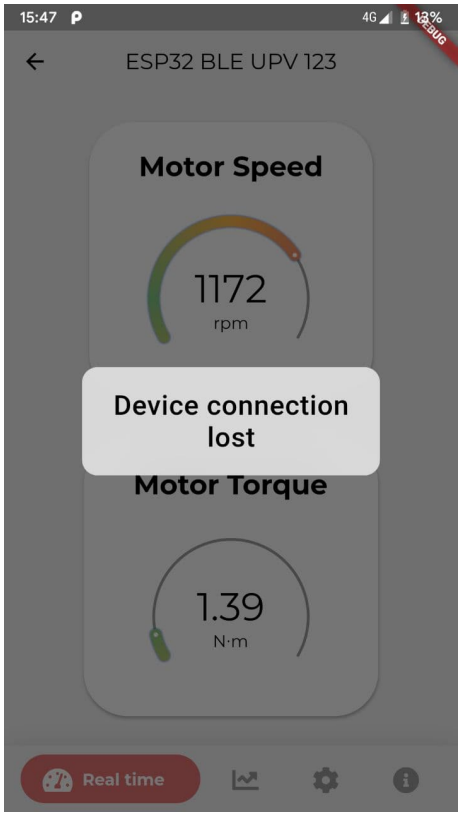

Figura 5.5: Mensaje de dispositivo perdido

– **RealTimeTab:** En esta vista, se sitúan dos animaciones circulares que reaccionan a los últimos valores recibidos por Bluetooth de velocidad y par, además de mostrarlos de manera numérica. Para ello, se obtienen los valores desde el *Provider* de Bluetooth, suscribiéndose a los cambios que puedan producirse.

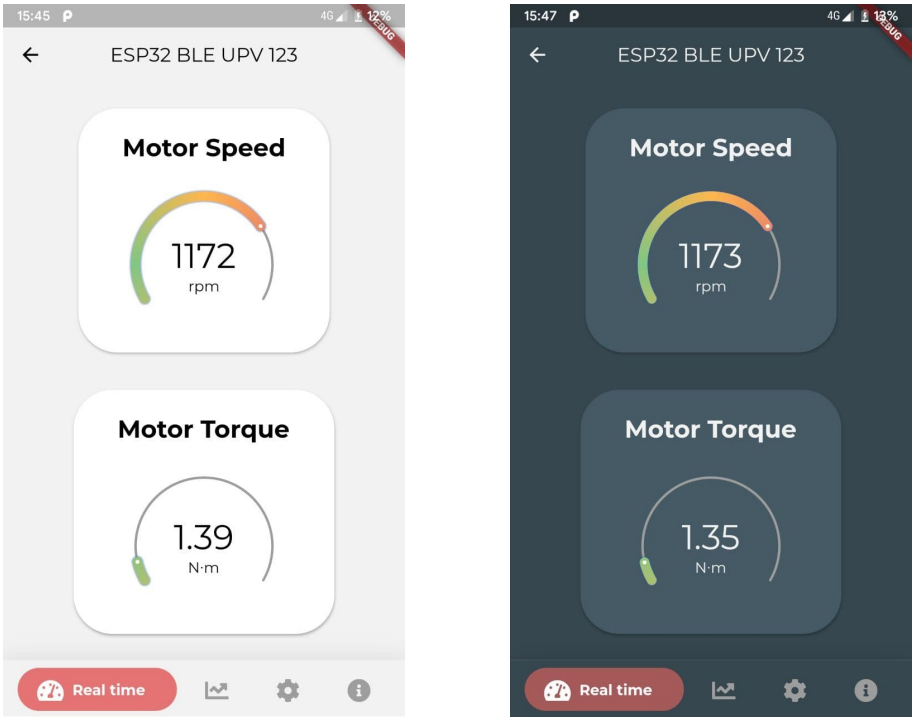

Figura 5.6: Pantalla de tiempo real

– **ReportTab:** Al igual que en la pantalla de valores en tiempo real, en todo momento se recibe un *stream* desde el *Provider* de la conexión Bluetooth, por lo que tenemos en todo momento los últimos valores recibidos, que son mostrados en la parte superior de cada gráfica como referencia en tiempo real. Los *widgets* principales que nos encontramos en esta pantalla son:

  * Botón *Capture*: Con cada pulsación, el valor actual de par y velocidad del *Provider* de Bluetooth son añadidos a las listas del *Provider* de valores capturados.
  * Botón *Clear*: Este botón emplea el método de vaciar las listas del *Provider* de datos capturados.
  * Botón *Share*: Con este botón se genera una alerta con una entrada de texto para exportar los valores capturados en la aplicación seleccionada, **Figura 5.8**. Las listas del *Provider* de datos capturados se convierten a un formato con dos decimales con coma (formato

español) y se crea un archivo csv de una ruta temporal listo para exportar.w

* Gráficas de par y velocidad: Estas gráficas reaccionan de manera reactiva a la lista de valores de par y velocidad del *Provider* de datos capturados.

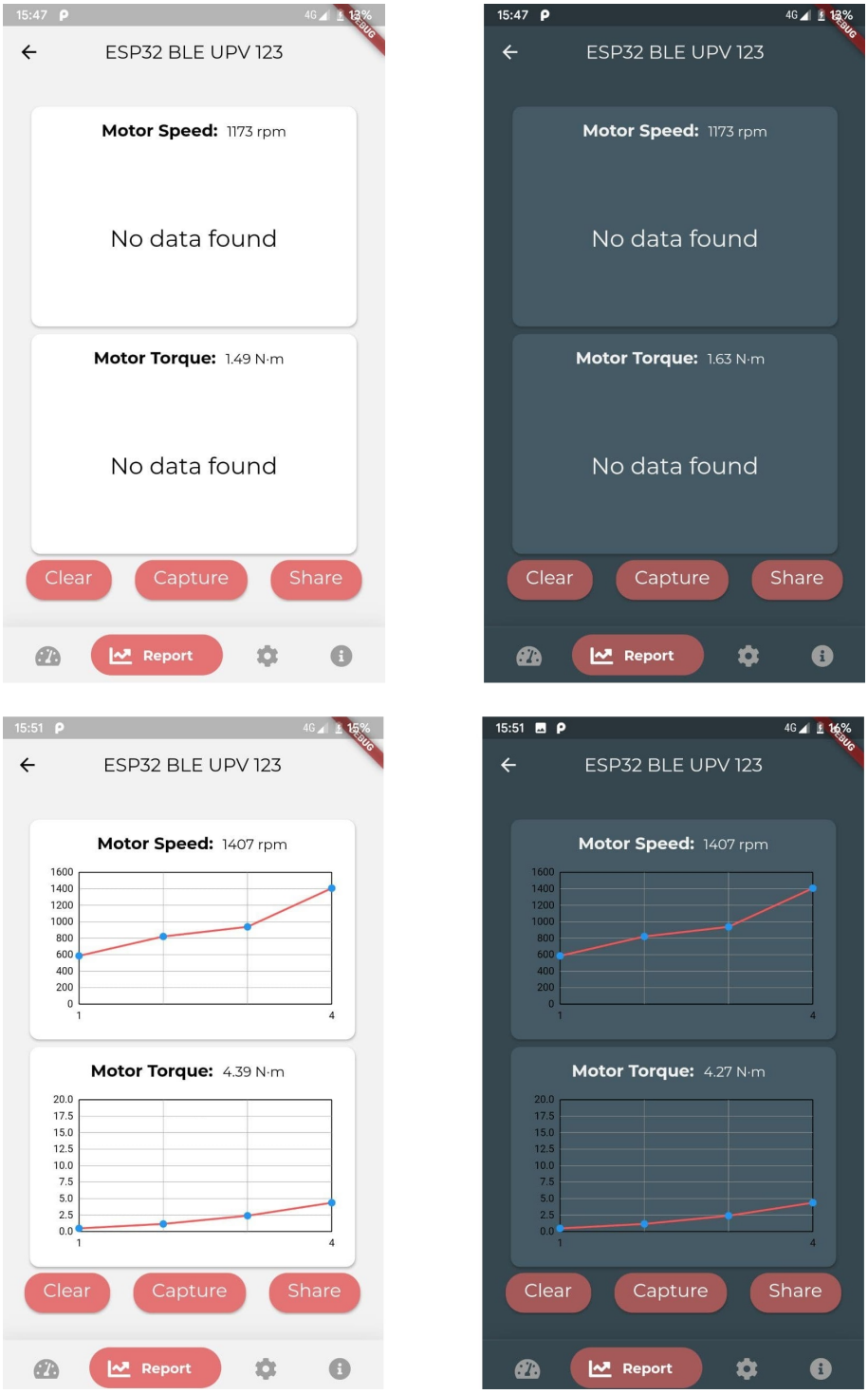

Figura 5.7: Pantalla de capturas

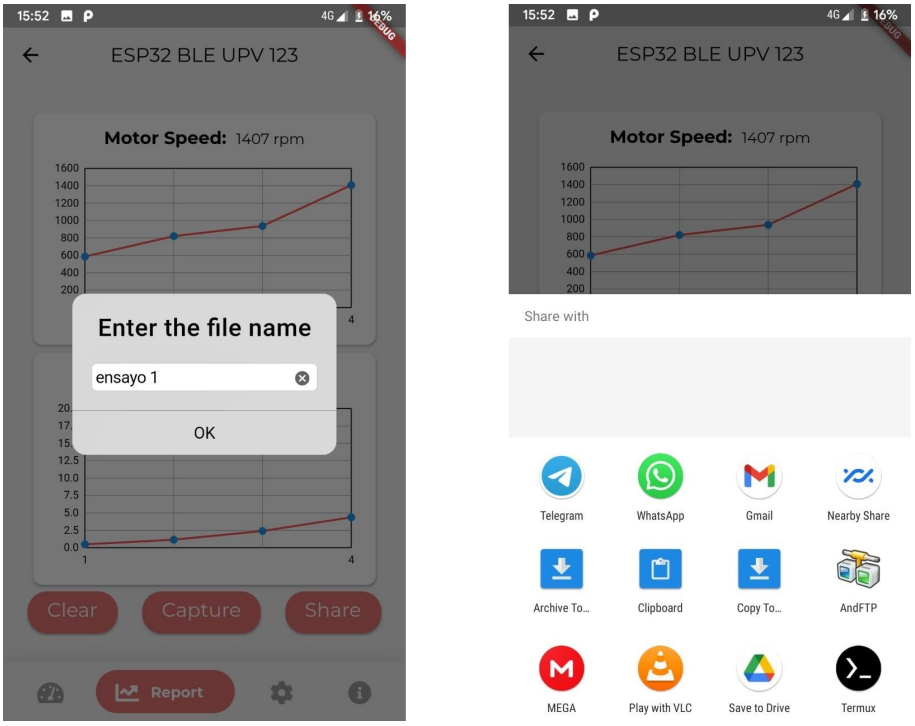

Figura 5.8: Pantallas de exportación

– **SettingsTab:** Esta pantalla permite realizar ajustes a la aplicación y al sistema embebido conectado. Cuando uno de los deslizadores detecta un cambio de posición acude al método del *Provider* de Bluetooth para enviar estos nuevos valores de configuración de la pantalla *LCD* del sistema embebido. De la misma manera se envía los ajustes de la red a la que se conectaría el sistema embebido. Se ha de tener en cuenta que este método tiene un límite de tiempo mínimo para envíos, ya que, en el caso contrario, en un ajuste del usuario sobrecargaría al sistema embebido con nuevos envíos por cada pequeño movimiento. Por otra parte, el interruptor de tema comprueba el tema actualmente seleccionado desde el *Provider* de gestión de temas para mostrar su estado. Con su modificación por parte del usuario cambia de tema, que, al estar todas las pantallas de la aplicación a la escucha de notificaciones del *Provider* de la gestión de tema, los cambios se ven reflejados de manera instantánea en toda la aplicación.

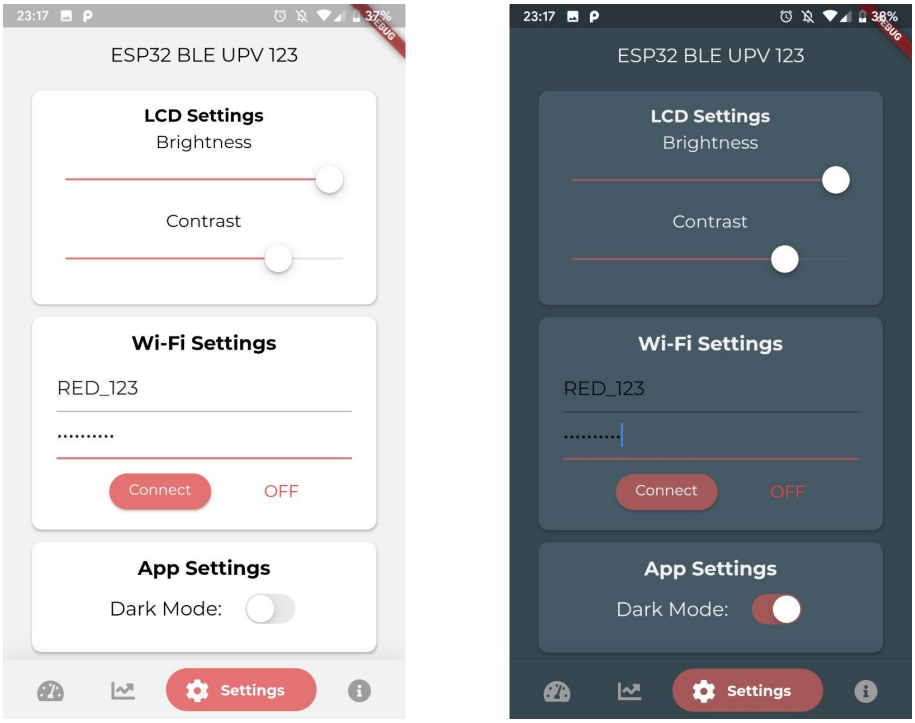

Figura 5.9: Pantalla de ajustes

– **InfoTab:** Esta última vista simplemente muestra unos *assets* con enlaces a modo de información.

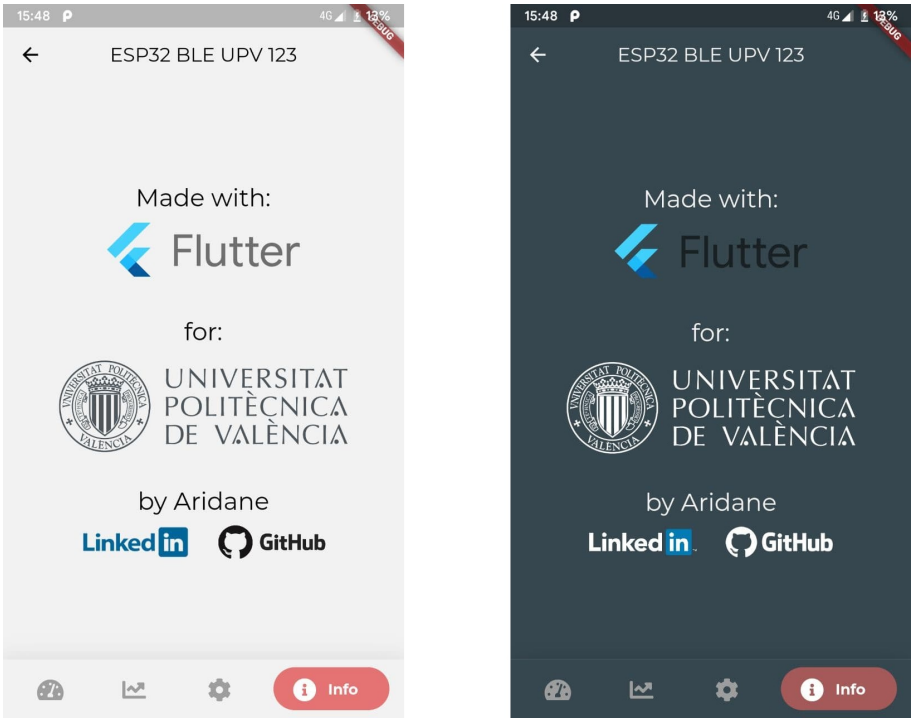

Figura 5.10: Pantalla de información

5.3 Plataforma *IoT*

Una vez decididas las tecnologías que se utilizan en la plataforma *IoT*, en la siguiente imagen se muestra el diagrama de la arquitectura con estas aplicadas. Se puede observar también los puertos por los que se comunican los contenedores. Los contenedores están con la configuración en modo puente, por lo que estos pueden comunicarse unos con otros dentro de la red de Docker mediante su *IP* o nombre del contenedor (gracias a su servidor *DNS* interno) y sus puertos.

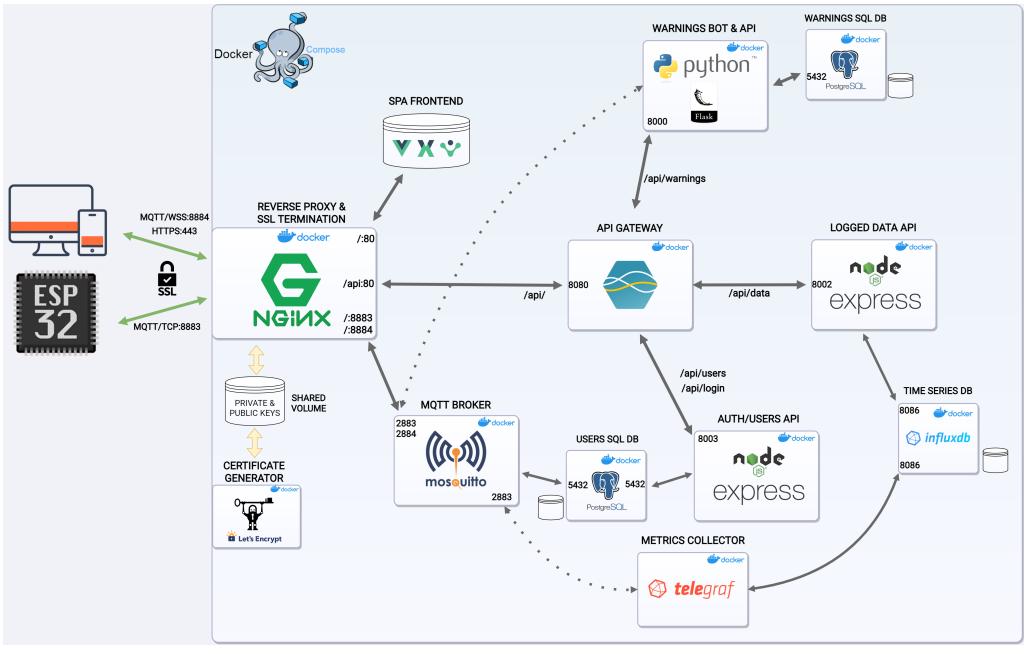

Figura 5.11: Diagrama de la arquitectura con sus tecnologías

En los siguientes apartados se comentarán las consideraciones que se han tenido en el desarrollo de cada componente de la plataforma.

5.3.1 Reverse Proxy & SSL termination

Este componente se forma a partir del Dockerfile, el cual, junto con la imagen oficial de nginx, crea una carpeta en el contenedor donde transfiere el contenido de la carpeta *dist*, que a su vez tiene la exportación de la web desarrollada.

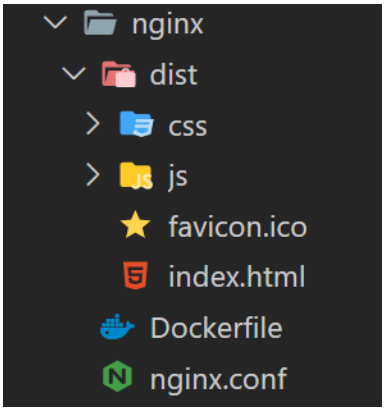

Figura 5.12: Directorio de desarrollo de *Reverse Proxy & SSL termination*

La configuración de nginx se realiza mediante el fichero *nginx.conf*. En este se configuran los 3 servidores para las conexiones de entrada:

- Servidor *HTTPS* en puerto 443: Se utiliza para las peticiones de la aplicación web a la *APIs*. Nginx permite crear un servidor web, por lo que la web alojada en el directorio *dist* se sirve directamente en este puerto.
- Servidor *TCP* en puerto 8883 con *SSL*: Los nodos *IoT* utilizan este puerto para enviar sus medidas por MQTT. Dentro de la red de Docker, el *stream TCP* se transfiere al contenedor del *broker MQTT* por el puerto 2883.
- Servidor *websockets* en puerto 8884 con *SSL*: Para obtener las medidas en tiempo real en la web de monitorización se utiliza este puerto, con el que se establece la conexión de MQTT sobre *websockets*. De igual manera que las conexiones MQTT directamente sobre *TCP*, estas se transfieren al contenedor del *broker MQTT* por el puerto 2884.

Para permitir a nginx que realice la función de terminación *SSL*, en los servidores se indica el protocolo de *SSL* utilizado y la localización del certificado y clave privada. De esta manera, la petición es desencriptada antes de ser redirigida al componente de la plataforma correspondiente.

5.3.2 SPA Frontend

Como se ha indicado, esta web se desarrolla en un directorio separado con la tecnología Vue.js, que también proporciona una herramienta para exportar la web almacenada en el contenedor de nginx. En la siguiente imagen se muestra la organización del desarrollo de la aplicación web:

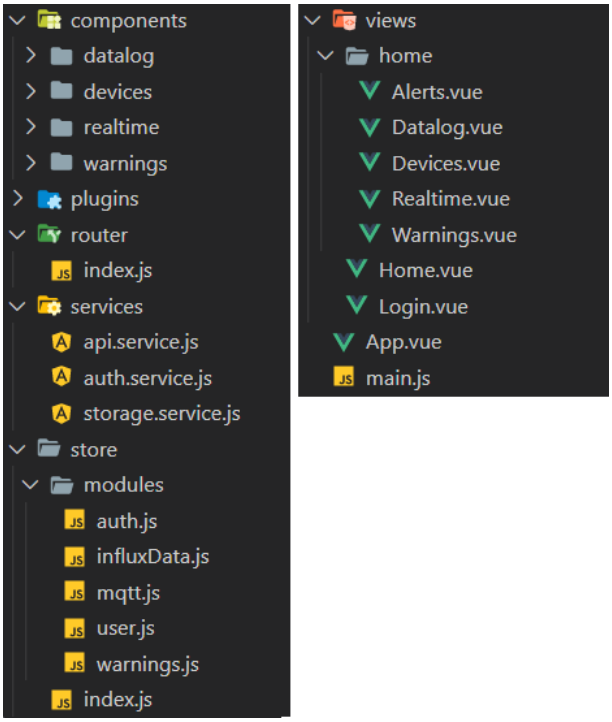

Figura 5.13: Directorio de desarrollo de *SPA Frontend*

La aplicación Vue.js comienza en el *main.js*, el cual crea la propia aplicación Vue e importa los módulos *store* que utiliza la extensión Vuex para controlar el estado de la aplicación y el módulo *router* que gestiona las rutas de la aplicación con la extensión Vue Router.

El estado de la aplicación se ha divido en módulos. Cada uno de los módulos centraliza datos relacionados que se pueden acceder desde cualquier parte de la aplicación, además de proporcionar unos métodos para obtener estos datos y modificarlos, ya que tratar directamente con ellos llevaría a un comportamiento impredecible en la aplicación. Los módulos son los siguientes:

- **auth.js:** Contiene el estado de autenticación y *token* si fuera el caso.
- **influxData.js:** Almacena las últimas medidas de un nodo *IoT* pedidas a la *API* junto con la identificación del nodo.
- **mqtt.js:** Gestiona datos relacionados con la conexión MQTT para la recepción de medidas en tiempo real.
- **user.js:** Tras realizar una autenticación con éxito, este módulo almacena la información del usuario, incluyendo su identificación, tipo de usuario y permisos sobre los nodos *IoT*.
- **warnings.js:** Contiene la lista de avisos creados por el usuario tras pedirlo a su *API*.

Además, en la carpeta *services*, se proporciona unos métodos para tratar con elementos externos a la aplicación, como son el almacenamiento local del navegador y las llamadas a las *APIs*.

- **api.service.js:** Proporciona servicios para realizar operaciones *CRUD* (*CREATE*, *READ*, *UPDATE*, *DELETE*) sobre las APIs de la plataforma gestionando el *token* de autenticación en las cabeceras.
- **auth.service.js:** Permite autenticarse y salir de la cuenta con sus métodos.
- **storage.service.js:** Gestiona el almacenamiento de *tokens* en el almacenamiento local.

En cuanto a las vistas, el primer elemento de la aplicación Vue que engloba a todos los demás componentes es la vista de *Router*, que registra las dos vistas posibles, **login** y **Home**. Esta última, a su vez, tiene 4 vistas anidadas, **Alerts**, **Datalog**, **Devices**, **Realtime** y **Warnings**.

Finalmente, siguiendo con la metodología de componentes, en la carpeta *components* se crean partes de la web que son reutilizables a partir de unos argumentos de entrada, obteniendo una web mejor organizada para su mantenimiento y ampliación.

A continuación, se muestran las vistas ya mencionadas con sus componentes.

- **login:** En esta primera vista de la aplicación se muestra un diálogo de autenticación. Para realizar esta acción se utilizan los módulos *auth.js* y *mqtt.js*. Si el proceso es correcto, se cambia a la ruta de la vista **Home** e inicia la conexión MQTT para recibir las medidas en tiempo real.

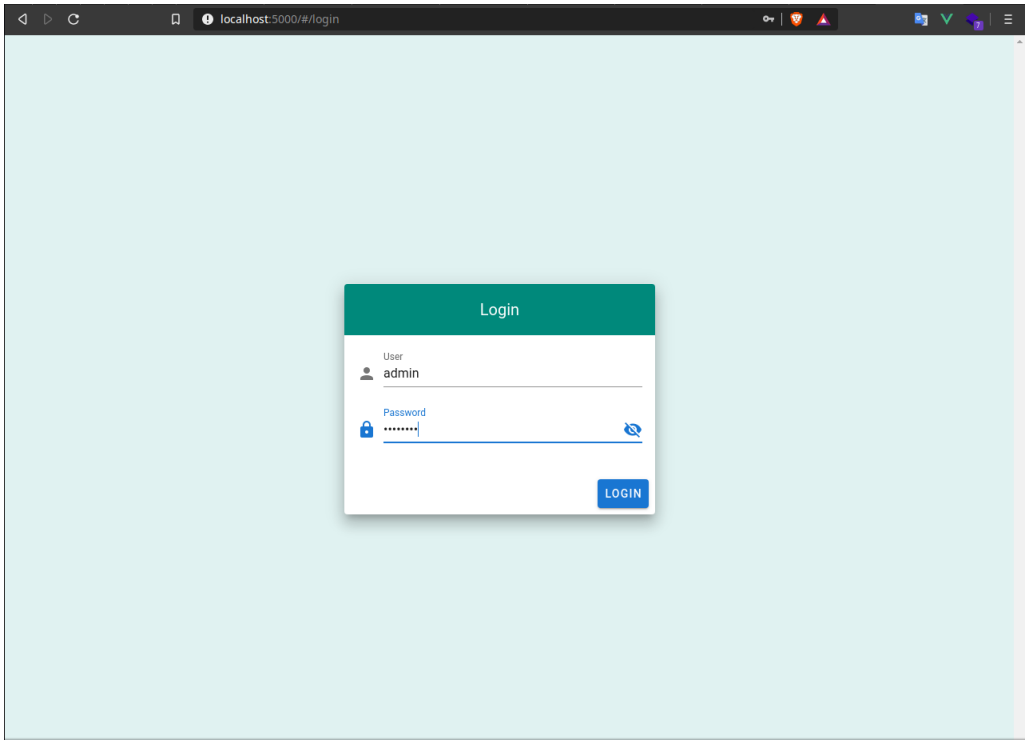

Figura 5.14: Vista de *Login*

- **Home:** En la vista principal de la aplicación se indica el usuario en la parte superior derecha, junto con la posibilidad de salir de la cuenta y volver a la vista de **login**. Además, en el menú izquierdo se da la posibilidad de cambiar entre las diferentes vistas que contiene anidadas **Home**. En esta vista se utilizan el módulo *auth.js* para permitir salir de la cuenta y *user.js* para obtener los datos del usuario.

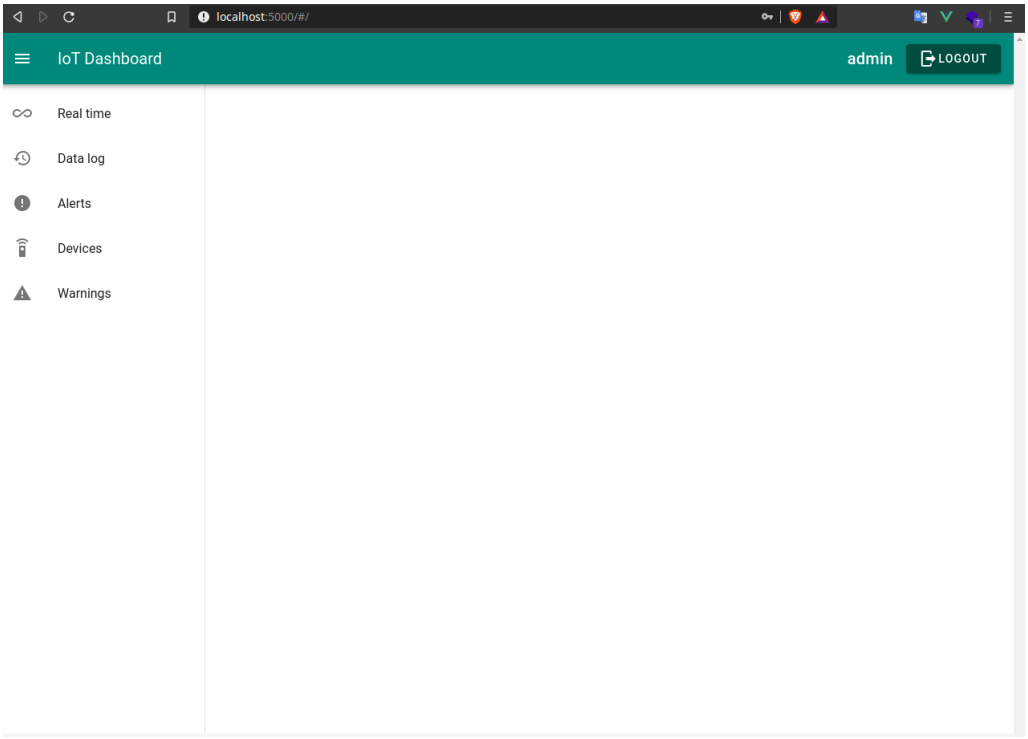

Figura 5.15: Vista de *Home*

- **Realtime:** Para mostrar las medidas en tiempo real de los nodos *IoT*, con el módulo *user.js* se obtienen los permisos del usuario. Según el nodo seleccionado se mostrarán los valores obtenidos en tiempo real.

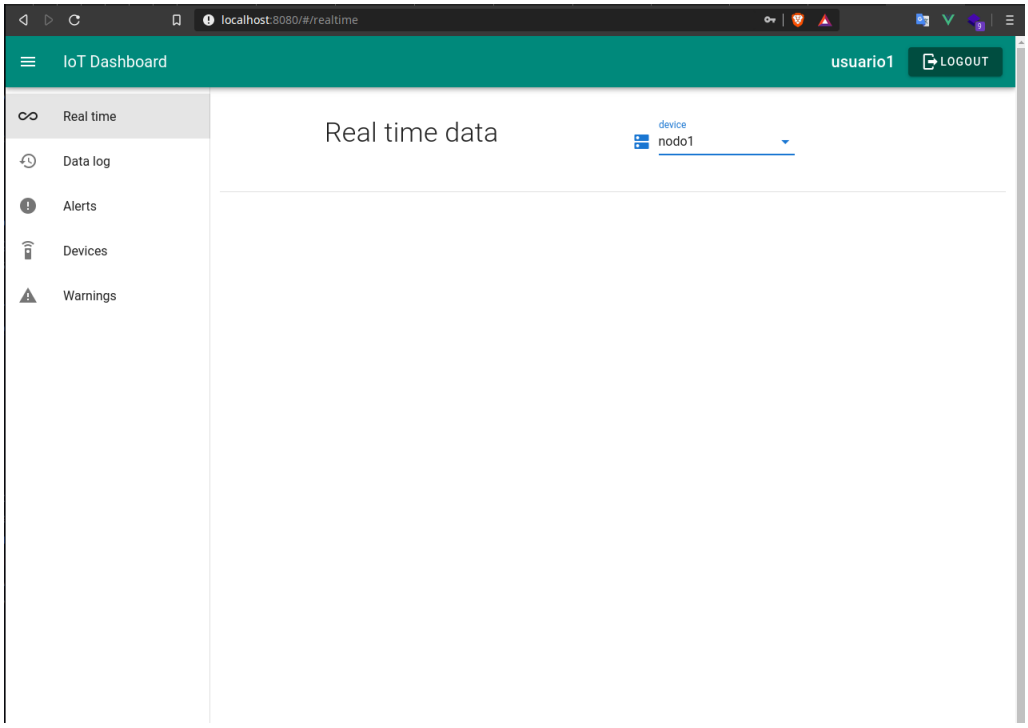

Figura 5.16: Vista de *Realtime*

- **Datalog:** De la misma manera, con el módulo *user.js* se obtienen los permisos del usuario y, a partir de un dispositivo seleccionado, con el módulo *influxData* se proporciona gráficas con las medidas entre dos tiempos seleccionados.

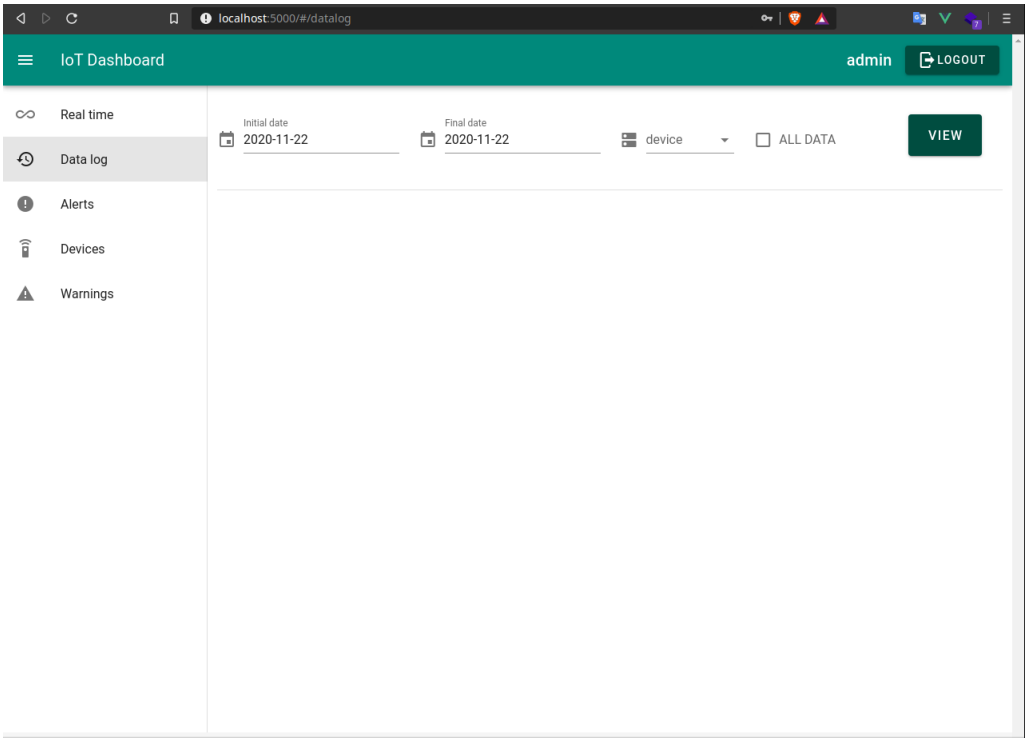

Figura 5.17: Vista de *Datalog*

- **Devices:** En esta vista se listan los dispositivos registrados sobre los que se tiene permiso con el módulo *user.js* y, en caso de ser administrador, se permite crear nuevos usuarios y eliminar uno existente.

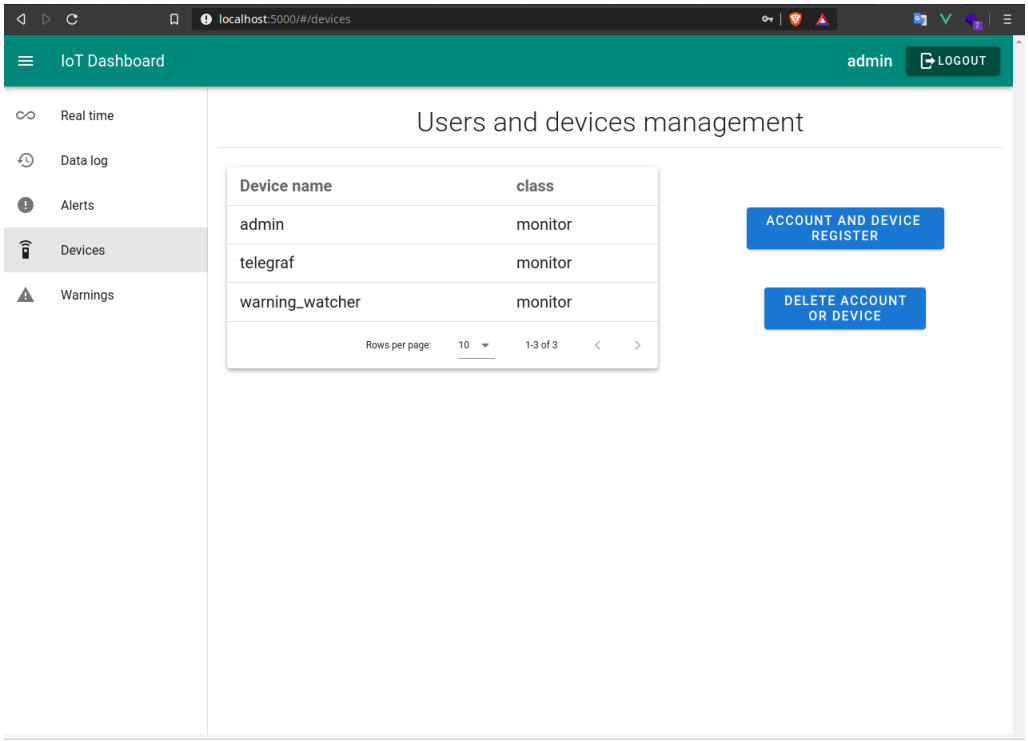

Figura 5.18: Vista de *Devices*

- **Alerts:** Igual que en la vista de **Datalog**, según los dispositivos sobre los que se tengan permisos, se muestras las todas las alertas junto con la identificación del dispositivo, clase y mensaje de la alerta.

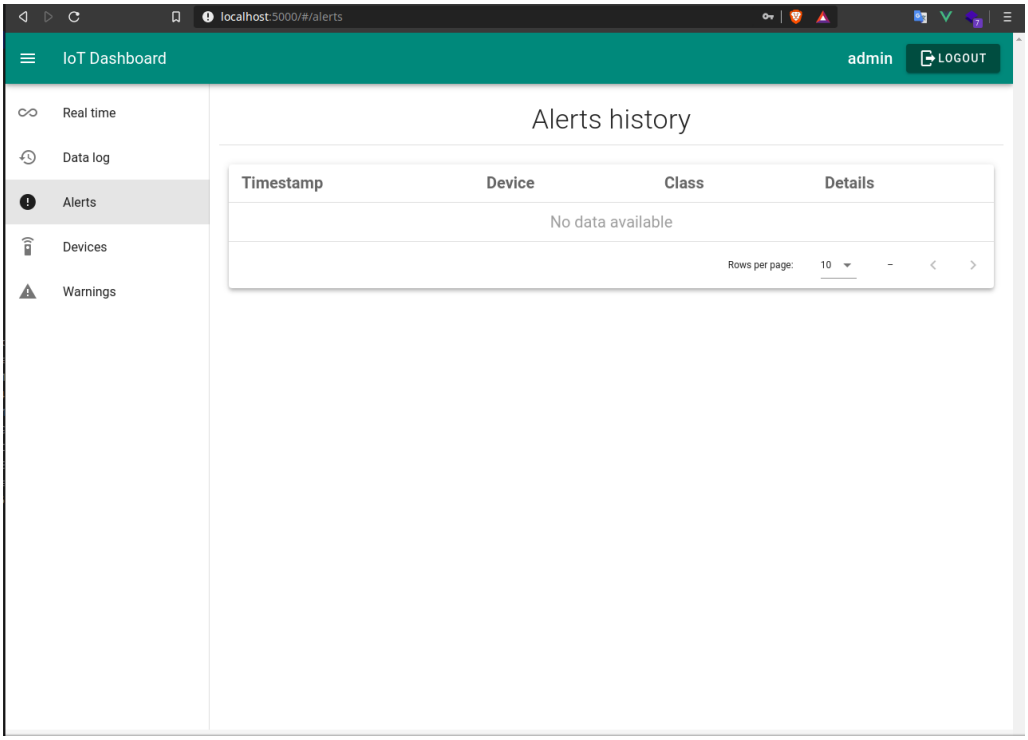

Figura 5.19: Vista de *Alerts*

- **Warnings:** En esta última vista de la aplicación, en una lista se encuentran los avisos creados por el usuario sobre los nodos *IoT* con el módulo *warnings.js* y permite crear nuevos avisos o eliminar uno existente. Además, se proporciona un enlace directo al *BOT* que comunica los avisos.

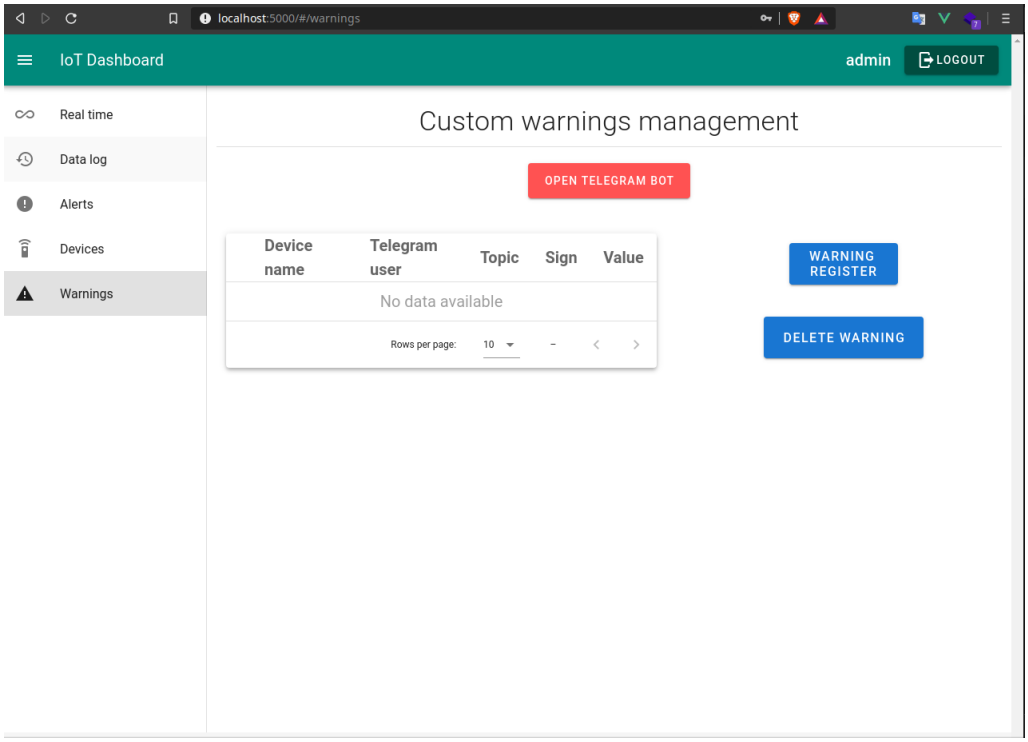

Figura 5.20: Vista de *Warnings*

5.3.3 Auth/Users DB

La base de datos donde se almacena la información de los usuarios se crea mediante el fichero Dockerfile, el cual carga una imagen de Postgres en un contenedor Docker y copia la configuración inicial de la base de datos del fichero *start.sql*.

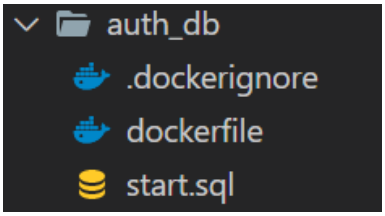

Figura 5.21: Directorio de desarrollo de *Auth/Users DB*

Esta configuración inicial crea 3 tablas:

1. **account:** Almacena los datos de usuario, que son el identificador único, usuario, contraseña, estado de administrador y el tipo de usuario.

Nombre	Tipo
id	Serial
user_mqtt	Varchar(15)
pass_mqtt	Varchar(80)
super	Smallint
user_class	Varchar(20)

Tabla 5.5: Tabla SQL account

2. **acls:** Contiene los *topics* MQTT que un usuario es capaz de interactuar. En el proceso de registro de un nodo *IoT*, se da permisos al usuario de escribir en los *topic* **data_log/<username>/#** y **alerts/<username>/#**. Por otra parte, a un usuario de monitorización se le da permisos de lectura en estos mismos *topics* de todos los nodos *IoT* sobre los que tiene permisos.

Nombre	Tipo
id	Serial
user_mqtt	Varchar(15)
topic	Varchar(20)
rw	Smallint

Tabla 5.6: Tabla SQL acls

3. **perm:** Esta última tabla guarda las relaciones de permisos de un usuario monitor sobre los nodos *IoT*.

Nombre	Tipo
id	Serial
user1	Varchar(15)
user1	Varchar(15)

Tabla 5.7: Tabla SQL perm

Tras crear las tablas, también se introducen las cuentas iniciales de administrador, *Warning watcher* y *telegraf* con permisos a todos los usuarios. Los dos últimos

son utilizados por el *BOT* de avisos y el recolector de métricas respectivamente para acceder al *Broker MQTT*.

5.3.4 MQTT Broker

El *Broker MQTT* se basa en un contenedor de Mosquitto, al cual se le integra una extensión para exigir la autenticación de los usuarios con la base de datos **Auth/User**. Además, el *script run.sh* que se ejecuta al comienzo del contenedor, el cual espera a que esté operativa la base de datos para continuar.

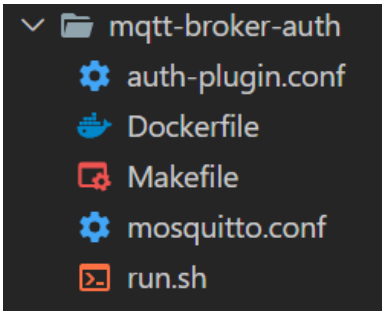

Figura 5.22: Directorio de desarrollo de *MQTT Broker*

La configuración del *Broker* y la extensión de autenticación se realizan en dos ficheros por separado. Por una parte, en el fichero *mosquitto.conf* se inhabilita la posibilidad de entrar como anónimo y se abren dos puertos de escucha, el 2883 por *TCP* y el 2884 con *websockets*. En cuanto a la configuración *auth-plugin.conf*, en este fichero se introducen los datos de la base de datos donde se obtienen las credenciales de los usuarios y las peticiones SQL que debe realizar para comprobar el usuario, estado de administrador y permisos sobre *topics*.

5.3.5 API Gateway

Todos los ficheros de la siguiente imagen son generados por el software con una configuración base y, para nuestra configuración, el único fichero modificado es la configuración *gateway.config.yml*.

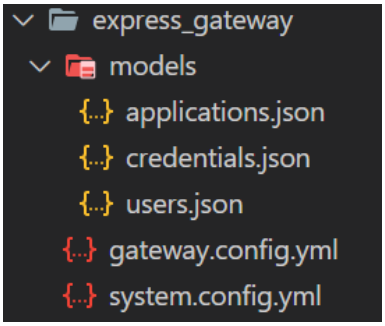

Figura 5.23: Directorio de desarrollo de *API Gateway*

Este fichero contiene el puerto de entrada (8080), los puntos finales donde se redirigen las peticiones (microservicios) y las rutas donde se espera tener peticiones a su entrada. Estas dos listas se unen junto con las políticas que se desean aplicar en cada una. Las rutas son las siguientes:

- **/api/login** $\longrightarrow$ **auth_api:8003**
- **/api/users*** $\longrightarrow$ **auth_api:8003**
  - Necesario *token*.
- **/api/data*** $\longrightarrow$ **influx_api:8002**
  - Necesario *token*.
- **/api/warnings*** $\longrightarrow$ **warnings_api:8000**
  - Necesario *token*.

5.3.6 Auth/User API

Este microservicio, al desarrollarse sobre Nodejs, con el Dockerfile se utiliza un contenedor oficial de Nodejs, añadiendo un script de entrada que espera a la conexión de la base de datos **Auth/User**.

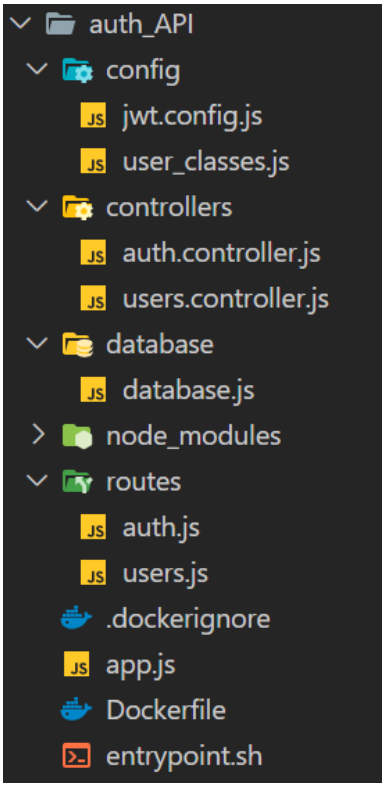

Figura 5.24: Directorio de desarrollo de *Auth/User API*

En la organización de la API, se ha seguido el esquema típico de una aplicación con el *middleware* Express. En el inicio de la aplicación *app.js* se registran las rutas de la *API*, que se redirigen a sus ficheros *auth.js* y *users.js*. A su vez, en cada ruta, según el método utilizado por la petición, esta se lleva a la función apropiada en el controlador, *auth.controller.js* y *users.controller.js*.

Por otra parte, la información de la base de datos se encuentra en *database.js*.

5.3.7 Logged Data API

El desarrollo de este microservicio ha sido idéntico al anterior, teniendo una única ruta y controlador para obtener las medidas de un nodo entre dos fechas. En el *script* inicial se espera a la conexión de la base de datos de series temporales.

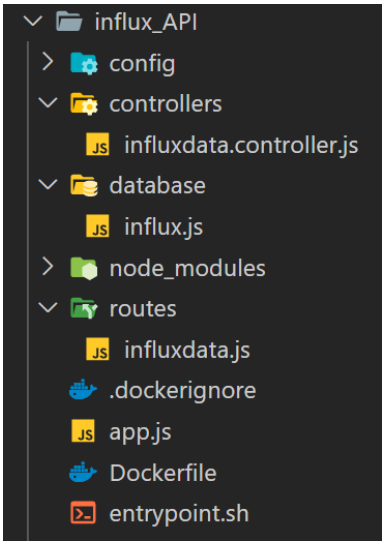

Figura 5.25: Directorio de desarrollo de *Logged Data API*

5.3.8 Time Series DB

Esta base de datos de series temporales con influxdb se monta directamente con la imagen oficial para contenedor Docker, a la cual se le añade un volumen con la configuración *influxdb.conf*. La configuración de la base de datos se trata de la que se incluye por defecto, donde se escucha al puerto 8086 sin autenticación ni encriptación. Este puerto sólo es accesible por contenedores dentro de la plataforma *IoT*.

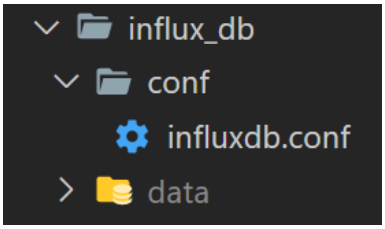

Figura 5.26: Directorio de desarrollo de *Time Series DB*

5.3.9 Warnings BOT & API

Este contenedor contiene el microservicio que proporciona la *API* para interactuar con los avisos y configurar el *BOT* de avisos por la aplicación de mensajería Te-

legram. Al ser desarrollado en Python, el fichero Dockerfile que configura el contenedor crea una imagen de Ubuntu, donde se instala Python3 y la dependencias utilizadas en el microservicio, además de incluir el *script* inicial que espera por las bases de datos de usuarios y avisos.

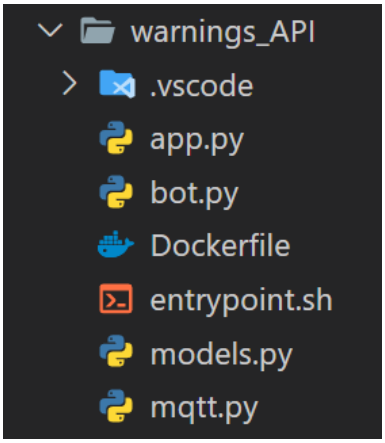

Figura 5.27: Directorio de desarrollo de *Warnings BOT & API*

La aplicación comienta en *app.py*, donde se importa e inicia el BOT (*bot.py*). Este *BOT* se suscribe en el *Broker MQTT* a los *topics* de los nodos *IoT* sobre los que se ha registrado avisos el usuario que utiliza la plataforma y, cuando uno de los valores recibidos a estos *topics* supera por el límite superior o inferior (según la configuración del aviso), se le envía por mensaje al usuario por la aplicación de mensajería Telegram.

Después de inicializar el *BOT*, se registran las rutas de las *APIs* con Flask, donde se permite crear y eliminar avisos. Cuando se registra uno nuevo, además de añadirlo en la base de datos, el *BOT* MQTT se suscribe a los nuevos *topics* del nodo *IoT* correspondiente.

Finalmente, en el fichero *models.py* se registran los modelos de las tablas de las bases de datos de avisos y usuarios para facilitar su uso.

5.3.10 Warnings DB

El desarrollo de esta base de datos es idéntico a la base de datos **Auth/Users**.

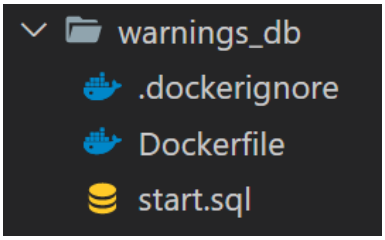

Figura 5.28: Directorio de desarrollo de *Warnings DB*

En este caso, las tablas creadas son las siguientes:

1. **warning:** Almacena los avisos creados con su identificación única, nombre de usuario de Telegram, nombre de usuario monitor, nombre del nodo *IoT*, *topic* para avisar, valor límite e indicación de límite superior o inferior.

Nombre	Tipo
id	Serial
tel_username	Varchar(10)
username	Varchar(10)
device_username	Varchar(20)
topic	Varchar(20)
data_value	Real
data_sign	Varchar(2)

Tabla 5.8: Tabla SQL warning

2. **tel_user:** Almacena la relación nombre de usuario de Telegram y su identificador de *chat*, que se registra la primera vez que se abre conversación con el *BOT*. Esto se debe a que un mensaje no puede ir dirigido directamente a un nombre de usuario, sino a su *chat_id*.

Nombre	Tipo
id	Serial
tel_chat_id	Varchar(10)
tel_username	Varchar(10)

Tabla 5.9: Tabla SQL tel_user

5.3.11 Metrics Collector

Finalmente, este componente de la plataforma *IoT* que recolecta todas las medidas enviadas por los nodos al *Broker* MQTT hasta la base de datos de Series Temporales, se crea mediante el Dockerfile con una imagen de Alpine y un *script* inicial que espera por la inicialización de la base de datos de Series Temporales.

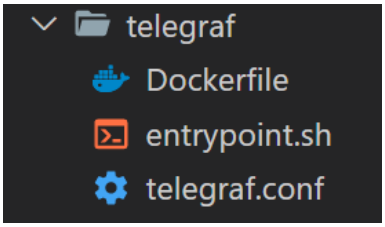

Figura 5.29: Directorio de desarrollo de *Metrics Collector*

La configuración de este software se realiza por medio del fichero *telegraf.conf*. Esta configuración es sencilla, se indica las entradas y salidas. Como entrada tenemos el *Broker* MQTT en la dirección **tcp://mqtt_broker_auth:2883** con los *topics* **data_log/#** y **alerts/#**. De esta manera, todos las mediciones de los nodos son registradas. Los tipos de datos se definen como *float* para el primer *topic* y *string* para el segundo. Por otra parte, la salida del recolector se indica como base de datos influxdb con dirección **http://influx_db:8086**.

5.4 Implantación

Tras el desarrollo de los sistemas, estos se llevan a la práctica para poder realizar las pruebas finales.

5.4.1 Sistema embebido de medición

A partir del circuito diseñado, en unas placas de pruebas se montan los *kits* de desarrollo, los encapsulados, los componentes pasivos, el *LCD* y el transductor de corriente, **Figura 5.30**.

En cuanto al puesto de prácticas. El motor asincrónico se conecta al variador de frecuencia M440 de Siemens, configurado para el control manual desde un potenciómetro externo. Por otra parte, el motor de imanes permanente se lleva al variador

ACS M1 con el que podemos controlar el par resistente. Además, se instala la pinza amperimétrica en una fase del motor de imanes permanentes, que, a su vez, se lleva a un multímetro para comparar con los valores obtenidos con el nuevo transductor de corriente.

Relacionado con la interacción del circuito a elementos externos:

- El encoder del motor asincrónico tiene una terminación de conector DB-9 hembra. Debido a esto, se realiza un cable de conector DB-9 macho con 3 conductores con la alimentación de 12v, la masa y la salida del tren de pulsos para llevarlos a la la etapa de acondicionamiento de señal de las placas de desarrollo.

- Una de las fases que va del motor de imanes permanentes al su variador de frecuencia se alarga para poder llevarlo a las placas de desarrollo y pasarlo por el transductor de corriente no invasivo.

- La salida del DAC del sistema embebido, tras el amplificador operacional, se lleva a un multímetro para comprobar la tensión de salida de realimentación.

- Para alimentar este circuito se utiliza una fuente externa que proporciona la tensión de 12V.

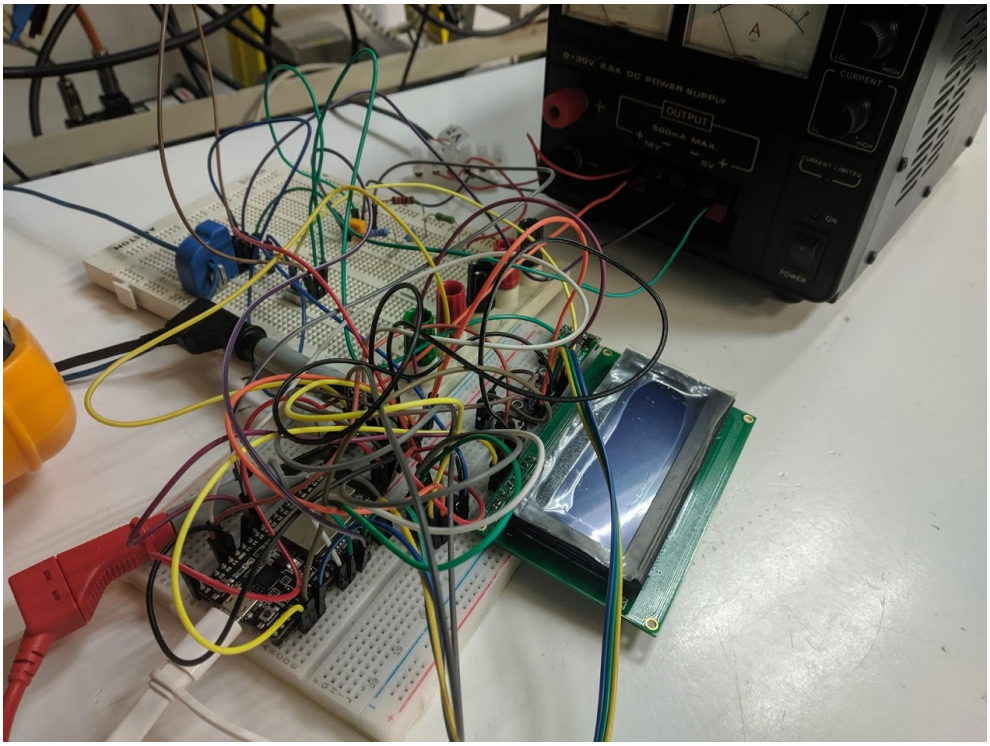

Figura 5.30: Circuito montado en una *Breadboard*

5.4.2 Aplicación móvil

Una vez desarrollada la aplicación, se realiza una compilación *AOT* para producción al sistema operativo Android, obteniendo un instalable *.apk*.

Esta aplicación tiene la ventaja de poder ejecutarse en dispositivos con los sistemas operativos iOS e Android con el mismo código fuente, pero las pruebas se han realizado con Android por no disponer de un dispositivo móvil con iOS ni un ordenador con MacOS que, junto con el entorno de desarrollo Xcode, permita compilar la aplicación.

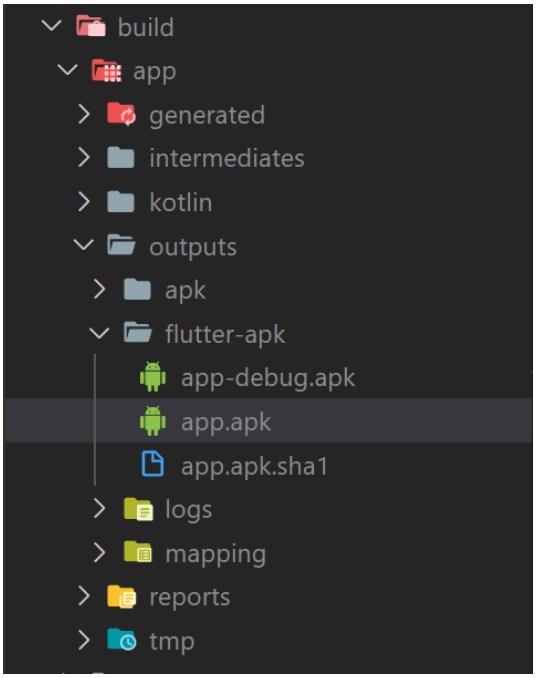

Figura 5.31: Aplicación exportada con el kit de desarrollo Flutter

5.4.3 Plataforma IoT

La ventaja del desarrollo con Docker es evitar la preparación de la máquina de ejecución con todo el software y dependencias. Para realizar las pruebas, en una máquina Linux x86-64 con el software Docker instalado, se levantan todos los contenedores orquestados mediante la herramienta Docker-compose.

CAPÍTULO

6

Pruebas

Tras implementar el diseño de los sistemas, se procede a las pruebas de cada uno para comprobar el cumplimiento de los objetivos.

6.1 Sistema embebido de medición

Para comprobar el correcto funcionamiento el sistema embebido se ha realizado 3 mediciones de velocidad y par controlando los variadores de frecuencia del motor asincrónico y de imanes permanentes.

6.1.1 Medida de velocidad

Para las pruebas de velocidad, la velocidad del motor se mide con un tacómetro hacia su eje, que tiene 4 pegatinas reflectantes, **Figura 6.1**. Por otra parte, la medida de velocidad realizada por el sistema embebido se comprueba en la pantalla *LCD*, **Figura 6.2**. Finalmente, la tensión realimentación que proporciona el sistema se observa en un multímetro, **Figura 6.3**.

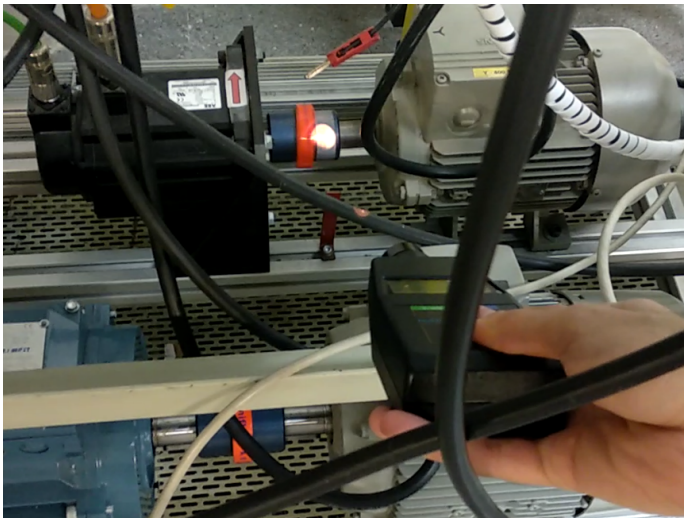

Figura 6.1: Medida de la velocidad con tacómetro

Figura 6.2: Velocidad medida por el sistema embebido mostrada en el *LCD*

Figura 6.3: Tensión de salida de realimentación

En la siguiente tabla se proporciona los resultados de las velocidades y tensiones (rango $0 - 10\,V$) con frecuencias de referencia que incluyen la mitad de la nominal y nominal del motor.

Velocidad LCD (rpm)	**Velocidad tacómetro** (rpm)	**Tensión de realimentación** (V)
351	$1413/4 = 353{,}2$	2,37
747	$2994/4 = 748{,}5$	5,03
1479	$5919/4 = 1479{,}75$	9,91

Tabla 6.2: Ensayos de velocidad

6.1.2 Medida de par

En las pruebas de medida de par se hace uso de la misma pantalla *LCD* para comprobar el valor medido por el sistema embebido, **Figura 6.4**. Como comparación se emplea la pinza amperimétrica conectada a un multímetro, **Figura 6.5**.

Figura 6.4: Par medido por el sistema embebido mostrado en el *LCD*

Figura 6.5: Tensión medida de la pinza amperimétrica

En la configuración de la pinza, $10\,A$ lo mide como $1\,V$ a su salida. Además, en el motor de imanes permanentes $15{,}5\,N \cdot m$ equivale a $14{,}4\,A$. Para obtener el

valor de par desde la tensión medida utilizamos la siguiente expresión:

$$T = 10 \cdot \frac{15{,}5}{14{,}4}\,, N \cdot m \tag{6.1}$$

En la siguiente tabla se recogen los valores de las 3 pruebas de par:

Par LCD $(N \cdot m)$	**Tensión de pinza** (V)	**Par calculado** $(N \cdot m)$
$1,55$	$0,145$	$1{,}56$
$7,17$	$0{,}65$	7
$15,69$	$1,431$	$15{,}4$

Tabla 6.4: Ensayos de par

6.2 Aplicación móvil

Para las pruebas de la aplicación móvil se han realizado las mismas 3 medidas de velocidad y par, las cuales se envían desde el sistema embebido a la aplicación por Bluetooth. De igual manera, para las medidas de velocidad, el variador de frecuencia del motor de imanes permanentes permanece apagado y no se fuerza par resitente, mientras que con el variador del motor asincrónico se controla la velocidad rotación, **Figura 6.6**.

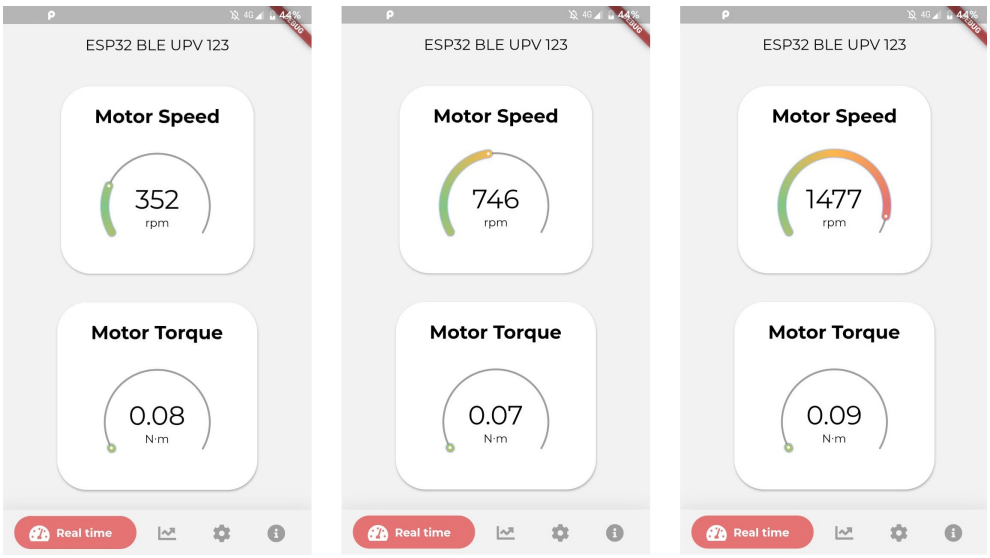

Figura 6.6: Medidas de velocidad en tiempo real

Por otra parte, en la prueba del par se deja una frecuencia fija en el variador del motor asincrónico y se modifica el par resistente del motor de imanes permanentes con su variador de frecuencia. Como el control del motor asincrónico es de bucle abierto, al aumentar el par resistente se pierde velocidad que no llega a recuperar, **Figura 6.7**.

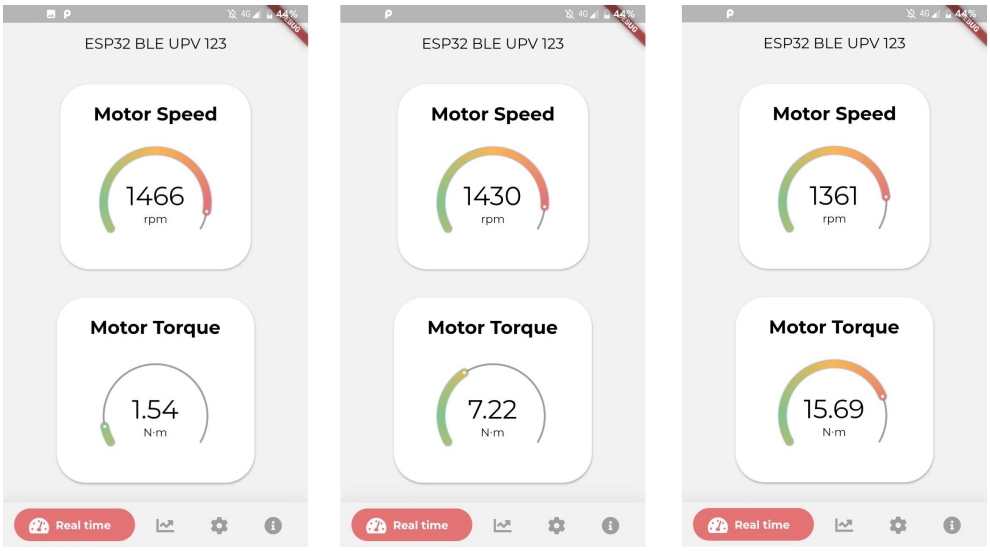

Figura 6.7: Medidas de par en tiempo real

Además, los 3 valores del ensayo de par se capturan en la pantalla de reporte, pudiendo visualizar el ensayo en las gráficas de la aplicación móvil, **Figura 6.8** y la hoja de cáclulo exportada, **Figura 6.9**.

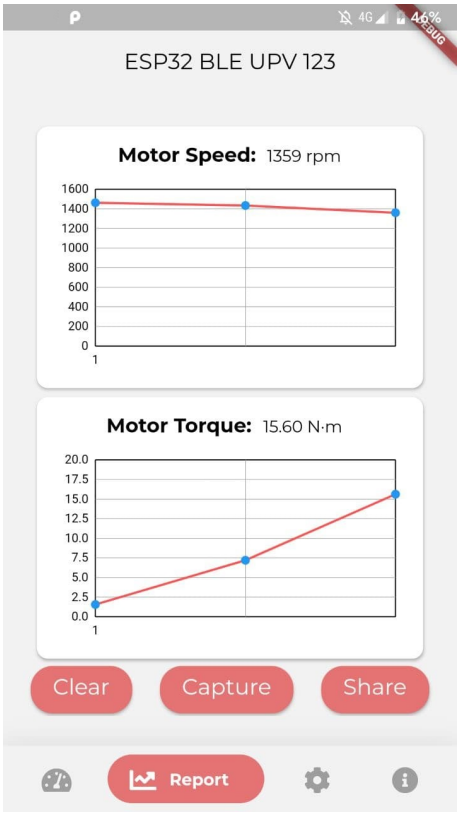

Figura 6.8: Capturas de las medidas

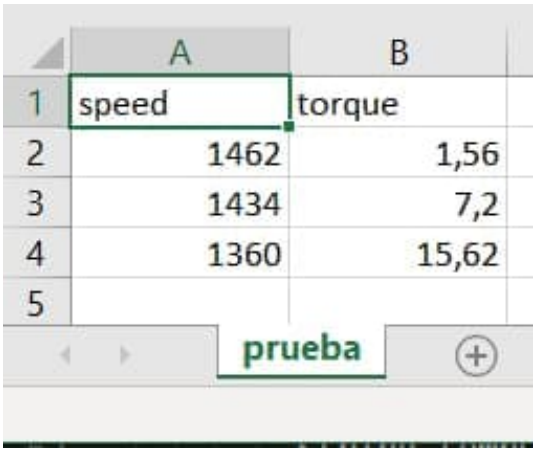

Figura 6.9: Hoja de cálculo con las capturas

Finalmente, para comprobar los valores enviados desde la aplicación móvil hacia el sistema embebido, en la **Figura 6.10** se muestra en la primera fila 3 valores

de brillo de la pantalla, mientras que en la segunda fila se encuentran 3 valores de contraste diferente.

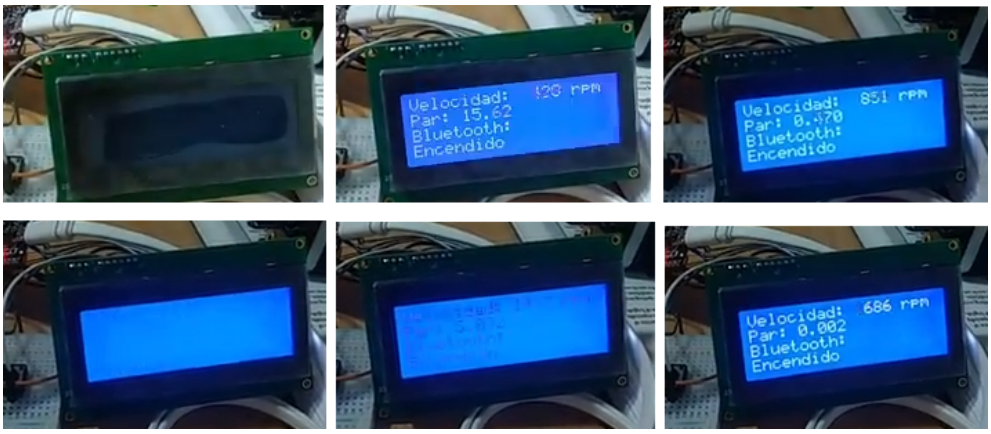

Figura 6.10: Configuración de brillo y contraste desde la aplicación móvil

6.3 Plataforma IoT

Con el objetivo de probar la plataforma *IoT*, se ha utilizado la herramienta **MQTT Explorer** para simular un nodo *IoT* y de esta manera publicar en *topics* determinados con unas credenciales para la autenticación. En los siguientes apartados se probarán las diferentes funcionalidades que se le permite a un usuario monitor desde la aplicación Web.

6.3.1 Autenticación

En la pantalla de *login*, la única acción posible tras la creación de la plataforma es entrar con la cuenta de administrador. En la siguiente imagen se muestra el error que sucede cuando se introduce unas credenciales incorrectas.

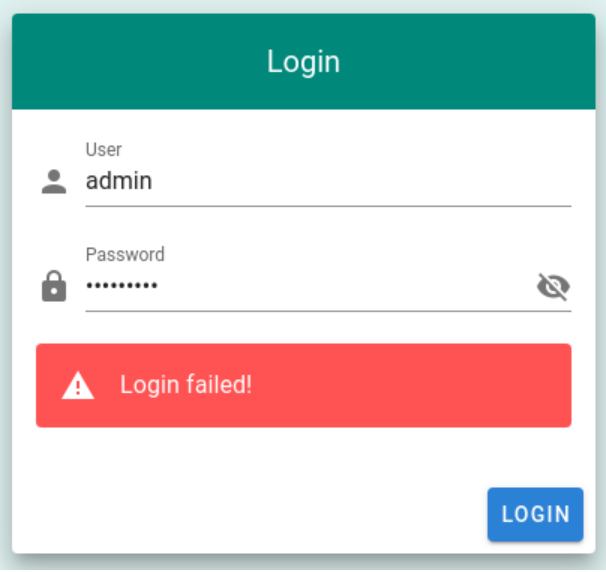

Figura 6.11: Mensaje de error en la autenticación de la plataforma *IoT*

6.3.2 Gestión de usuarios del administrador

Una vez dentro de la pantalla principal (*Home*), en la vista de usuarios tenemos la lista de todos los usuarios registrados junto con su clase. Además, como administrador se nos proporciona la posibilidad de crear nuevos usuarios.

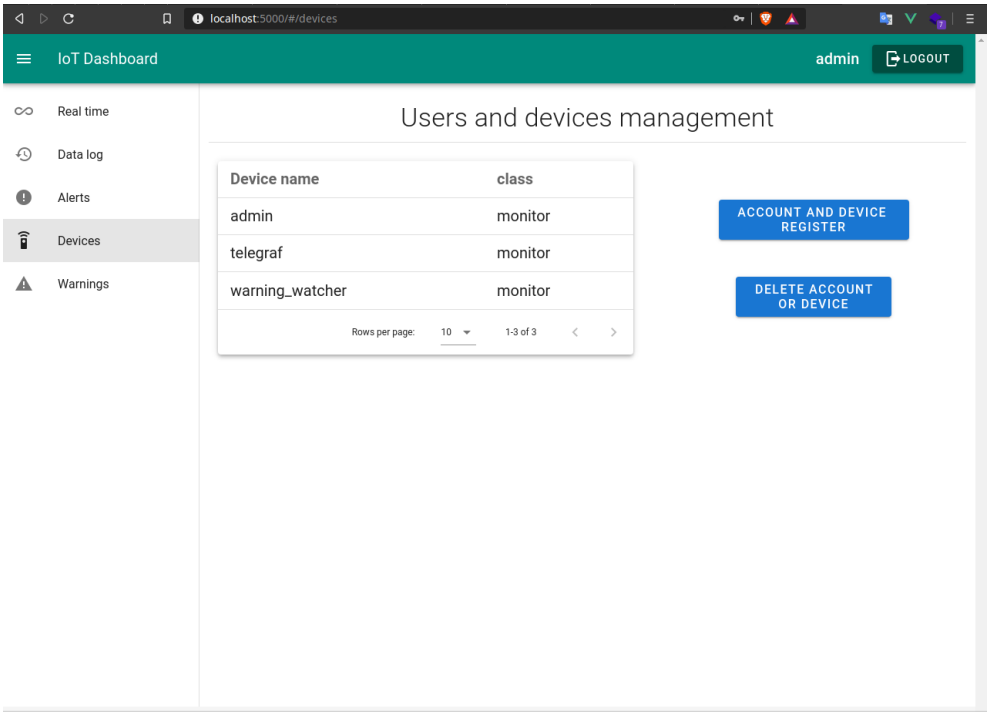

Figura 6.12: Pantalla de usuarios de la plataforma *IoT*

Con esta opción se abre un formulario con el nombre de usuario, contraseña, tipo de usuario y, en caso de ser de tipo *monitor*, otra opción donde se listan todos los usuarios que se han registrado con un tipo que no sea *monitor* para darle permisos de monitorización sobre los mismos.

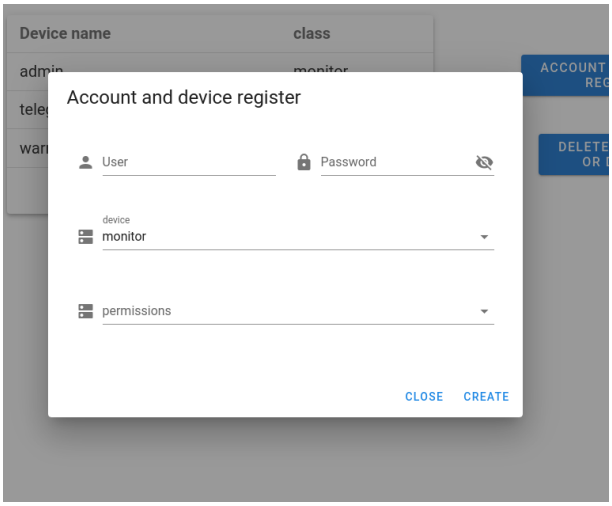

Figura 6.13: Dialogo de creación de nuevo usuario

Además, en la opción de eliminación de usuarios, al tener solamente 3 usuarios con atributo de administrador, la lista aparece vacía.

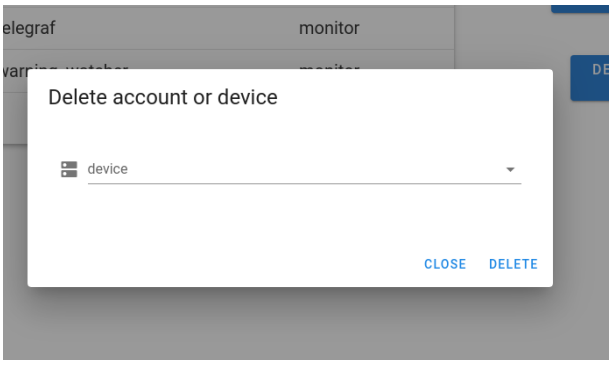

Figura 6.14: Dialogo de eliminación de un usuario

En este momento se crean los siguientes usuarios:

- **nodo1:** Usuario de tipo nodo *IoT AC Motor*.
- **nodo2:** Usuario de tipo nodo *IoT AC Motor*.
- **usuario1:** Usuario de tipo monitor con permisos sobre **nodo1**
- **usuario2:** Usuario de tipo monitor con permisos+ sobre **nodo2** y **nodo3**

- **usuarioborrar:** Usuario para comprobar la eliminación de usuarios.

En esta captura se muestra como, si no se trata de un usuario de tipo monitor, desaparece la opción de añadir permisos sobre otros usuarios:

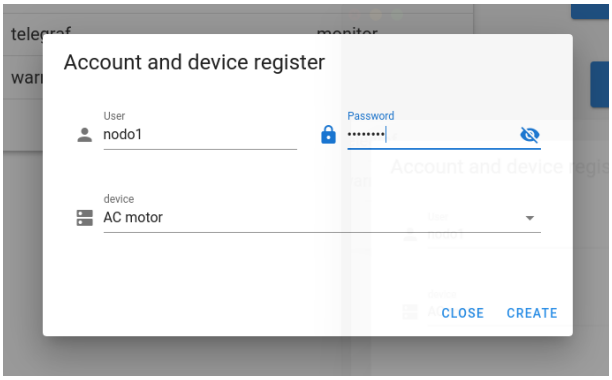

Figura 6.15: Registro del usuario *nodo1*

En las siguientes dos capturas se realiza la creación de los 2 usuarios de monitorización:

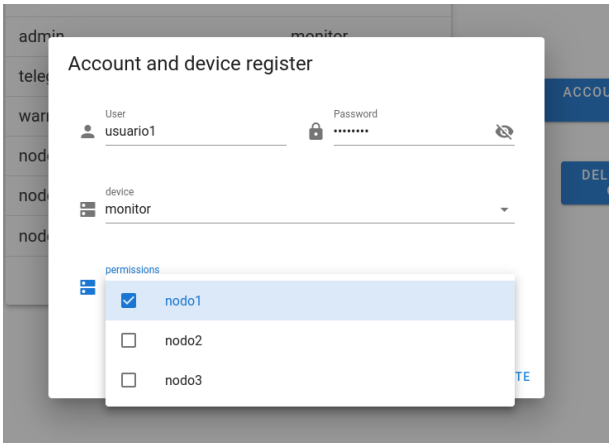

Figura 6.16: Registro del usuario *usuario1*

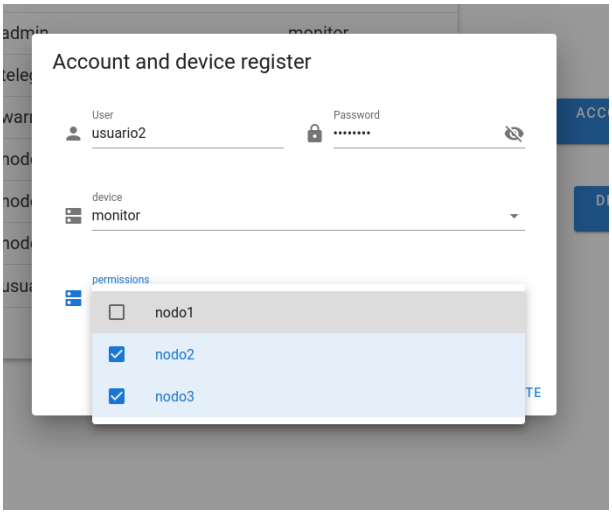

Figura 6.17: Registro del usuario *usuario2*

Tras el proceso de registro, el administrador puede observar la siguiente lista de usuarios:

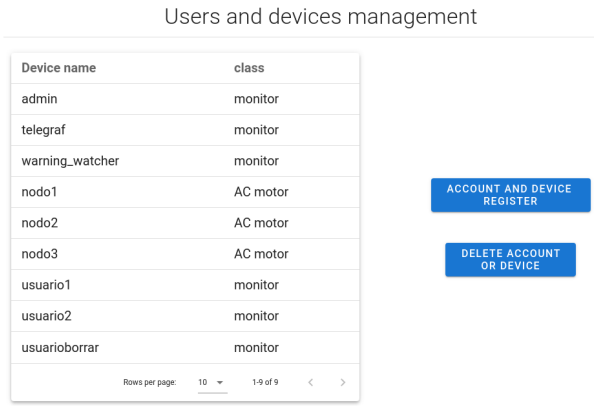

Figura 6.18: Lista de usuarios tras la creación de estos

A partir del botón de borrado, seleccionamos el usuario de prueba para su eliminación:

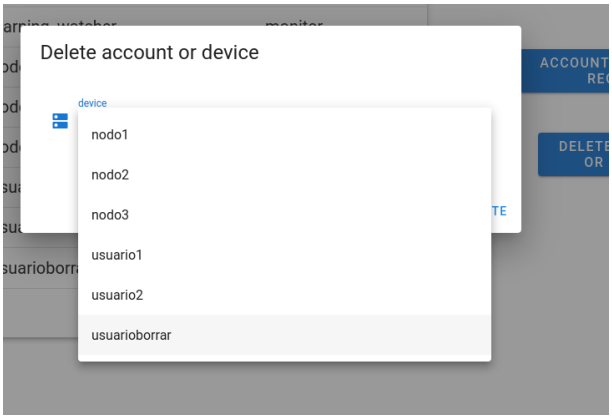

Figura 6.19: Eliminación del usuario de prueba

Finalmente, el administrador de la plataforma observa los siguientes usuarios:

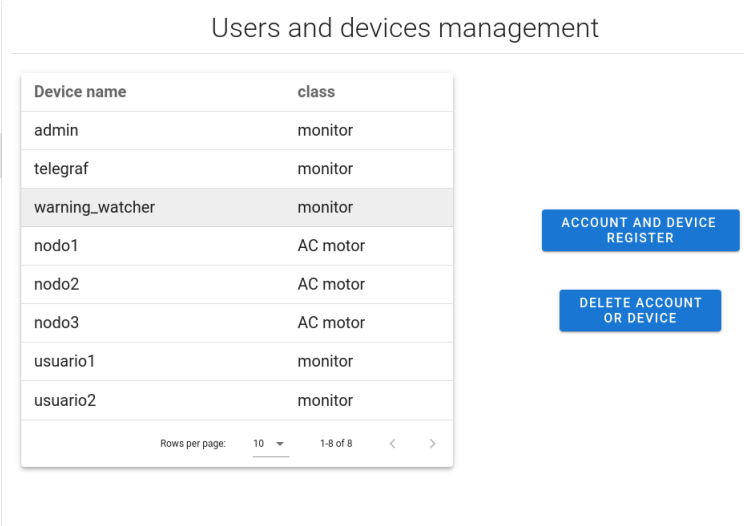

Figura 6.20: Lista final de usuarios para las pruebas de la plataforma

Por otra parte, al entrar en la plataforma con los usuarios registrados *usuario1* y *usuario2*, sólo se ven los usuarios sobre los que tienen permisos, sin los usuarios administrador y sin capacidad de crear y eliminar otros usuarios:

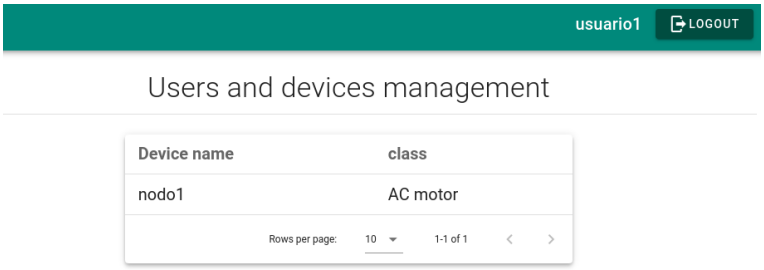

Figura 6.21: Lista de usuarios mostrado por el *usuario1*

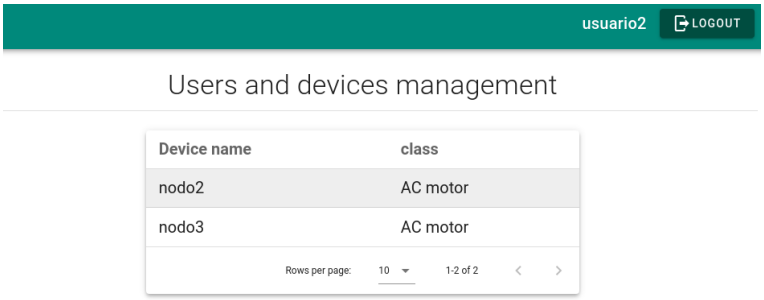

Figura 6.22: Lista de usuarios mostrado por el *usuario2*

6.3.3 Ver medidas de un nodo *IoT* en tiempo real

Para comprobar las mediadas en tiempo real se entra en la plataforma con *usuario1* y en MQTT Explorer con *nodo1*. En el caso de este usuario, sólo tenemos la opción de ver las medidas de *nodo1*.

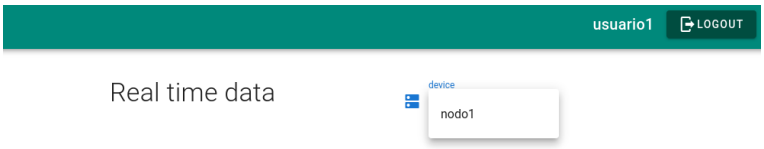

Figura 6.23: Selección de *nodo1* para la monitorización en tiempo real

Al seleccionar este nodo, según su tipo (AC Motor en este caso), se dibujan los gráficos con la información correspondiente:

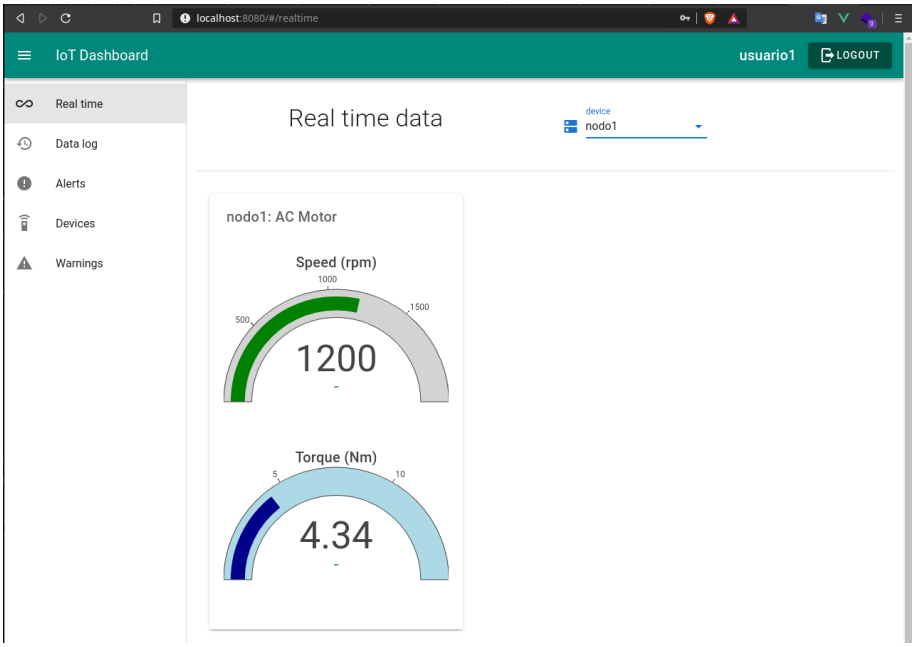

Figura 6.24: Medidas en tiempo real enviadas por *nodo1* a través de MQTT

En la siguiente captura se comprueba como a *nodo1* no se le permite la entrada al *Broker MQTT* con una contraseña incrrecta.

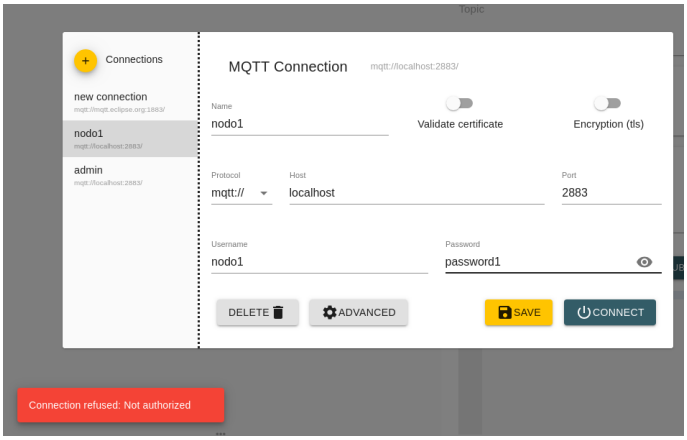

Figura 6.25: Conexión fallida de *nodo1* con una contraseña incorrecta

Otro escenario de error es la publicación de medidas en un *topic* de MQTT en el que el nodo no tiene permisos, en este caso, al intentar *nodo1* publicar en un *topic* de *nodo2* se le deniega la acción por parte del *Broker MQTT*.

```
mqtt_broker_auth_1  | 1606089518: |-- SUPERUSER: nodo1
mqtt_broker_auth_1  | 1606089518: |-- user is 0
mqtt_broker_auth_1  | 1606089518: |-- USERNAME: nodo1, TOPIC: data_log/nodo2/speed, acc: 2
mqtt_broker_auth_1  | 1606089518: |--   postgres: topic_matches(data_log/nodo1/#, data_log/nodo1/#) == 0
mqtt_broker_auth_1  | 1606089518: |--   postgres: topic_matches(alerts/nodo1/#, alerts/nodo1/#) == 0
mqtt_broker_auth_1  | 1606089518: |-- aclcheck(nodo1, data_log/nodo2/speed, 2) AUTHORIZED=0 by none
mqtt_broker_auth_1  | 1606089518: |--  Cached  [1C6D5C14B82FC52AAABC9D43E989821080181 6D7] for (mqtt-explorer-c05a14c0,nodo1,2)
mqtt_broker_auth_1  | 1606089518: Denied PUBLISH from mqtt-explorer-c05a14c0 (d0, q0, r0, m0, 'data_log/nodo2/speed', ... (4 bytes))
```

Figura 6.26: Mensaje de error del *Broker MQTT* al publicar en un *topic* sin permisos

Finalmente, en el caso de publicar en el *topic* correcto, el *Broker MQTT* acepta el envío.

```
mqtt_broker_auth_1  | 1606089562: |-- mosquitto_auth_acl_check(..., mqtt-explorer-c05a14c0, nodo1, data_log/nodo1/speed, 2)
mqtt_broker_auth_1  | 1606089562: |-- aclcheck(nodo1, data_log/nodo1/speed, 2) CACHEDAUTH: 0
mqtt_broker_auth_1  | 1606089562: Received PUBLISH from mqtt-explorer-c05a14c0 (d0, q0, r0, m0, 'data_log/nodo1/speed', ... (4 bytes))
```

Figura 6.27: Envío correcto de medida de *nodo1*

6.3.4 Obtener gráficas del histórico de un nodo *IoT*

En la comprobación de la obtención del histórico de medidas de un nodo, como usuario se utiliza *usuario1* y, de la misma manera que en la prueba anterior, se publican diversos mensajes en los *topic* de *speed* y *torque* con *nodo1*, que simula un motor.

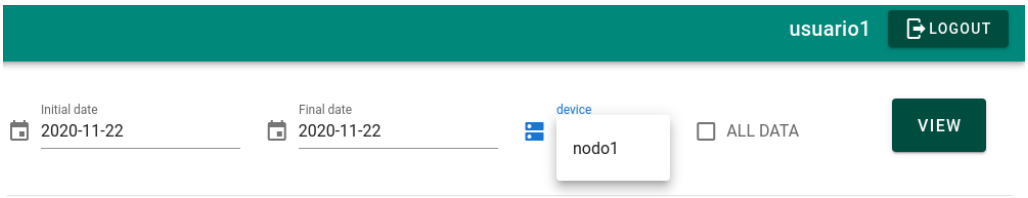

Figura 6.28: Selección de *nodo1*

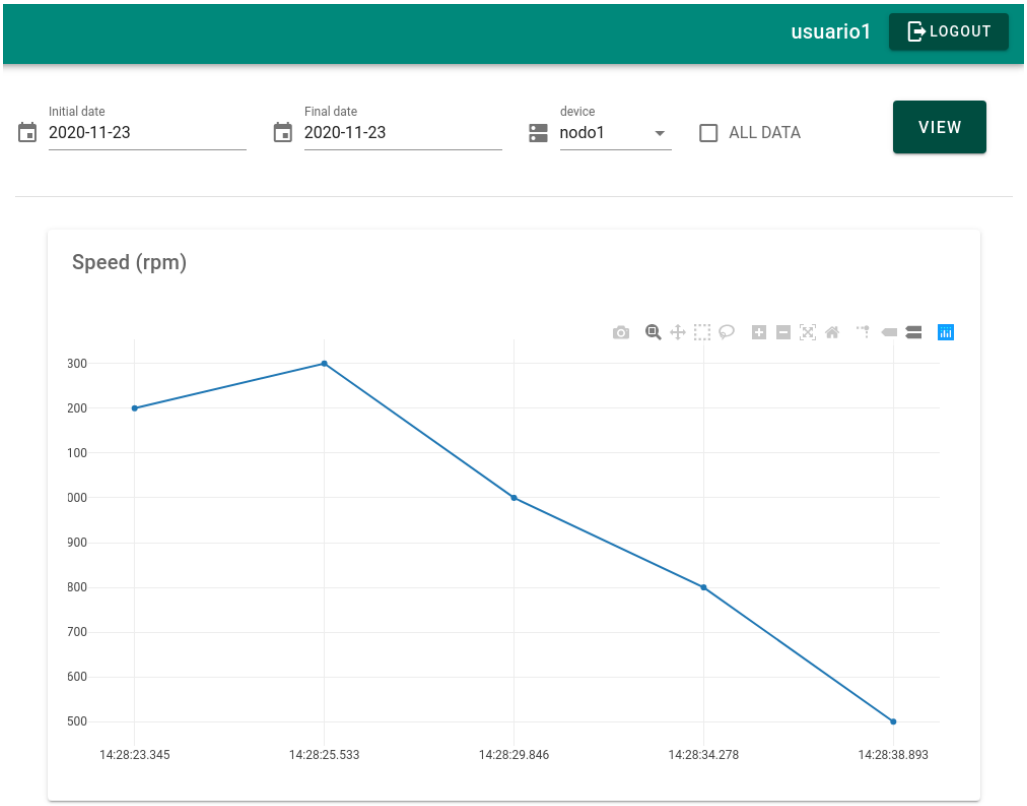

Figura 6.29: Histórico de medidas de velocidad de *nodo1*

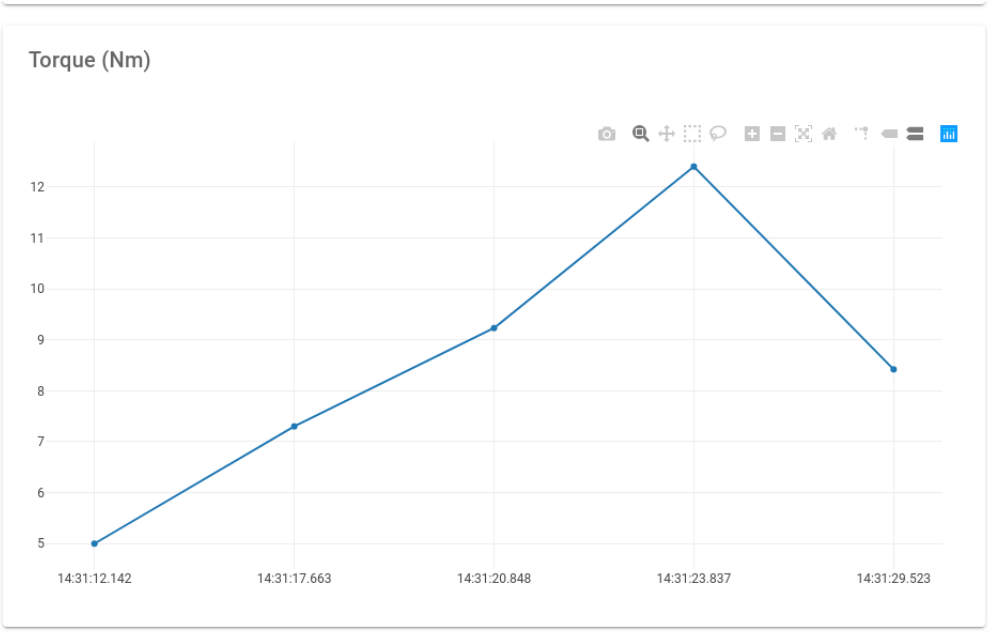

Figura 6.30: Histórico de medidas de par de *nodo1*

Se puede observar que, para la elección de las fechas, se despliega un calendario para facilitar al usuario la selección. Otra opción es señalar la casilla de obtener todos los datos.

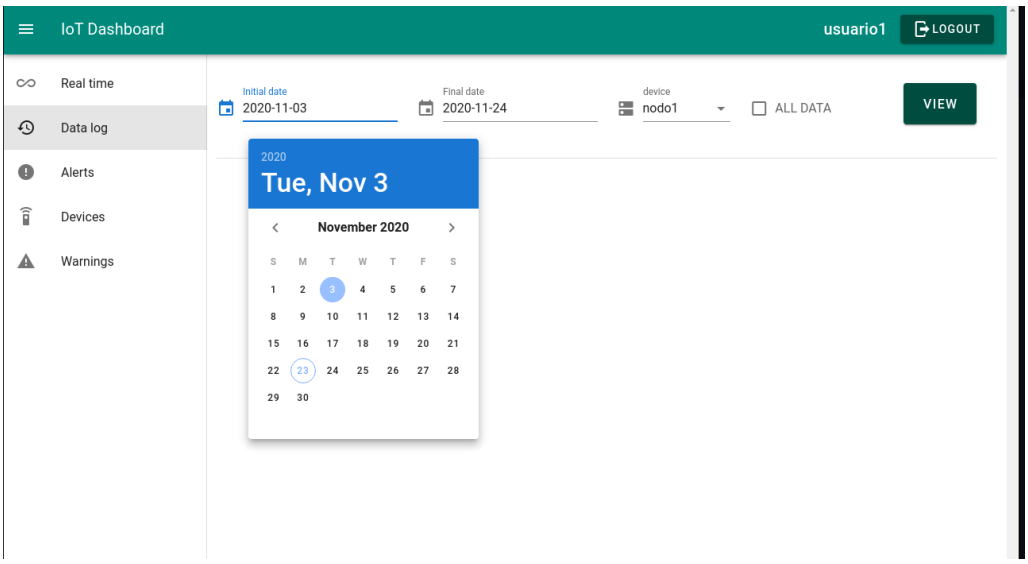

Figura 6.31: Desplegable de selección de fecha

6.3.5 Visualización de las alertas de los nodos *IoT*

De manera similar a la prueba anterior, con *nodo1* y *nodo2* se envían unos mensajes de alerta. En el caso de *usuario1*, se pueden ver las alertas de *nodo1*, mientras que *usuario2* tiene permisos sobre *nodo1* y *nodo2*.

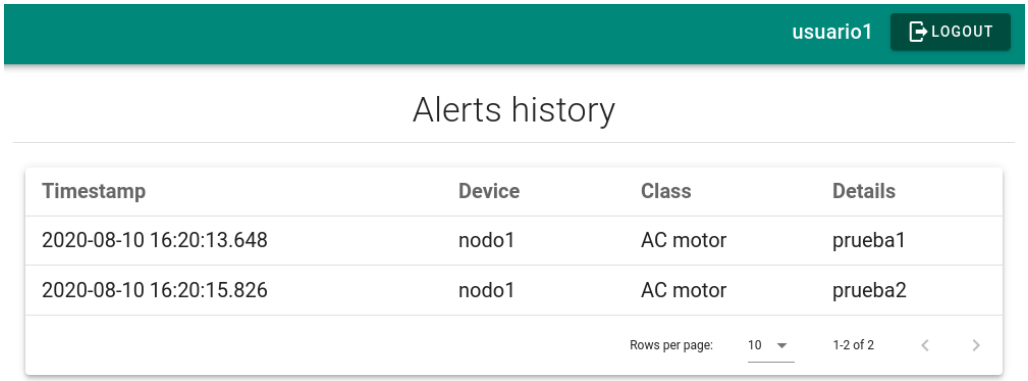

Figura 6.32: Historial de alertas de los nodos de *usuario1*

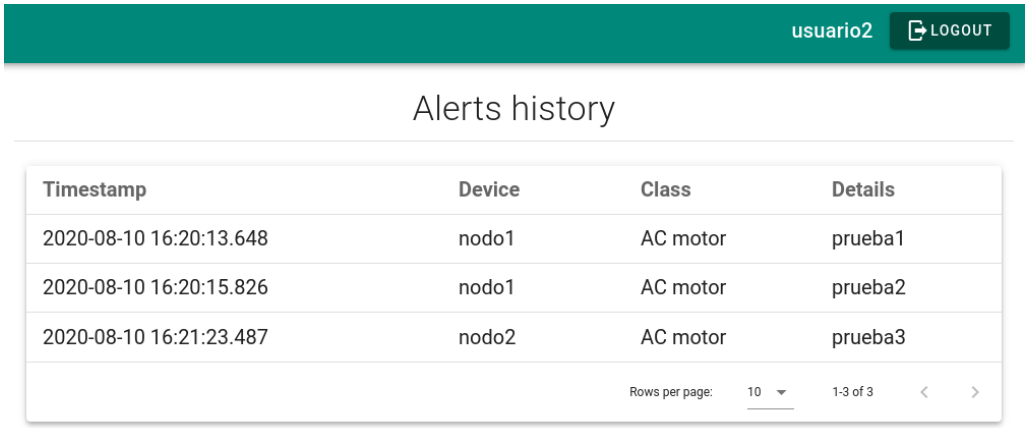

Figura 6.33: Historial de alertas de los nodos de *usuario2*

6.3.6 Gestión de avisos de un nodo *IoT*

Como última funcionalidad de la plataforma tenemos la gestión de avisos personalizados. La prueba se realiza con *usuario2* y *nodo2*.

El primer paso es registrar un aviso indicando el usuario de la aplicación de mensajería Telegram que lo debe recibir. En este caso se crea un aviso en el campo de velocidad, cuando un valor supere los 2300.

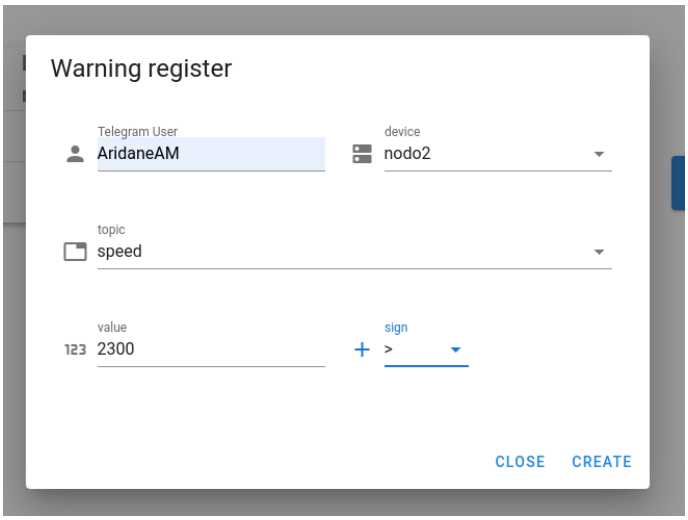

Figura 6.34: Registro de nuevo aviso sobre *nodo2*

Una vez creado el aviso, en la lista se muestran todos los que se han creado hasta el momento.

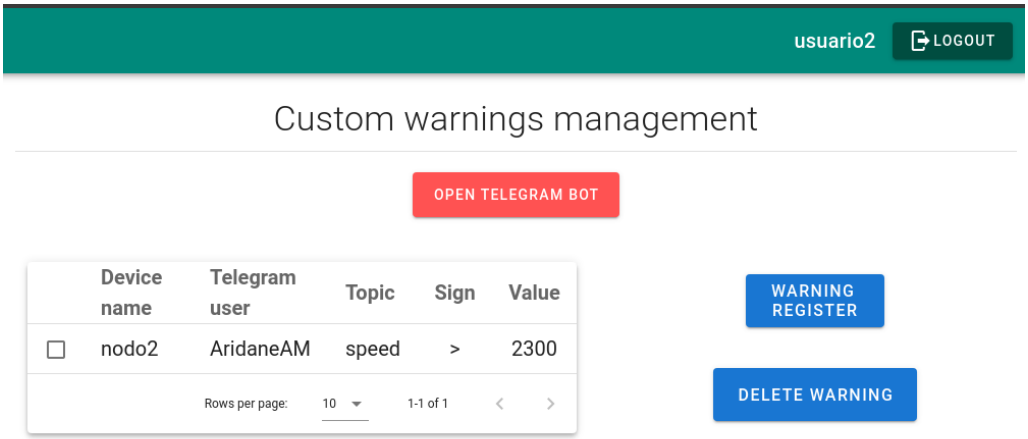

Figura 6.35: Lista de avisos creados por *usuario1*

Como muestra de la recepción de mensajes por parte del microservicio de avisos, en su log se observa la llegada de un valor que no supera el límite y, por lo tanto, no es avisado.

```
warnings_api_1      | topic:  speed
warnings_api_1      | username:  nodo2
warnings_api_1      | payload:  2299
```

Figura 6.36: Valor recibido en el campo del aviso que no pasa el límite

Por otra parte, en la siguiente captura se comprueba que, cuando el valor generado por *nodo2* supera el límite, el microservicio crea el mensaje y lo envía con el *BOT*.

```
warnings_api_1      | topic:  speed
warnings_api_1      | username:  nodo2
warnings_api_1      | payload:  2400
warnings_api_1      | message:  topic speed of nodo2: 2400
```

Figura 6.37: Detección de valor que cumple las condiciones del aviso *nodo1*

Finalmente, en la siguiente captura se observa la llegada del aviso al usuario Telegram que se ha registrado:

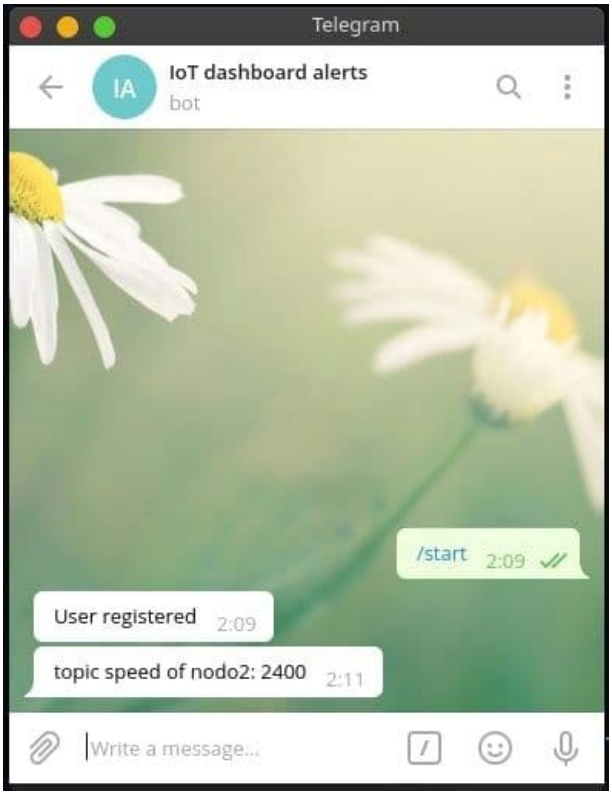

Figura 6.38: Recepción de aviso generado por *nodo1*

CAPÍTULO

7

Conclusiones

El propósito de este trabajo era el diseño de un sistema embebido para la monitorización de un motor asincrónico junto con una aplicación móvil que facilite la toma de medidas. Además, como software complementario, se propuso la creación de una plataforma para el internet de las cosas.

A partir de estos objetivos, para delimitar el alcance del trabajo se realizaron unas especificaciones de requisitos, lo cual ha evitado entrar en un bucle infinito de continuas mejoras y ampliaciones de los sistemas, sobre todo software.

Para cubrir los requisitos obtenidos, el siguiente paso ha sido diseñar el circuito electrónico del sistema embebido y y la arquitectura de los sistemas de software. Este paso previo ha facilitado el desarrollo al asentar unas bases que dejan menos libertad a la improvisación.

En cuanto a las tecnologías utilizadas en el trabajo, gracias a conocer la plataforma de desarrollo que utiliza el microcontrolador ESP32, no se han obtenido grandes problemas durante el desarrollo. Por otra parte, uno de los objetivos personales conseguidos era aprender a realizar aplicaciones móvil con el *framework* Flutter, ya que es un kit de desarrollo con menos de 2 años desde su primera versión estable que está en auge y al que grandes compañías han transferido su aplicación nativa. Además, se trata del kit de desarrollo del próximo sistema operativo multiplataforma Google Fuchsia. Finalmente, con la plataforma *IoT*, también se

ha logrado aplicar el conocimiento previo en desarrollo de aplicaciones web a un entorno industrial con el internet de las cosas.

En cuanto a la arquitectura de la pataforma *IoT*, se ha podido observar las ventajas de un diseño de microservicios, donde se permite futuras ampliaciones de la plataforma y mejoras de los actuales microservicios sin afectar en el desarrollo unos a otros. Como ejemplo, un tipo de microservicio posible a ampliar sería un análisis de las medidas con el objetivo de entrenar una red neuronal para la predicción de fallas.

Relacionado con ampliaciones del trabajo, ya comprobado su correcto funcionamiento, el siguiente paso a realizar será el diseño de una *PCB* para la integración final en el panel del laboratorio. Por otra parte, del trabajo ha surgido un nuevo objetivo personal, que consiste en mejorar la plataforma *IoT* con el conocimiento adquirido a medida que se realizaba el trabajo y ampliarla con nuevos tipos de dispositivos, para finalmente7,03 publicarla en un repositorio de código libre.

CAPÍTULO

8

Presupuesto

En este capítulo se muestra el presupuesto para realizar el prototipo, donde se incluyen las horas de desarrollo y montaje del diseño junto con el coste de los materiales utilizados.

En la siguiente tabla se lista el coste total de los materiales que forman el prototipo del sistema embebido:

Componente	Coste unitario (€)	Cantidad	Coste total (€)
Encoder 1XP8001-1/1024	487,60	1	487,60
Transductor LTSR 6-NP	13,24	1	13,24
LCD NHD-0420D3Z-NSW-BBW	20,62	1	20,62
ADC ADS1015	6,95	1	6,95
DAC MCP4922	2,22	1	2,22
Amplificador operacional TLC272	1,00	1	1,00
Microcontrolador ESP32 devkitc v4	6,98	1	6,98
Convertidor DC-DC MP1584EN	1,61	1	1,61
Kit Cables dupont	4,95	2	9,90
Placa de pruebas	5,81	2	11,62
Total (sin IVA)			**561,74**

Tabla 8.1: Coste material

Por otra parte, en esta tabla se indica las horas dedicadas al proyecto junto con su coste:

Servicio	Coste por hora (€)	horas	Coste total (€)
Diseño del circuito del sistema embebido	20,00	50	1000,00
Diseño y desarrollo de software embebido	20,00	100	2000,00
Diseño y desarrollo de aplicación móvil	20,00	80	1600,00
Diseño y desarrollo de plataforma IoT	20,00	100	2000,00
Montaje de circuito y pruebas	15,00	15	225,00
Total (sin IVA)			**6825,00**

Tabla 8.2: Servicios prestados

Finalmente, para realizar el prototipo del sistema embebido se han empleado las siguientes herramientas:

Herramienta	Coste (€)	Amortización (años)	Coste/h (€)	horas	Total (€)
Osciloscopio *Digilent AD* 2	220,91	10	0,0025	165	0,41
Multímetro IDM62T 2	87,66	10	0,00096	165	0,16
Fuente de alimentación EP-603	84,28	10	0,00096	165	0,15
Ordenador portátil HP Pavilion	661,15	8	0,0094	345	3,25
Total (sin IVA)					**3,97**

Tabla 8.3: Herramientas utilizadas

El presupuesto de ejecución material del prototipo, teniendo en cuenta los costes materiales, los servicios y el uso de las herramientas es de:

Concepto	Coste (€)
Materiales	561,74
Servicios	6825,00
Herramientas	3,97
Total	**7390,71**

Tabla 8.4: Presupuesto de ejecución material

Añadiendo los gastos generales y el beneficio industrial, obtenemos el siguiente presupuesto total:

Concepto	Coste (€)
PEM	7390,71
Gastos generales (13 %)	960,79
Beneficio industrial (6 %)	443.44
Total	**8794,94**

Tabla 8.5: Presupuesto total de realización del prototipo

A continuación, se proporciona el coste del sistema embebido como producto final en PCB para diferentes cantidades:

Componente	Coste 1u (€)	Coste 100u (€)	Coste 1000u (€)
Encoder 1XP8001-1/1024	487,60	48760,00	487600,00
Transductor LTSR 6-NP	13,24	1025,62	10256,19
LCD NHD-0420D3Z-NSW-BBW	20,62	1629,00	15340,00
ADC ADS1015	6,95	695,00	6950,00
DAC MCP4922	2,22	207,81	2078,10
Amplificador operacional TLC272	1,00	88,18	881,81
Microcontrolador ESP32 devkitc v4	6,98	698,00	6980,00
Convertidor DC-DC MP1584EN	1,61	161,00	1610,00
Conectores *header* PCB	1,66	129,08	922,03
Conectores montaje para PCB	4,70	417,00	3330,00
PCB (2 capas 120mm x 100mm)	10,00	160,00	925,00
Coste total (sin IVA)	**556,58**	**53970,61**	**536873,13**
Coste por unidad (sin IVA)	**556,58**	**539,70**	**536,87**

Tabla 8.6: Coste de los materiales del sistema embebido en diferentes cantidades

Finalmente, incluyendo el desarrollo y las herramientas utilizadas duramente el mismo, el precio final es de:

Concepto	Coste 1u (€)	Coste 100u (€)	Coste 1000u (€)
Materiales	556,58	53970,61	536873,13
Servicios	6825,00	6825,00	6825,00
Herramientas	3,97	3,97	3,97
Coste total (sin IVA)	**7385,55**	**60799,58**	**543702,10**
Coste por unidad (sin IVA)	**7385,55**	**607,99**	**543,70**

Tabla 8.7: Presupuesto de ejecución material para diferentes cantidades

Añadiendo los gastos generales y el beneficio industrial, obtenemos el siguiente presupuesto total:

Concepto	Coste 1u (€)	Coste 100u (€)	Coste 1000u (€)
PEM	7385,55	60799,58	543702,10
Gastos generales (13 %)	960,12	7903,94	70681,27
Beneficio industrial (6 %)	443.13	3647,97	32622,13
Coste total (sin IVA)	**8789,47**	**72351,49**	**647005,5**
Coste por unidad (sin IVA)	**8789,47**	**723,51**	**647,01**

Tabla 8.8: Presupuesto total de realización del producto para diferentes cantidades

Bibliografía

[1] Siemens, *1XP8001-1 User Manual*, 01 2005. https://www.tme.eu/Document/6ec579aeb5c80dedadc469158bd4cd2c/1XP8001-1.pdf. 7, 34

[2] LEM, *LTSR 6-NP User Manual*, 09 2017. https://www.lem.com/sites/default/files/products_datasheets/ltsr_6-np.pdf. 8, 38

[3] Newhaven Display, 2661 Galvin Ct. Elgin IL, 60124, *NHD-0420D3Z-NSW-BBW-V3 User Manual*. https://www.newhavendisplay.com/specs/NHD-0420D3Z-NSW-BBW-V3.pdf. 9, 39, 88

[4] Siemens, Postfach 3269, D-91050 Erlangen, Federal Republic of Germany, *MICROMASTER 440 User Manual*, 08/2013 ed., 08 2013. https://cache.industry.siemens.com/dl/files/204/23708204/att_60463/v1/440_PLi_23708204_en_0106.pdf. 10, 32

[5] "Iso/iec/ieee international standard - systems and software engineering – life cycle processes – requirements engineering," *ISO/IEC/IEEE 29148:2018(E)*, pp. 1–104, 2018. 11

[6] Espressif, *ESP32 Datasheet*, 04 2020. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. 30, 35, 42

[7] Microchip, *MCP4922 Datasheet*, 5 2010. http://ww1.microchip.com/downloads/en/DeviceDoc/22250A.pdf. 32, 33, 42, 87

[8] Texas Instruments, *TLC272 OP-AMP Datasheet*, 2 2002. https://www.ti.com/lit/ds/symlink/tlc272.pdf. 34

[9] Texas instruments, *ADS1015 Datasheet*, 1 2018. https://www.ti.com/lit/ds/symlink/ads1015.pdf. 38, 42, 86

[10] Monolithic Power, *MP1584 Datasheet*, 8 2011. https://www.monolithicpower.com/en/documentview/productdocument/index/version/2/document_type/Datasheet/lang/en/sku/MP1584/document_id/204/. 41

[11] J. S. F. J. A. Cook, "Embedded software architecture," 2008. 43

[12] C. Kok-Soon, "Uml and function-class decomposition for embedded software design," 2006. 43

[13] "About *FreeRTOS*." https://www.freertos.org/about-RTOS.html. Accedido: 2020-07-17. 68

[14] C. A. Townsend, K.; Cufi, *Getting Started with Bluetooth Low Energy*. O'Reilly Media, 2014. 68

[15] G. C. Hillar, *MQTT Essentials - A Lightweight IoT Protocol*. Packt Publishing Ltd., 2017. 71

[16] "Arquitectura de flutter." https://flutter.dev/docs/resources/architectural-overview. Accedido: 2020-07-20. 78

[17] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 ed., 1994. 88