

Trabajo de Fin de Grado

Dispositivo IoT a través del protocolo MQTT

IoT device through the MQTT protocol

Jesús Carmelo González Domínguez

La Laguna, 14 de julio de 2023

D. **Jonay Tomás Toledo Carrillo**, con DNI 78698554-Y profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Dispositivo IoT a través del protocolo MQTT”

ha sido realizada bajo su dirección por **Jesús Carmelo González Domínguez**, con DNI 43379857-V.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

Agradecimientos

En primer lugar, agradecer el incondicional apoyo que me ha prestado mi familia a lo largo de todos estos años, especialmente en la recta final.

Agradecer también a mi pareja por brindarme las palabras necesarias que lograron que sacaran fuerzas cuando me encontraba perdido.

Finalmente, agradecer a mi tutor de Trabajo Final de Grado Jonay Tomás Toledo Carrillo, ya que, sin su aprobación, esta propuesta de proyecto no habría salido adelante. Además, sus numerosos consejos y experiencia fueron de vital importancia para que este trabajo cumpliera las expectativas propuestas.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-
NoComercial-CompartirIgual 4.0 Internacional.

Resumen

Teniendo en cuenta la gran importancia que los dispositivos IoT están recibiendo en nuestro día a día, la finalidad de este trabajo ha sido explorar la viabilidad y el desarrollo de un dispositivo IoT en todos sus ámbitos, desde la programación del SoC ESP32, hasta la parte del servidor, la cual se encarga de publicar todo el stack de aplicaciones necesarias para recoger, interpretar y mostrar la información recogida por dicho ESP32.

El dispositivo que se va a mostrar a lo largo de todo este trabajo es un termostato, el cual posee una interfaz gráfica local que permite al usuario interactuar con él, gracias al panel táctil integrado en la placa de desarrollo. También, este dispositivo es capaz de trabajar en modo sin conexión, gestionar redes WiFi y conectarse a un servidor, con el fin de visualizar todos los datos recogidos por sus cuatro sensores, reportar alertas, etc.

Por la parte del servidor, todo el software ha sido desplegado en contenedores Docker, formando el siguiente stack de aplicaciones: Traefik, un reverse proxy que gestiona las comunicaciones entrantes al servidor y aporta los certificados SSL necesarios; Portainer, un gestor de contenedores de Docker; Mosquitto, un bróker para las comunicaciones del protocolo MQTT; InfluxDB, una base de datos gestionada por marcas de tiempo; NodeRED, una herramienta gráfica que se sitúa entre Mosquitto e InfluxDB y que permite crear unos flujos de trabajo que interpretan y manejan la información entre ambas aplicaciones; finalmente, Grafana, la aplicación responsable de publicar toda la información recogida en InfluxDB de forma visual y fácil de entender.

Para llevar este proyecto a cabo se han requerido tener conocimientos multidisciplinares, ya que se necesitan conocimientos de programación, diseño CAD, networking, contenedores, electrónica, administración de sistemas, etc.

Finalmente, esta memoria mostrará todas las capacidades que el sistema desarrollado puede llevar a cabo.

Palabras clave: IoT, ESP32, MQTT, Docker, Mosquitto, Traefik, NodeRED, InfluxDB, Portainer, Grafana.

Abstract

Bearing in mind the increasing influence of IoT devices in our everyday, the purpose of this work was to explore the viability and development of such device in every aspect, from the programming of the ESP32 to the configuration of the server, which is responsible of publishing the full stack of applications required, in order to gather, interpret and show all the data provided by the ESP32.

The device developed is a thermostat that has a local GUI. Said GUI allows the user to interact with the device thanks to the touch panel included with the development kit. By the way, this device is capable of working locally, manage WiFi networks and it connects to a server too, with the objective of providing a visualization of every data gathered by its four sensors, manage alerts, etc.

From the server side, every software deployed is in Docker containers, forming the following stack of applications: Traefik, a reverse proxy which manages every communication coming to the server and provides SSL certificates too; Portainer, a Docker management software; Mosquitto, a broker that manages the communication through the MQTT protocol; InfluxDB, a time-based database; NodeRED, a graphical tool that sits between Mosquitto and InfluxDB, which creates workflows that interpret and manage the whole information between said applications; finally, Grafana; an application that is responsible of publishing every data located in InfluxDB, in a visual and easy to understand way.

In order to finish this project, it has been required multidisciplinary knowledge, due to the need of programming, CAD software design, networking, electronics, containers and system administration skills.

Finally, this study will show every capability that this system is able to do.

Keywords: IoT, ESP32, MQTT, Docker, Mosquitto, Traefik, NodeRED, InfluxDB, Portainer, Grafana.

Índice general

Capítulo 1	Introducción.....	1
1.1	Motivación.....	1
1.2	Estado del arte	2
1.3	Objetivos a cumplir	3
1.4	Fases.....	4
1.5	Estructura de la memoria	4
Capítulo 2	Investigación	6
2.1	¿Qué es el IoT?.....	6
2.1.1	¿Cuáles son sus aplicaciones?.....	6
2.1.2	¿Qué estructura suele tener?	7
2.1.3	Protocolos IoT.....	8
2.2	Requisitos del sistema.....	10
2.3	Elección de componentes.....	11
2.3.1	A nivel de hardware	11
2.3.1.1	Microcontrolador.....	11
2.3.1.2	Sensores	15
2.3.1.3	Reloj	15
2.3.1.4	Relés	16
2.3.1.5	Regulador de potencia.....	16
2.3.1.6	Buck Converter.....	17

2.3.1.7 Servidor.....	17
2.3.2 A nivel de software.....	17
2.3.2.1 MQTT	18
2.3.2.2 Librerías de programación del ESP32.....	19
2.3.2.3 Docker.....	19
2.3.2.4 Dominio y DNS.....	20
Capítulo 3 Implementación.....	21
3.1 Diseño de las conexiones hardware.....	21
3.2 Programación del microcontrolador	22
3.2.1 Controlador LCD LovyanGFX.....	22
3.2.2 Controlador panel táctil FT6236	24
3.2.3 Librería FreeRTOS.....	24
3.2.3.1 Tareas.....	25
3.2.3.2 Temporizadores.....	27
3.2.4 Librería LVGL	29
3.2.5 Librería RTC.....	30
3.2.6 Librería sensores.....	30
3.2.7 Actuadores	31
3.2.8 Librería WiFi	33
3.2.9 Librería AsyncMQTTClient	34
3.3 Configuración del servidor	35
3.3.1 Servidor, Dominio y DNS.....	35
3.3.2 IoT Stack – Docker	36
3.3.2.1 Traefik	37
3.3.2.2 Portainer.....	38
3.3.2.3 Mosquitto.....	38
3.3.2.4 NodeRED.....	39

3.3.2.5 InfluxDB.....	40
3.3.2.6 Grafana.....	40
3.4 Diseño de la carcasa.....	41
3.4.1 Fusion360	42
3.4.2 Impresión del modelo.....	42
Capítulo 4 Funcionamiento	43
4.1 Diagrama de funcionamiento	43
4.2 Pruebas realizadas	44
4.2.1 Visualización de conexiones en Traefik.....	44
4.2.2 Conexión con el bróker <i>Mosquitto</i>	44
4.2.3 Visualización de datos en NodeRED.....	45
4.2.4 Visualización de conexiones en InfluxDB.....	46
4.2.5 Visualización de conexiones en Grafana.....	47
Capítulo 5 Conclusiones y líneas futuras.....	48
Capítulo 6 Summary and Conclusions.....	49
Capítulo 7 Presupuesto.....	50
Capítulo 8 Anexos	51
8.1 Código del termostato.....	51
8.2 Ficheros de configuración del servidor.....	83
Bibliografía	88

Índice de figuras

Figura 1: Google Trends, termino de búsqueda: IoT[1].....	2
Figura 2: Diagrama de flujo del proyecto	4
Figura 3: Ejemplo estructura IoT.....	7
Figura 4: Modelo de 3, 4 y 5 capas respectivamente[3]	8
Figura 5: Vista de la placa de desarrollo por delante.....	14
Figura 6: Vista de la placa de desarrollo por detrás.....	14
Figura 7: DS18B20 y AM2301 respectivamente	15
Figura 8: DS3231 alimentado por una batería tipo LIR2032.....	15
Figura 9: Módulo de relés de 2 canales, comúnmente usado en Arduino.....	16
Figura 10: Regulador de potencia o Dimmer PWM	16
Figura 11: Regulador de potencia o Dimmer PWM.....	17
Figura 12: Estructura de un mensaje MQTT [38].....	18
Figura 13: Ejemplo de comunicaiación en MQTT [39].....	18
Figura 14: Esquema de conexiones entre los módulos y el ESP32.....	21
Figura 15: Esquema de conexiones en corriente alterna.....	22
Figura 16: Ejemplo de configuración del bus SPI.....	23
Figura 17: Obtención de los pines necesarios en la documentación oficial	23
Figura 18: Definición de los pines SDA y SCL.....	24
Figura 19: Obtención de los pines SDA y SCL.....	24
Figura 20: Datos sobre las tareas.....	26
Figura 21: Handler de la tarea runUI en el fichero task.cpp	26
Figura 22: Creación de la tarea runUI en el fichero task.cpp.....	26
Figura 23: Gestión de memoria para una tarea en FreeRTOS [36].....	27
Figura 24: Función runUI en el fichero task.cpp.....	27
Figura 25: Definición del timer.....	28

Figura 26: Lanzamiento de eventos en setup() en el fichero main.cpp.....	28
Figura 27: Definición de los eventos dentro de networking.cpp.....	28
Figura 28: Creación de la tarea dentro de la función configMQTT() en el fichero networking.cpp.....	29
Figura 29: Cración del menú WiFi en la pantalla de bienvenida en el fichero mcu_ui.cpp.....	29
Figura 30: Visualización del código de la Figura 1.26 en la pantalla del dispositivo.....	29
Figura 31: Asignación del segundo bus I2C, especificado con TwoWire(1) en el fichero rtc.cpp.....	30
Figura 32: Función del control del bombillo UVB en el fichero actuators.cpp.....	31
Figura 33: Función inicialización del dimmer en el fichero actuators.cpp.....	32
Figura 34: Función que fija el valor PWM en el fichero actuators.cpp.....	32
Figura 35: Función que controla el brillo de la pantalla en el fichero task.cpp.....	33
Figura 36: Función que prueba una red en el fichero networking.cpp.....	33
Figura 37: Función que conecta a una red WiFi, llamada en el fichero main.cpp	34
Figura 38: Función definida en el fichero networking.cpp.....	34
Figura 39: Función que suscribe el tópico de MQTT en el fichero networking.cpp	35
Figura 40: Visualización del servidor en la consola de Hetzner.....	36
Figura 41: Configuración del DNS dentro de Cloudflare.....	36
Figura 42: Configuración SSL dentro del fichero de configuración traefik.yml	37
Figura 43: Red externa de Docker proxy-network.....	37
Figura 44: Sección de Portainer dentro del docker-copose.yml.....	38
Figura 45: Sección de Mosquitto dentro del docker-copose.yml	39
Figura 46: Sección de NodeRED dentro del docker-copose.yml	39
Figura 47: Sección de InfluxDB dentro del docker-copose.yml.....	40
Figura 48: Sección de InfluxDB dentro del docker-copose.yml.....	40
Figura 49: Visualización en distintos ángulos de la carcaca.....	42
Figura 50: Visualización en distintos ángulos de la carcaca impresa.....	42
Figura 51: Visualización en distintos ángulos de la carcaca impresa.	43
Figura 52: Visualización de las rutas creadas en el dashboard de Traefik.	44
Figura 53: Visualización por serial del comportamiento del termostato.....	44
Figura 54: Visualización de los logs de Mosquitto en Portainer.....	45
Figura 55: Visualización de los nodos en la interfaz de NodeRED.....	45
Figura 56: Datos recibidos desde NodeRED, visualizados en InfluxDB.....	46
Figura 57: Configuración de la query a InfluxDB en Grafana.....	47
Figura 58: Visualización de todos los sensores en el Dashboard de Grafana.....	47

Índice de tablas

Tabla 1 Presupuesto	50
---------------------------	----

Capítulo 1 Introducción

En este capítulo se hará una breve introducción sobre lo que tratará esta memoria y como está estructurada. Además, se realizará un breve análisis y se describirá en un diagrama como se han tomado las decisiones que han influenciado la elaboración de este trabajo.

1.1 Motivación

La motivación para hacer la investigación y la elaboración de este proyecto ha surgido desde la mera curiosidad, con el fin de analizar, aprender, evaluar e implementar en cierta forma todo lo relacionado con el IoT¹.

El principal objetivo de este proyecto es trabajar todo el *stack*² que interviene en este conjunto de tecnologías, desde la elaboración de un prototipo que integre un microcontrolador, sensores y actuadores, hasta la parte del servidor, donde también será necesario realizar una investigación para comprender qué requisitos habrá que satisfacer con el fin de obtener el mejor de los resultados posibles.

El prototipo que se va a desarrollar en este trabajo será el de un termostato, al ser un gran candidato en el que aplicar el concepto IoT y al tener una utilidad bastante amplia en el día a día.

Dicho termostato será desplegado en un terrario³ dado que en este tipo de entornos es necesario cumplir con unas determinadas condiciones climáticas, las cuales este dispositivo se encargará de supervisar, controlar e informar sobre aspectos como la temperatura-humedad en múltiples zonas, comportamientos inesperados (fallo en la alimentación de algún dispositivo calefactor), etc.

²*stack*: Conjunto de tecnologías implicadas en la creación, desarrollo y ejecución de un determinado servicio o aplicación.

³*terrario*: Recipiente en el que se reproducen las condiciones ambientales necesarias para distintos seres de vida total o parcialmente terrestres.

Cabe destacar que actualmente en el mercado, no existen muchas alternativas que satisfagan las necesidades para este tipo de entornos, ya que las ofertas que hacen los fabricantes se limitan a controlar una temperatura, activar un determinado dispositivo y poco más. El termostato que se va a desarrollar no solo quiere cumplir con esas características, sino que busca ir un poco más allá añadiendo funcionalidades del *Internet de las Cosas*, aportando al usuario final un mayor control y una visión más amplia de lo que está ocurriendo en los espacios que desee controlar.

Finalmente, para que este dispositivo pueda convertirse en una realidad, deberá apoyarse en una infraestructura constituida por un microcontrolador, sensores, reguladores de voltaje, relés, reguladores de potencia, una carcasa impresa en 3D y un servidor para alojar el propio *stack* de aplicaciones.

1.2 Estado del arte

Desde dispositivos *wearables*⁴, neveras inteligentes, interruptores y domótica en general, la importancia del concepto IoT no ha hecho más que aumentar en la última década.

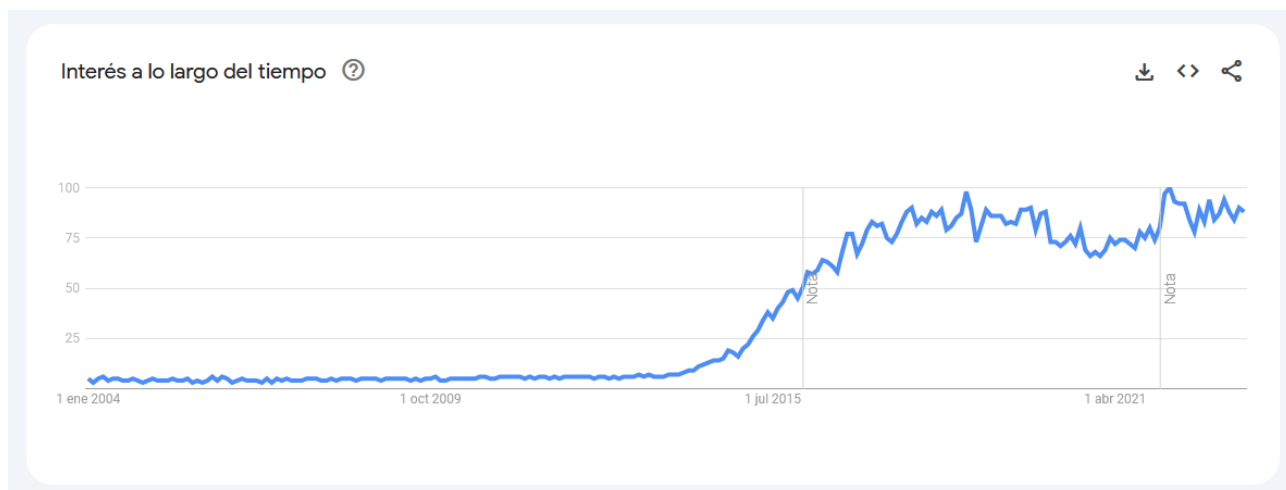


Figura 1: Google Trends, termino de búsqueda: IoT[1]

Parte del mérito de esta creciente evolución es gracias a la accesibilidad del código abierto, tanto a nivel de hardware como software, que ha fomentado que varios programadores independientes y empresas puedan desarrollar sus propias soluciones, ya sean con fines lucrativos o para solucionar determinados problemas o necesidades, y es que, teniendo en cuenta que actualmente vivimos en la sociedad de la información, estos dispositivos han demostrado ser una herramienta indispensable a la hora de recolectar e interpretar los datos de prácticamente cualquier aplicación imaginable.

⁴*wearables*: Todos aquellos dispositivos tecnológicos que son susceptibles de ser utilizados como vestimenta, por ejemplo, un reloj inteligente (*smartwatch*).

A parte del factor del hardware de código abierto, hay que tener en cuenta que esto también ha llevado a que casi cualquier microcontrolador⁵ esté disponible para la población a un precio asumible, por lo que cualquier persona con las habilidades y conocimientos necesarios, será capaz de desplegar un sistema que responda a sus necesidades. Entre los dispositivos de hardware libre más sonados, nos podemos encontrar con las Raspberry Pi, ESP32, Arduino, RISC-V, etc.

1.3 Objetivos a cumplir

Una vez analizada la situación actual de los dispositivos IoT, se ha tomado la decisión de desarrollar un prototipo de termostato que cumpla con los siguientes objetivos:

- Analizar, comprender e implementar tecnologías IoT.
- Programar un microcontrolador, con el fin de ofrecer una GUI⁵ para el usuario, lectura de temperaturas y un sistema de control de iluminación.
- En base a las lecturas obtenidas por el microcontrolador, enviarlas a un servidor online para su posterior procesamiento.
- Ofrecer una interfaz web al usuario donde pueda ver dicha información.
- Finalmente, diseñar un prototipo que integre toda la electrónica dentro de una carcasa, con el fin de servir como plataforma de desarrollo de cara a seguir implementando mejoras tanto a nivel de hardware como software.

⁵GUI: *Graphical User Interface* o interfaz gráfica de usuario. Permite a un usuario interactuar de manera visual con un dispositivo.

1.4 Fases

Para poder llevar este proyecto a cabo, se ha llevado un proceso muy similar al descrito por el siguiente diagrama de flujo:

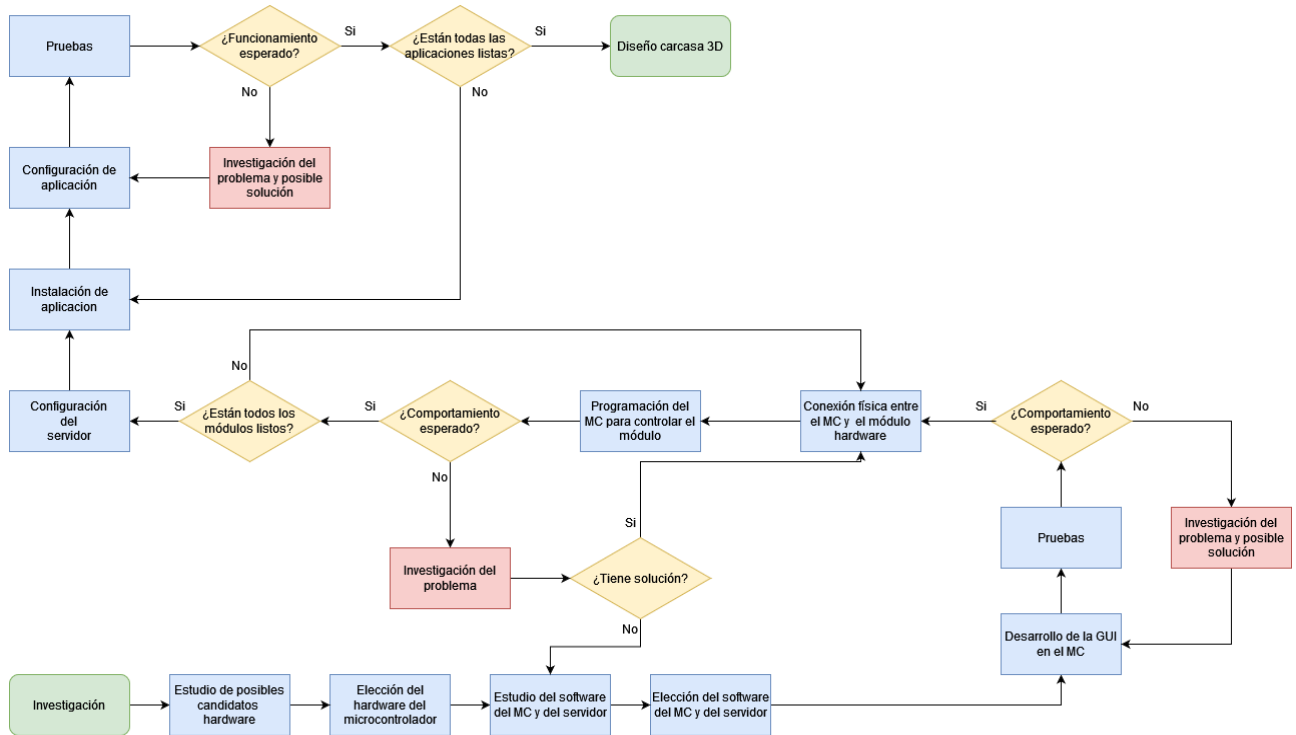


Figura 2: Diagrama de flujo del proyecto

Hay que mencionar que se ha obviado la parte del diseño 3D dado que habría complicado aún más la lectura de este diagrama, por lo que dicho proceso será desarrollado más adelante en la sección 3.4 *Diseño de la carcasa*.

1.5 Estructura de la memoria

A partir de aquí, esta memoria se centrará en explicar todo el trabajo realizado para llegar al prototipo final, comenzando por hacer un primer análisis en el *Capítulo 2 Investigación Previa*. En este capítulo se describirán los diversos conceptos con los que se tendrá que trabajar a lo largo del proyecto, se referirán los diversos protocolos disponibles y las elecciones tanto a nivel de hardware como software que se han tomado.

En el *Capítulo 3 Implementación*, se tratará todo lo relacionado con el proceso de desarrollo e integración de todos los componentes vinculados en este sistema, haciendo una breve explicación y justificación en la toma de decisiones.

Siguiendo al capítulo 3, el *Capítulo 4 Pruebas Realizadas* mostrará una serie de imágenes y explicaciones de como el sistema se comporta bajo los diversos escenarios posibles que se presentarán en el día a día.

Finalmente, en los capítulos 5 y 6, se hablará sobre las conclusiones que se han obtenido al realizar este trabajo, igual que de las líneas futuras y las mejoras que están pendientes de ser adaptas al dispositivo actual. Todo esto seguido del capítulo 7 que muestra un presupuesto de todos los componentes implicados y mano de obra. La memoria se cierra con una bibliografía que contiene todos los enlaces que fueron necesarios para elaborar todo el sistema.

Capítulo 2 Investigación

Partiendo de la introducción realizada en el capítulo anterior, en este capítulo se comenzará a hacer una serie de definiciones sobre los conceptos que serán de importancia para esta memoria. También, se hará un análisis de las diversas tecnologías disponibles y una breve justificación de aquellas que hayan sido seleccionadas para formar parte del prototipo final.

2.1 ¿Qué es el IoT?

El *Internet of Things* o Internet de las cosas en español, es el concepto de conectar dispositivos de distintas o similares características, con el fin de crear una red de comunicación y poder así establecer un proceso de intercambio de datos.

2.1.1 ¿Cuáles son sus aplicaciones?

Partiendo de que la finalidad del IoT es la recopilación de datos y la monitorización de diversos sistemas, este puede ser aplicado a prácticamente cualquier dispositivo. Algunos ejemplos de sus posibles aplicaciones son los siguientes:

- **Electrodomésticos:** Hay neveras capaces de guardar un registro de los alimentos y avisar al usuario en caso de que un determinado alimento falte o esté cercano a su fecha de caducidad.
- **Ropa:** Los relojes inteligentes son capaces hoy en día de monitorizar diversos parámetros de la salud del usuario, como los ciclos del sueño, pulso, etc.
- **Interruptores:** En este apartado puede entrar prácticamente cualquier sistema de gestión eléctrica. Aquí el IoT es bastante útil ya que puede reportar consumos, estados de activación, comportamientos inesperados, etc.
- **Sensores:** Múltiples sensores no solo se limitan a obtener una lectura, si no que ya son capaces de conectarse a una red para ofrecer los datos que van obteniendo.
- **Vehículos:** Capaces de ofrecer datos sobre consumo, ubicación, control remoto, etc.

- **Termostatos:** Uno de los ejemplos más comunes ya que obtienen datos a través de sensores y pueden controlar otros sistemas. Son bastante útiles ya que sus aplicaciones son flexibles, como la monitorización de la temperatura, monitorización del consumo eléctrico, reporte de datos obtenidos, reporte de alertas en caso de comportamientos no deseados, etc.

2.1.2 ¿Qué estructura suele tener?

La figura que se muestra a continuación puede ser uno de los ejemplos de estructuras de un sistema IoT.

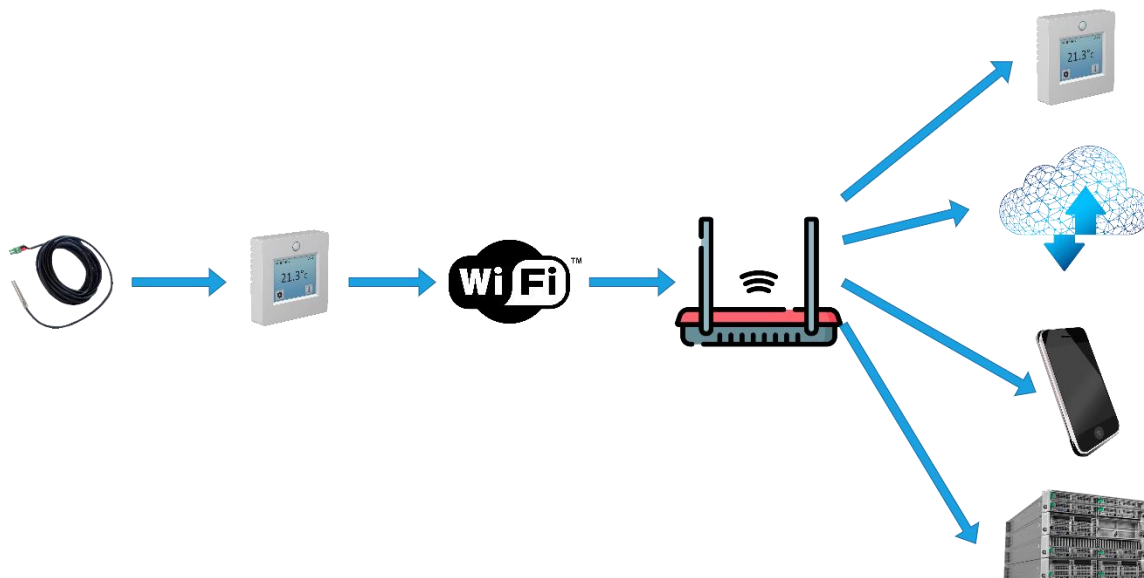


Figura 3: Ejemplo estructura IoT

En ella podemos ver como un sensor se comunica con un termostato reportando un dato, por ejemplo, temperatura. Luego, el termostato se conecta a un dispositivo de red mediante WiFi, o también, Ethernet. Una vez llega al dispositivo de red, este se podría comunicar con otro dispositivo IoT, una nube IoT, un dispositivo personal como un *smartphone*, un ordenador y un servidor alojado de forma local o en Internet.

Como se ha mencionado antes, esto es solo un caso muy sencillo de muchos. En IoT, se suelen clasificar estas estructuras por modelos de capas, empezando por los modelos de tres capas, hasta incluso los de seis en algunos casos, en función de las necesidades del sistema. Dentro de estos modelos, el ejemplo previo podría clasificarse como una arquitectura de tres-capas[2], dado que se puede identificar un dispositivo físico en la capa de percepción (termostato destinado a la recolección de datos), un dispositivo en la capa de red (en este caso, un enrutador) y finalmente una capa de aplicación (donde se sirven los datos a los usuarios u otros dispositivos).

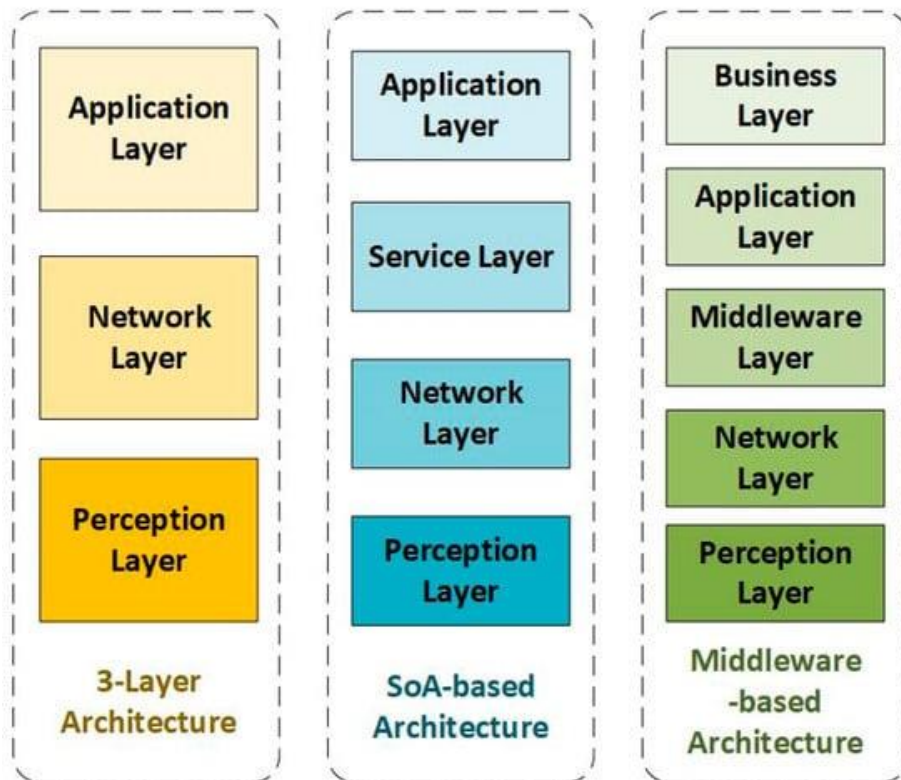


Figura 4: Modelo de 3, 4 y 5 capas respectivamente^[3]

2.1.3 Protocolos IoT

Para que todos los dispositivos en una estructura IoT sean capaces de comunicarse efectivamente, independientemente de las características de cada uno de ellos, es necesario definir una tecnología de transmisión junto con un protocolo de comunicación.

Elegir con criterio una combinación de tecnología y protocolo es bastante importante, ya que hay que tomar las decisiones en base a las características que tenga o requiera la infraestructura. Hay combinaciones que se centran bastante en que el consumo de recursos sea el menor posible, para así reducir consecuentemente el consumo energético y aumentar el tiempo de operación de un determinado dispositivo alimentado por batería; otros, por el contrario, pueden llegar a necesitar que la conexión y transmisión sea lo más estable posible, con latencias⁵ muy bajas.

⁵**Latencia:** Suma de retardos temporales dentro de una red.

Entra las tecnologías de transmisión, se pueden considerar algunos ejemplos, entre ellos, los siguientes:

- **WiFi (*Wireless Fidelity*)**[\[4\]](#): Tecnología inalámbrica de corto-medio alcance que permite la conexión de dispositivos a una red local o a Internet sin necesidad de cables.
- **Bluetooth**[\[5\]](#): Tecnología inalámbrica de corto alcance que permite la conexión y comunicación entre dispositivos.
- **LTE (*Long-Term Evolution*)**[\[6\]](#): Tecnología inalámbrica de alta velocidad utilizada en redes móviles.
- **NB-IoT (*Narrowband Internet of Things*)**[\[7\]](#): Tecnología inalámbrica que se enfoca en aplicaciones que requieren una transmisión de datos esporádica y de baja velocidad.
- **ZigBee**[\[8\]](#): Tecnología inalámbrica que destaca por su bajo consumo de energía, capacidad de autoorganización en redes de malla, lo que le permite escalar con facilidad.
- **LoRa (*Long Range*)**[\[9\]](#): Tecnología inalámbrica que destaca por su capacidad de penetración de objetos y su capacidad para proporcionar conectividad en áreas rurales o remotas por su gran alcance.
- **Ethernet**[\[10\]](#): Tecnología por cable que utiliza un sistema de protocolos para enviar y recibir datos de manera rápida y confiable.

En cuanto a los protocolos de comunicación, algunos de los muchos ejemplos son:

- **LoRaWAN**[\[11\]](#): Utiliza la tecnología LoRa. Sus principales características son su largo alcance, bajo consumo de energía, capacidad para generar redes de malla y seguridad (cifrado extremo a extremo y autenticación).
- **AMQP (*Advanced Message Queuing Protocol*)**[\[12\]](#): Pensado para sistemas de gran tamaño por su escalabilidad, ofrece características de control de mensajería avanzado, enrutamiento, integridad de los mensajes, seguridad (cifrado extremo a extremo y autenticación). Usa un modelo de mensajería y es común en redes empresariales.
- **HTTP (*Hypertext Transfer Protocol*)**[\[13\]](#): Aunque su función principal no esté pensada para dispositivos IoT, se puede utilizar perfectamente. Es un protocolo muy extendido y versátil, con capacidad de encriptación y gestión de la comunicación por solicitudes (GET-POST). Usa un modelo cliente-servidor.
- **CoAP (*Constrained Application Protocol*)**[\[14\]](#): Protocolo muy flexible y de bajo uso de recursos. Se puede utilizar en tándem con HTTP para comunicarse con determinadas aplicaciones WEB. Funciona bien en redes con bajas prestaciones, pudiendo funcionar incluso con mensajes SMS. Usa un modelo cliente-servidor.

- **DDS (Data Distribution Service)**[15]: Pensado para ser utilizado en entornos que la ejecución en tiempo real es crucial (industria, medicina, aviación, etc.), utiliza un modelo de suscripción-publicación (este modelo será detallado más adelante), ofrece QoS⁶, muy robusto, seguro, escalable, complejo etc.
- **MQTT**[16]: Protocolo muy ligero y escalable, utiliza un modelo de suscripción-publicación, QoS, conexión persistente, reconexión automática, capacidad de añadir características de seguridad en las últimas versiones, muy expandido en la industria y probado.

Teniendo todo esto en cuenta, en los siguientes apartados de este mismo capítulo se desarrollará una explicación de la combinación elegida para el funcionamiento del prototipo final.

2.2 Requisitos del sistema

Cuando se decidió que el prototipo iba a ser un termostato, era importante definir los requisitos que todo el sistema/infraestructura debía satisfacer:

- El sistema debe hacer uso de un microcontrolador con capacidad de conectarse a Internet, preferiblemente por WiFi.
- Debe tener al menos 4 sensores, de los cuales 2 de ellos serán solo de temperatura y los otros 2 restantes serán temperatura-humedad.
- Se debe poder controlar un sistema de tres luces en total: dos de ellos por tiempo y el restante por tiempo y temperatura, por lo que para esto hará falta una serie de relés, un reloj y un regulador de potencia en corriente alterna. Estos bombillos serán llamados UVA, que es el encargado de generar luz y calor; UVB, necesario para ciertas especies de animales; Plants, encargado de producir la luz necesaria para el crecimiento de las plantas.
- Se debe ofrecer al usuario una interfaz local donde poder ver la lectura de los sensores, el estado de los relés, información relativa a la conexión, etc.
- El microcontrolador debe ofrecer una forma de cambiar o gestionar la red WiFi de la que hará uso.
- Será necesario disponer de un servidor online donde poder alojar el sistema o las aplicaciones encargadas de recoger, interpretar y mostrar los datos al usuario.

⁶**QoS**: *Quality of Service* o calidad de servicio es un mecanismo que sirve para medir el rendimiento o el estado de una red, pudiendo negociar ciertos parámetros de la comunicación en función del valor QoS.

- El dispositivo debe ser capaz de trabajar en varios escenarios:
 - Sin conexión a Internet.
 - Con conexión a Internet.
 - Con conexión a Internet, pero sin conexión al servidor.
- El sistema debe ser capaz de disponer de un sistema de alertas al igual que una interfaz web para la visualización de los datos obtenidos.

2.3 Elección de componentes

En base a la investigación y la definición de requisitos previa, se muestra a continuación todos los componentes involucrados en la infraestructura propuesta, haciendo hincapié en la elección del microcontrolador. Nótese que elementos como los bombillos, herramientas, cables y conectores no serán tratados en este apartado.

2.3.1 A nivel de hardware

2.3.1.1 Microcontrolador

Como bien se comentó antes, existen cada vez más candidatos en este apartado. Se podría utilizar algún SBC⁷ de Raspberry Pi (por ejemplo 2, 3A o 3B+), Orange Pi, ODroid, etc. pero estas opciones en el momento actual o son bastante caras o no están disponibles en stock debido a la actual crisis del silicio[5].

Si se opta por la vía de un MCU, al igual que los SBC⁷, existen multitud de opciones, pero a diferencia de estos, los microcontroladores son bastante más baratos. Con la intención de simplificar la fase de desarrollo, existen en el mercado diversas placas de desarrollo que integran ya estos microprocesadores junto con otras características adicionales, como pueden ser la accesibilidad de los puertos de comunicación, integración de los módulos WiFi, integración de paneles LCD, etc.

Teniendo en cuenta que el prototipo que se va a desarrollar no requiere de unas prestaciones muy altas, la opción más razonable es la de optar por una placa de desarrollo que integre un MCU⁸. Dicho esto, podemos tener en cuenta los siguientes grandes nombres en el mercado de hardware de código abierto para realizar una elección:

⁷**SBC:** *Single Board Computer* o computadora monoplaca es un ordenador completo que integra en una sola placa todos los componentes necesarios (microprocesador, memoria, etc) para poder funcionar.

⁸**MCU:** Microcontrolador.

- **Arduino:** Una de las compañías más conocidas en el espacio por su gran conjunto de herramientas y facilidad de uso. Cualquier placa de desarrollo de Arduino como la UNO R3, Mega o Nano son buenos candidatos, pero necesitarían de módulos externos para poder obtener las características WiFi, Bluetooth o un panel LCD. Recientemente, Arduino lanzó al mercado la UNO R4 y la GIGA R1, ambas con capacidades WiFi, pero sigue haciendo falta añadir un módulo externo para implementar una pantalla.
- **ESP32:** Desarrollado por la compañía *Espresif Systems* como versión superior al popular ESP8266. Es cada vez más común y utilizado para sistemas IoT por traer ya incluidos el WiFi-Bluetooth y existen multitud de versiones y placas de desarrollo con diferentes capacidades. De entre todas estas placas, ha destacado una en especial, la **MaTouch ESP32-S3 SPI** (del fabricante *Makerfabs*) que integra un panel táctil LCD, todo en una misma placa.
- **RISC-V:** Principalmente conocido por ser una ISA⁹, hay microcontroladores que ya integran dicho conjunto específico de instrucciones. Diferentes compañías han creado placas de desarrollo como la GD32VF103T-START (del fabricante *GigaDevice*) o la **MaTouch ESP32-C3** (del fabricante *Makerfabs*).

Estos son solo algunos de los ejemplos de placas de desarrollo disponibles en el mercado, pero considerando que ambos modelos de Makerfabs ya presentan unas características bastante atractivas de cara al prototipo, se ha optado por escoger uno de sus modelos, concretamente el **MaTouch ESP32-S3 SPI** al disponer de fácil acceso de los pines GPIO y al tener también una arquitectura más estable y probada por la comunidad.

Para analizar las capacidades de esta placa, hay que fijarse en las características generales del propio **MCU ESP32-S3**:

- Doble procesador de 32bit Xtensa LX7 a 240Mhz máximo
- RISC-V ULP (Ultra Low Power) coprocesador, para gestionar las tareas del reloj interno en el modo Deep-sleep
- 2.4GHz Wifi - 802.11b/g/n
- Bluetooth 5, BLE + Malla
- 8MB adicionales QSPI PSRAM
- Modo Ultra Low Deep-Sleep

⁹**ISA:** *Instruction Set Architecture* o conjunto de instrucciones es una especificación que detalla las instrucciones con las que trabajará un procesador determinado.

- Conector USB-C con protección de retroalimentación.
- USB Nativo + USB Serial JTAG + USB OTG
- RGB LED de bajo consumo
- Antena 3D de alta ganancia y un conector tipo FL para antenas externas.
- 45 GPIOs
- 4 interfaces SPI
- 3 interfaces UART
- 2 interfaces I2C
- Controlador PWM de 8 canales
- Una ranura para una tarjeta microSD

Ahora sí, observando la placa de desarrollo, nos encontramos que el fabricante ha configurado el ESP32-S3 para que la placa pueda ofrecer las siguientes características (omitiendo esta vez las características comunes al propio ESP32-S3):

- Pantalla LCD de 3.5 pulgadas
- Una interfaz disponible para I2C, 11 pines GPIO, una salida de 3.3V y GND.
- Driver del LCD: ILI9488, conectado por SPI
- Resolución LCD: 320 * 480
- Driver del panel táctil: FT6236
- Botón de reset y flash.
- Fuente de alimentación por USB-C de 5.0V.
- Tamaño de la placa: 66mm * 85mm

Finalmente, es importante no confundir el modelo de la placa de desarrollo SPI¹⁰ con el modelo que utiliza la interfaz paralela, ya que la placa de desarrollo que utiliza esta última interfaz no dispone de los 11 GPIO¹¹ que ofrece la versión SPI. La versión paralela hace este sacrificio en pro obtener una pantalla con una tasa de refresco mucho más alta, dando una experiencia al usuario mucho más fluida con el contenido que puede ver en la pantalla.



Figura 5: Vista de la placa de desarrollo por delante



Figura 6: Vista de la placa de desarrollo por detrás

¹⁰**SPI:** *Serial Peripheral Interface* o bus SPI es un estándar de comunicaciones en sistemas electrónicos.

¹¹**GPIO:** *General Purpose Input/Output* o pin de entrada/salida de propósito general, es un tipo de pin que puede ser configurado según las necesidades del desarrollador.

2.3.1.2 Sensores

Respecto a los sensores hay múltiples opciones posibles. Dos sensores de temperatura **DS18B20** y dos sensores temperatura-humedad **AM2301** serán suficientes.



Figura 7: DS18B20 y AM2301 respectivamente

2.3.1.3 Reloj

Para llevar un control del tiempo y mantener la hora en caso de un fallo de energía, es interesante utilizar el módulo de reloj en tiempo real **RTC-DS3231**, el cual integra una batería para poder mantener la configuración en estos casos.

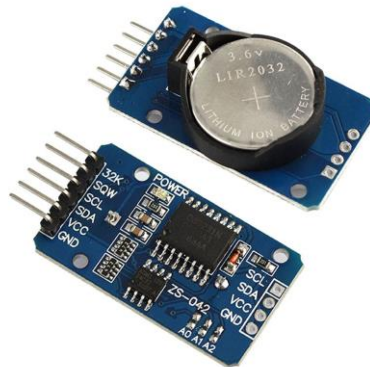


Figura 8: DS3231 alimentado por una batería tipo LIR2032

2.3.1.4 Relés

Dos de los bombillos a 220V serán activados-desactivados gracias a un módulo de relés de dos canales.



Figura 9: Módulo de relés de 2 canales, comúnmente usado en Arduino

2.3.1.5 Regulador de potencia

Para controlar el bombillo encargado de generar el calor, es necesario disponer de un regulador que controle la potencia en función de la lectura que aportará alguno de los dos sensores. Es por eso por lo que hará falta un regulador de potencia 220V AC, que controlará dicho bombillo mediante PWM¹².

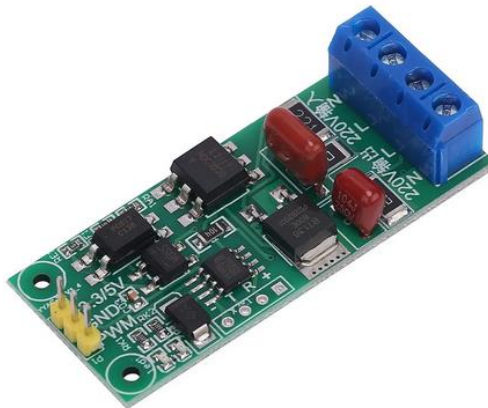


Figura 10: Regulador de potencia o Dimmer PWM

¹²**PWM:** *Pulse Width Modulation* o modulación por ancho de pulsos es una técnica de modificación de señales la cual varía el ciclo de trabajo en una señal periódica.

2.3.1.6 Buck Converter

El termostato podrá obtener la alimentación necesaria gracias a un *buck-converter* o circuito convertidor reductor. Su principal función es recibir un voltaje de entrada de una determinada fuente de alimentación externa y convertirlo a un voltaje que pueda utilizar la electrónica. Para este caso, este circuito convierte 9V a 5V que alimentan la placa de desarrollo y a los relés directamente. En el capítulo 3 se muestra un esquema de conexiones para poder comprender mejor como se ha realizado este proceso.



Figura 11: Regulador de potencia o Dimmer PWM

2.3.1.7 Servidor

En este apartado, se presentan varias opciones. Se podría alojar el servidor localmente, utilizando cualquier ordenador, pc, etc. o también se podría contratar un **VPS**¹³ en la nube. Dado que en este proyecto se pretende investigar como funcionaría un dispositivo IoT, se ha optado por esta última opción al ser la opción más realista.

Existen múltiples proveedores de VPS. Para este sistema en concreto se ha optado por el proveedor *Hetzner* dado que ofrece una máquina con buenas características a un precio razonable.

La máquina posee dos CPU virtuales, 4GB de RAM, 40GB de almacenamiento SSD y 20TB de tráfico.

2.3.2 A nivel de software

Teniendo claro qué hardware va a ser utilizado, ahora se puede decidir qué software se va a utilizar en conjunto.

¹³**VPS**: Partición de un servidor virtual.

2.3.2.1 MQTT

De entre todos los protocolos comentados en el apartado *2.1.3 Protocolos IoT*, se ha escogido MQTT por su rapidez, sencillez, características de seguridad (autenticación, TSL/SSL), sus tres niveles de *QoS* y la comunidad que lo respalda.

MQTT realiza sus comunicaciones apoyándose en el protocolo TCP/IP y los mensajes que envía constan de la siguiente estructura:

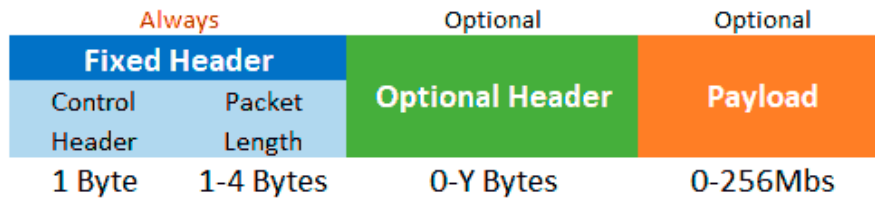


Figura 12: Estructura de un mensaje MQTT [38]

En la *Figura 1.11*, nos encontramos con: una cabecera fija de 2-5 bytes, que contiene un código de control, un identificador del mensaje, la longitud total del mensaje y opcionalmente, *flags*; seguido de una cabecera opcional que puede definir parámetros adicionales de la comunicación; finalmente, un *payload* o contenido que almacena la información que se desea transmitir, con un máximo de 256 Mbs.

Al igual que otros protocolos IoT, MQTT utiliza un modelo de publicación-suscripción. Dicho modelo publicación-suscripción funciona de la siguiente manera: el cliente se encarga de crear un tópico (por ejemplo, información relativa a los sensores de un entorno) y lo publica en un bróker, con el fin de que otros sistemas que se comuniquen con el bróker puedan suscribirse a dicho tópico, recibiendo así la información a medida que se vaya actualizando.

Como MQTT necesita de un bróker que gestione todas las peticiones que se puedan llegar a producir durante una comunicación, es importante tener una idea general de cómo se producen dichas comunicaciones. En la siguiente figura se puede observar un ejemplo de dicha comunicación entre un cliente MQTT y su bróker:

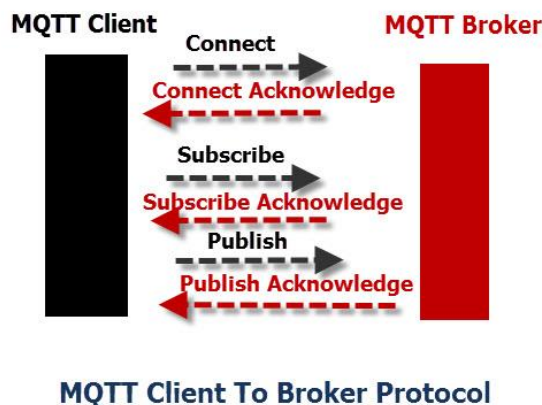


Figura 13: Ejemplo de comunicación en MQTT [39]

En el *Capítulo 4 Pruebas realizadas* se podrá comprender mejor cómo funciona este protocolo, dado que se mostrará un ejemplo de su funcionamiento.

2.3.2.2 Librerías de programación del ESP32

En cuanto a las librerías seleccionadas para programar el ESP32, destacan las siguientes:

- **LovyanGFX**[17]: Driver que controla el panel LCD.
- **FT6236**[18]: Driver que controla las interacciones con el panel táctil.
- **LVGL**[19]: Librería de gráficos predefinidos que ayuda a construir una GUI.
- **FreeRTOS**[20]: Librería que permite implementar un sistema de tareas similar al de un sistema operativo en tiempo real, además de gestión de eventos.
- **WiFi**[21]: Librería para poder gestionar la comunicación WiFi.
- **AsyncMQTTClient**[22]: Librería que permite implementar el protocolo MQTT dentro del ESP32.
- **ArduinoJSON**[23]: Librería que permite montar la información de todos los sensores en una sola estructura de datos json.
- **RTCLib**[24]: Librería encargada de configurar el reloj DS3231.
- **DallasTemperature**[25] y **DHT**[26]: Librerías que permiten la configuración de todos los sensores de temperatura y humedad.

2.3.2.3 Docker

Para preparar y configurar las aplicaciones dentro del servidor online, se ha tomado la decisión de utilizar los contenedores de Docker para todo el *stack*, lo que facilita a la larga el mantenimiento y gestión de este. En cuanto a las aplicaciones que conforman dicho *stack*, nos podemos encontrar con:

- **Portainer**[28]: Herramienta de gestión de contenedores. Ofrece las mismas capacidades que la CLI de Docker, con la diferencia de que toda la gestión es más accesible al interactuar con ella a través de una interfaz web.
- **Traefik**[29]: Un *reverse proxy* que gestiona todas las comunicaciones y peticiones con el resto de los contenedores. También, permite que el acceso web a dicho contenedores puedan obtener un certificado SSL de forma muy sencilla. Como alternativas a Traefik, están Nginx, Caddy, etc.
- **Mosquitto**[30]: Es el bróker que MQTT necesita para poder establecer y gestionar una comunicación entre el ESP32 y el servidor. Alternativas posibles son RabbitMQ, CrystalMQ, HiveMQ, etc.

- **NodeRED**[\[31\]](#): Esta aplicación ofrece una interfaz web que permite crear flujos de comunicación de forma gráfica entre diferentes aplicaciones. Alternativas posibles pueden ser un sistema de scripts que extraigan la información del bróker y la inyecten en una base de datos, exportadores o *mqtt-parsers*, etc.
- **InfluxDB**[\[32\]](#): Base de datos que funciona con marcas de tiempo. Alternativas pueden ser Prometheus, TimeScaleDB, MongoDB, etc.
- **Grafana**[\[33\]](#): Finalmente, la aplicación responsable de publicar las gráficas en base a los datos almacenados en la base de datos, además de ofrecer un sistema de monitorización y alertas. Alternativas a Grafana podrían ser Kibana, NetData, etc.

Como se habrá podido observar, a parte de las elecciones mostradas se ha comentado también posibles alternativas, lo que quiere decir que existen múltiples formas de configurar el *stack* de aplicaciones. No hay solo una única forma de resolver el problema.

2.3.2.4 Dominio y DNS

Finalmente, para facilitar el acceso y la gestión de las peticiones web en los contenedores, se ha optado por comprar el dominio *tfg-ull.cloud* y se ha configurado un servidor DNS utilizando la versión gratuita de *Cloudflare*.

Capítulo 3 Implementación

En el capítulo anterior se ha realizado la investigación, definido los requisitos y escogido los componentes. A partir de aquí, se explica cómo se han conectado y programado todos los elementos involucrados en la constitución de este termostato.

3.1 Diseño de las conexiones hardware

En este apartado se podrá entender como se han conectado físicamente todos los dispositivos. En la siguiente figura se muestra un esquema de como se han establecido las conexiones entre los diferentes módulos y la placa de desarrollo:

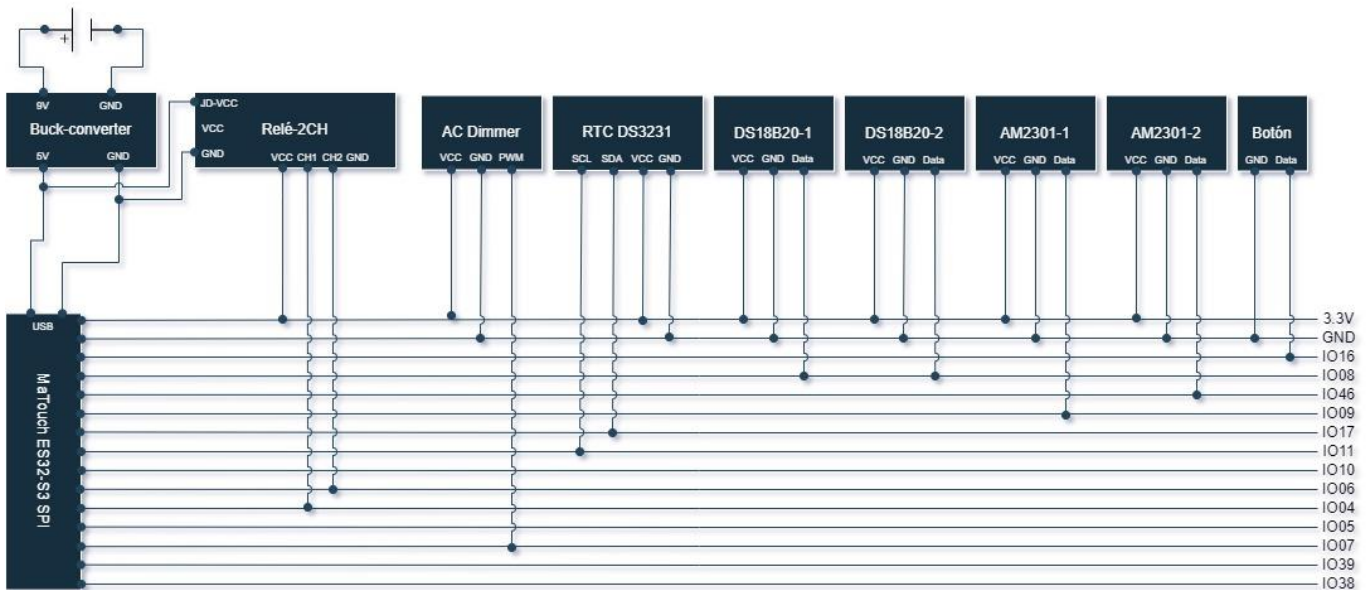


Figura 14: Esquema de conexiones entre los módulos y el ESP32

Y en esta otra figura se muestra un esquema de como se han establecido las conexiones que gestionan la parte de la corriente alterna, representando en marrón el cable de la fase y en azul el cable neutro:

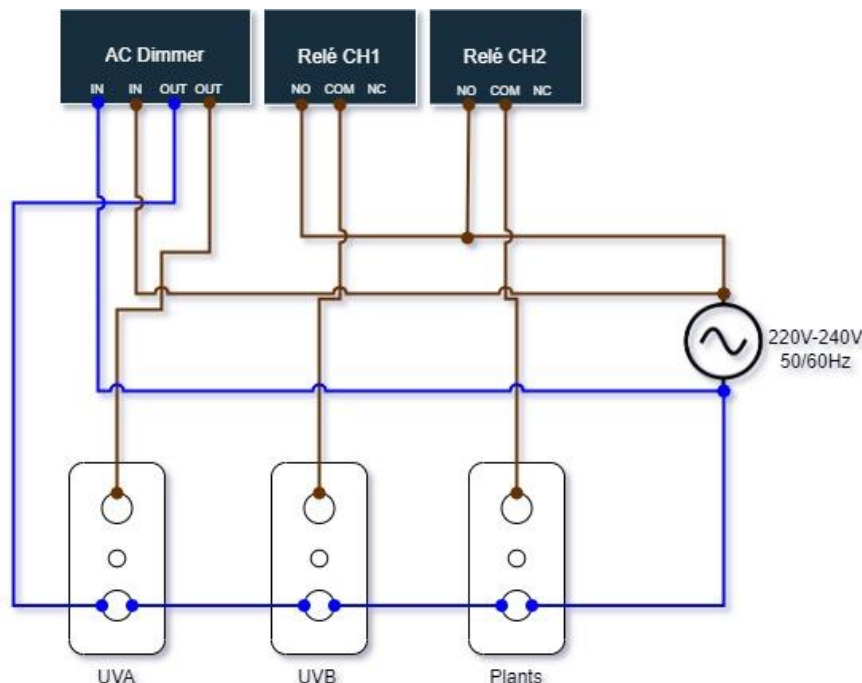


Figura 15: Esquema de conexiones en corriente alterna

3.2 Programación del microcontrolador

Toda la programación del ESP32 se ha hecho utilizando el framework¹⁴ de Arduino, utilizando *Visual Studio* y el plugin de *Platformio*[35], el cual permite conectarse al ESP32 de una forma muy sencilla.

3.2.1 Controlador LCD LovyanGFX

Este controlador es el responsable de que el panel LCD conectado al ESP32 pueda recibir la información por parte de la librería LVGL y así poder visualizar las distintas pantallas de la interfaz gráfica. Es un controlador muy versátil dado que ofrece la posibilidad de configurar el bus SPI al cual está conectado la LCD, las propias características de la LCD, el brillo con el que trabajará el panel y finalmente, también ofrece la posibilidad de controlar el panel táctil, si lo hubiera. Esta última característica,

¹⁴**Framework:** Marco de trabajo que incluye una serie de herramientas para el desarrollo.

sin embargo, no fue configurada, dado que no se logró que fuera tan preciso como si lo era con el controlador FT6263, analizado en el siguiente apartado.

Para poder configurar correctamente este controlador es necesario disponer de algún tipo de documentación por parte de los creadores de la placa de desarrollo[34] y también leer la documentación del creador del controlador, disponible en GitHub[17]. Se muestra la configuración del bus SPI como ejemplo.

```

// Configuración del bus SPI, para controlar la LCD
{
  auto cfg      = _bus_instance.config();
  cfg.spi_host  = SPI3_HOST;
  cfg.spi_mode  = 0;
  cfg.freq_write = 40000000;
  cfg.freq_read  = 16000000;
  cfg.spi_3wire  = true;
  cfg.use_lock   = true;
  cfg.dma_channel = 1;
  cfg.pin_sclk  = 14;
  cfg.pin_mosi  = 13;
  cfg.pin_miso  = 12;
  cfg.pin_dc    = 21;
  // Carga la configuración para preparar el bus SPI
  _bus_instance.config(cfg);
  _panel_instance.setBus(&_bus_instance);
}

```

Figura 16: Ejemplo de configuración del bus SPI

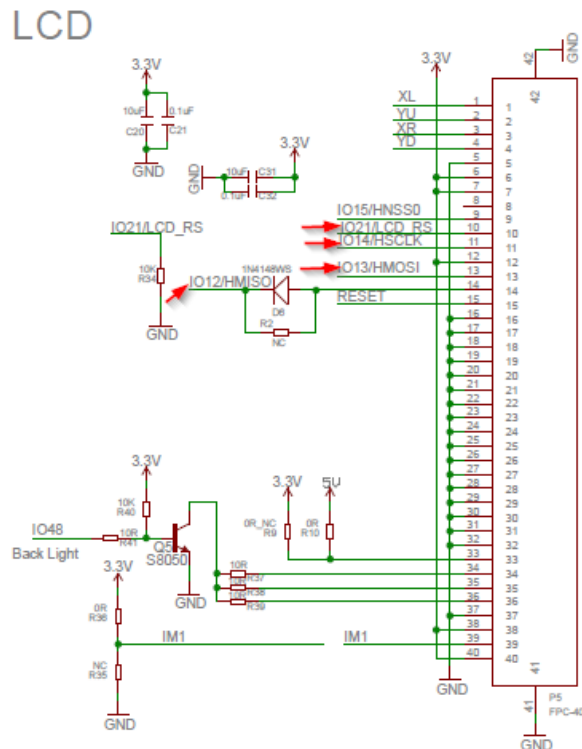


Figura 17: Obtención de los pines necesarios en la documentación oficial

3.2.2 Controlador panel táctil FT6236

Como bien se ha comentado antes, no se configuró el controlador táctil que ofrecía LovyanGFX. Se escogió el FT6236 porque aportó mejores resultados en cuanto a precisión se refiere. Su integración en el sistema es necesaria dado que gracias a ella se permite al usuario poder interactuar con la interfaz gráfica.

Esta librería apenas necesita de configuración adicional. Lo único que hay que añadir son los pines SDA y SCL del bus I2C al cual el panel táctil está conectado. Se puede encontrar la documentación en su repositorio de GitHub[18].

```
/**
 * @brief Pin SDA asignado al
 * de desarrollo:
 * https://github.com/Makerfa
 */
#define SDA_FT6236 38
/**
 * @brief Pin SCL asignado al
 * de desarrollo:
 * https://github.com/Makerfa
 */
#define SCL_FT6236 39
```

Figura 18: Definición de los pines SDA y SCL

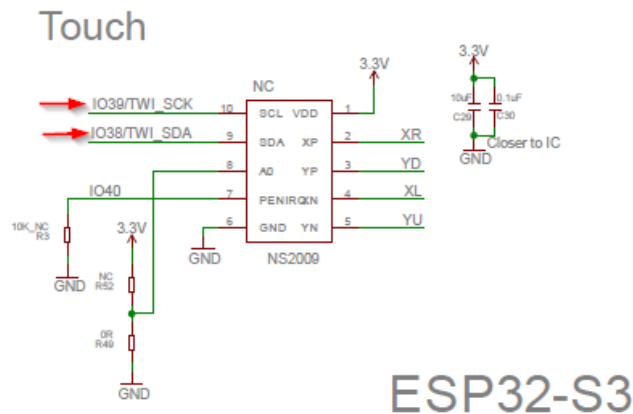


Figura 19: Obtención de los pines SDA y SCL

3.2.3 Librería FreeRTOS

El principal motivo de usar esta librería se debe a que facilita la implementación del código en los dos núcleos que posee el ESP32. Esto es interesante ya que permite que todos

los procesos sean ejecutados en el *core-1*, salvo el proceso encargado de publicar los datos al bróker MQTT, que lo hace desde el *core-0*. De esta forma se asegura que en ningún momento la transmisión de datos se vea interrumpida o condicionada por la ejecución de otros procesos.

FreeRTOS[20] está especializada en la implementación de sistemas operativos en tiempo real en diversos microcontroladores. De todas las características que ofrece, solo se utilizan dos de ellas, las tareas y los temporizadores.

3.2.3.1 Tareas

Las tareas han sido utilizadas en este proyecto para definir una organización a la hora de ejecutar diversos procesos en ambos núcleos del ESP32. Estas tareas son controladas por el *scheduler*¹⁵ de FreeRTOS que utiliza un sistema de prioridades fijas con *round-robin*¹⁶. Existe un total de diez tareas:

- ***runUI***: Tarea encargada de gestionar el *handler* de LVGL, el cual está a la escucha de si se produce un evento, como, por ejemplo, que al tocar un botón en la interfaz se produzca una acción programada.
- ***updateScreenBrighnessTask***: Tarea encargada de controlar las peticiones del botón físico que activa/desactiva el brillo de la pantalla.
- ***writeSensorsDataUITask***: Tarea encargada de escribir en la interfaz los datos recogidos por los sensores.
- ***runClockUITask***: Tarea encargada de escribir la hora en la interfaz.
- ***updateActiveTimeUVA******Task***: Tarea encargada de recoger la configuración horaria especificada por el usuario en la interfaz que define el tiempo de activad del bombillo UVA.
- ***updateActiveTimeUVB******Task***: Tarea encargada de recoger la configuración horaria especificada por el usuario en la interfaz que define el tiempo de activad del bombillo UVB.
- ***updateActiveTimePlants******Task***: Tarea encargada de recoger la configuración horaria especificada por el usuario en la interfaz que define el tiempo de activad del bombillo Plants.
- ***setMaxTemperatureUVA******Task***: Tarea encargada de recoger la temperatura máxima permitida del bombillo UVA, especificada por el usuario en la interfaz.
- ***controlLightsSystemTask***: Tarea encargada de controlar todo el sistema de

¹⁵**Scheduler**: Planificador que dictamina y organiza la ejecución de un determinado proceso.

¹⁶**Round-robin**: Generalizando, división de tiempo equitativa para todos los procesos en un sistema, de forma cíclica.

luces, en función del tiempo y la temperatura programada por el usuario desde la interfaz.

- ***publishMQTTBrokerTask***: Tarea encargada de publicar los datos recogidos por los sensores al bróker MQTT.

Tarea	Núcleo	Prioridad	Periodo de ejecución
runUI	1	3	5 ms
updateScreenBrightnessTask	1	1	140 ms
writeSensorsDataUITask	1	3	100 ms
runClockUITask	1	3	1000 ms
updateActiveTimeUVATask	1	3	100 ms
updateActiveTimeUVBTask	1	3	100 ms
updateActiveTimePlantsTask	1	3	100 ms
setMaxTemperatureUVATask	1	3	100 ms
controlLightsSystemTask	1	3	1000 ms
publishMQTTBrokerTask	0	3	5000 ms

Figura 20: Datos sobre las tareas

Teniendo en cuenta las características del *scheduler* antes comentadas, la única tarea que se vería interrumpida por tener una prioridad más baja sería *updateScreenBrigthnessTask*.

El proceso de creación de una tarea consta de tres partes, tomando como ejemplo a la tarea *runUI*:

1. La definición de un *handler*, que es quien determina cuando se para, se activa o se elimina una tarea.

```
TaskHandle_t lvglHandler = NULL;
```

Figura 21: Handler de la tarea *runUI* en el fichero *task.cpp*

2. La creación de la propia tarea, que consta de los siguientes parámetros:

```
xTaskCreatePinnedToCore(runUI, "UITask",
                        4096, NULL, 3, &lvglHandler, 1);
```

Figura 22: Creación de la tarea *runUI* en el fichero *task.cpp*

- a. **runUI**: Función a la que llamará periódicamente.
- b. **“UITask”**: Descripción de la tarea.
- c. **4096**: Cada tarea tiene su propio espacio de memoria definido. 4096 bytes es la cantidad de memoria asignada a esta.

RTOS Memory Allocation

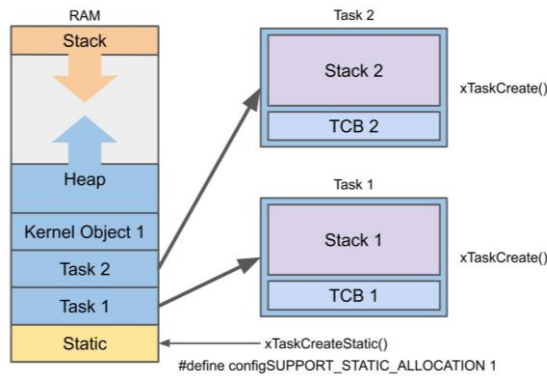


Figura 23: Gestión de memoria para una tarea en FreeRTOS [36]

- d. **NULL**: Aquí irían los parámetros, en caso de que la función *runUI* los necesitara.
 - e. **3**: La prioridad de la tarea.
 - f. **&lvglHandler**: El *handler* de la tarea.
 - g. **1**: El identificador del núcleo donde se va a ejecutar, en este caso, el segundo núcleo.
3. Definir la función que será llamada por la tarea. Todas las funciones que son llamadas por una tarea deben estar dentro de un bucle infinito y especificar antes del cierre de dicho bucle, con qué periodo se debería de ejecutar dicha tarea. En este caso, se ha especificado que la tarea se ejecute después de 5 ms, una vez se hayan finalizado todas las operaciones.

```
static void runUI (void *args) {  
    while (1) {  
        lv_timer_handler();  
        vTaskDelay(5 / portTICK_RATE_MS);  
    }  
}
```

Figura 24: Función *runUI* en el fichero *task.cpp*

3.2.3.2 Temporizadores

Al igual que las tareas, los *timers* o temporizadores también se utilizan para controlar tareas a lo largo del tiempo, con el beneficio de ser elementos más ligeros que una tarea.

La creación de estos temporizadores fue necesaria dado que cuando se intentó implementar la gestión de las comunicaciones WiFi y del protocolo MQTT con tareas, surgieron multitud de problemas en cuanto a la propia gestión de los eventos (conexión,

deconexión, etc.), bloqueos en los núcleos, etc.

Con los *timers* se simplifica todo este proceso, dado que cuando el ESP32 genera un evento, por ejemplo, de desconexión, se puede iniciar un *timer* específico para iniciar un proceso de reconexión.

Se debe mencionar que este sistema consta de dos timers, uno para gestionar la conexión WiFi y otro para gestionar las comunicaciones del protocolo MQTT.

El proceso de creación de un temporizador consta de las siguientes partes, tomando como ejemplo la gestión del WiFi:

- Definición del *timer*.

```
TimerHandle_t wifi_reconnect_timer;
```

Figura 25: Definición del timer

- Creación del sistema de eventos, que en este caso reportarán actividad en caso de obtener una IP o perder la conexión con el punto de acceso.

```
WiFi.onEvent(wifiEvent);
```

Figura 26: Lanzamiento de eventos en *setup()* en el fichero *main.cpp*

```
void wifiEvent(WiFiEvent_t event) {
  switch (event) {
    case SYSTEM_EVENT_STA_GOT_IP:
      Serial.println("Evento: Se ha obtenido IP");
      lv_obj_add_state(ui_wifiswitch, LV_STATE_CHECKED);
      connectMQTT();
      xTimerStop(wifi_reconnect_timer, 0);
      break;
    case SYSTEM_EVENT_STA_DISCONNECTED:
      Serial.println("Se ha desconectado del WiFi.");
      xTimerStop(mqtt_reconnect_timer, 0);
      xTimerStart(wifi_reconnect_timer, 0);
      lv_obj_clear_state(ui_wifiswitch, LV_STATE_CHECKED);
      break;
  }
}
```

Figura 27: Definición de los eventos dentro de *networking.cpp*

- Finalmente, la propia creación del *timer*: que consta de una descripción “WiFiTimer”; el tiempo de ejecución periódico que son 5000 ms; la opción de recrear el *timer* cada 5000ms, que está deshabilitada con *pdFalse*; un identificador del *timer*; finalmente, la función que ejecutará, que en este caso será la que se encarga de intentar reconectar el ESP32 al punto de acceso WiFi.

```
wifi_reconnect_timer = xTimerCreate("WiFiTimer",
                                   pdMS_TO_TICKS(5000),
                                   pdFALSE,
                                   (void*)0,
                                   reinterpret_cast<TimerCallbackFunction_t>(wifiReconnect)
                                   );
```

Figura 28: Creación de la tarea dentro de la función `configMQTT()` en el fichero `networking.cpp`

Como cierre a este apartado, cabe destacar que como se habrá podido observar en la figura, es el programador el encargado de gestionar cuando se activa y desactiva un *timer*, mediante el uso de las funciones `xTimerStop()` y `xTimerStart()`, respectivamente.

3.2.4 Librería LVGL

Light and Versatile Graphics Library o LVGL, es una librería especializada en generar gráficos en sistemas empotrados, como pueden ser las Arduino, ESP32, etc.

Consta de multitud de elementos gráficos predefinidos llamados *widgets*, los cuales pueden ser colocados y personalizados en prácticamente cualquier forma.

Gracias a ella, se ha podido generar los gráficos de la interfaz del microcontrolador. Para crear la GUI, se ha tenido que escribir el código necesario además de apoyarse en una herramienta gráfica que se integra con LVGL llamada *SquareLine Studio*[\[37\]](#).

```
void drawWiFiMenu(WiFiScanData wifi_data) {
    current_found_networks = wifi_data.size();
    if (ui_wifilist != NULL) {
        if (current_found_networks != last_found_networks) {
            lv_obj_clean(ui_wifilist);
            lv_list_add_text(ui_wifilist, "Opciones");
            ui_wifilistoptions = lv_list_add_btn(ui_wifilist, LV_SYMBOL_TRASH, "Borrar red actual");
            lv_obj_add_event_cb(ui_wifilistoptions, uiEventCleanFlash, LV_EVENT_ALL, NULL);
            ui_wifilistoptions = lv_list_add_btn(ui_wifilist, LV_SYMBOL_CLOSE, "Continuar con los ajustes actuales");
            lv_obj_add_event_cb(ui_wifilistoptions, uiEventCloseWiFiWindow, LV_EVENT_ALL, NULL);
            lv_list_add_text(ui_wifilist, "Redes disponibles");
            for (int i = 0; i < wifi_data.size(); i++) {
                ui_wifilistoptions = lv_list_add_btn(ui_wifilist, LV_SYMBOL_WIFI, wifi_data[i].c_str());
                lv_obj_add_event_cb(ui_wifilistoptions, uiEventWriteWiFicredentials, LV_EVENT_ALL, NULL);
            }
            last_found_networks = current_found_networks;
        }
        if (flash_message_box_requested) {
            drawCleanFlashMessageBox();
        } else if (credentials_message_box_requested) {
            drawCredentialsMessageBox();
        }
    }
}
```

Figura 29: Creación del menú WiFi en la pantalla de bienvenida en el fichero `mcu_ui.cpp`

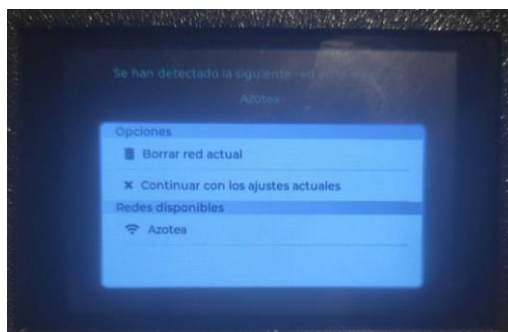


Figura 30: Visualización del código de la Figura 1.26 en la pantalla del dispositivo.

3.2.5 Librería RTC

Esta librería se encarga de aportar las herramientas necesarias para poder controlar el módulo de reloj en tiempo real DS3231. Gracias a esto se pueden obtener datos relacionados con la cuantificación del tiempo, lo que permitirá definir periodos de actividad para los bombillos que se desee controlar.

Aunque es muy sencilla de configurar, hay que tener en cuenta un aspecto muy importante a la hora de realizar la implementación. El ESP32 dispone de dos buses I2C y en este sistema, el primer bus ya está siendo utilizado por el panel táctil.

El módulo DS3231 necesita de un bus I2C para poder realizar la comunicación con el MCU, por lo que habrá que especificarle que debe escoger el segundo bus, al estar el primero ya ocupado.

```
RTC_DS3231 rtc;  
TwoWire i2c_rtc = TwoWire(1);
```

Figura 31: Asignación del segundo bus I2C, especificado con *TwoWire(1)* en el fichero *rtc.cpp*

Como se habrá podido observar en la figura anterior, con *TwoWire(1)* se está seleccionando el segundo bus. De haber escrito *TwoWire(0)*, habría un conflicto con el bus que está siendo utilizado por el panel táctil.

Entre las funciones que se ha desarrollado existen:

- **configRTC()**: Iniciar el bus I2C y fija una hora predefinida en caso de haber detectado una perdida en la alimentación de la batería.
- **reportTime()**: Devuelve la hora del instante en el que se solicita.

3.2.6 Librería sensores

Para poder obtener una lectura de temperatura y humedad por parte de los sensores, se ha utilizado dos librerías distintas:

- **DallasTemperature**: Se encarga de controlar los dos sensores DS18B20. Ambos están conectados a un bus llamado *1-Wire*, usando el *GPIO 8*. De esta librería, se están utilizando funciones como *begin()*, para inicializar el bus; *getAddress()*, para conocer la dirección de uno de los sensores; *is_connected()*, para averiguar sin un sensor se ha desconectado o no; *requestTemperatures()*, que solicita la temperatura a todos los sensores; finalmente, *getTempByIndex()*, reporta la temperatura en función del índice que se le pase como argumento, en este caso, como son solo dos sensores, bastará con pasar 0 o 1.

- **DHT:** Esta librería se usa para poder obtener los datos de los sensores AM2301, de tipo *DHT21*. Solo es necesario especificar el *GPIO* al cual está conectado cada sensor, en este caso, los *GPIO 9* y *46*. Se están utilizando las funciones: *begin()*, para inicializar un determinado sensor; *readTemperature()* y *readHumidity()* para obtener los valores de temperatura y humedad relativa respectivamente.

3.2.7 Actuadores

La implementación de los actuadores es crucial para poder controlar los relés y el regulador de potencia, dado que ellos son los encargados de activar y desactivar el sistema de calefacción e iluminación. También en esta parte se desarrolla la lógica del botón que enciende y apaga el brillo de la LCD.

En cuanto a los relés, su código es bastante sencillo. Consta de una función *configButtonGPIO()* para configurar los *GPIO 4* y *6* como salidas; una función *switch*, que activa o desactiva el estado del relé. Hay que destacar que los relés son activos a baja, por lo que, si se quiere encender un bombillo, habrá que activarlo pasando un *0* o *false* como argumento; finalmente, una función de control general que recibe como parámetros una hora de encendido y apagado para poder activar o desactivar un determinado bombillo.

En la siguiente figura se puede observar cómo se controla el bombillo UVB, obteniendo los datos de tiempo y actuando en función de él. Se notifica el estado del relé en la interfaz gráfica activando o desactivando un *switch* gráfico.

```

void controlUVBLight(const char* on_hour, const char* on_minute,
                    const char* off_hour, const char* off_minute) {
    DateTime now = rtc.now();
    int up_time = atoi(on_hour) * 100 + atoi(on_minute);
    int current_time = (int)now.hour() * 100 + (int)now.minute();
    int down_time = atoi(off_hour) * 100 + atoi(off_minute);

    if( up_time < current_time && current_time < down_time) {
        switchUVBRelay(false);
        lv_obj_add_state(ui_uvbswitch, LV_STATE_CHECKED);
    } else {
        switchUVBRelay(true);
        lv_obj_clear_state(ui_uvbswitch, LV_STATE_CHECKED);
    }
}

```

Figura 32: Función del control del bombillo UVB en el fichero *actuators.cpp*

El regulador de potencia se apoya en las funciones desarrolladas por *Espressif*, los desarrolladores del ESP32. La funciones en cuestión son las llamadas *ledc()*, que viene de “*led control*” y generan señales *PWM*, por lo que pueden ser utilizadas en cualquier dispositivo que requiera de dicha señal.

Para poder controlar dicho regulador, se ha realizado lo siguiente:

1. Se ha creado una función que configura el *GPIO 7* para el regulador y además se hace un *setup* de dicho pin. En dicho *setup*, se establece el canal *PWM* que será utilizado, se define la frecuencia con la que trabajará la

señal y finalmente, la resolución de la señal.

```
void configDimmerGPIO(void) {  
    ledcSetup(1,60, 10);  
    ledcAttachPin(DIMMER_GPIO, 1);  
}
```

Figura 33: Función inicialización del dimmer en el fichero `actuators.cpp`

2. Se ha creado una función que fija el valor de la señal *PWM*, que para este sistema va de 0 a 1023, al haber definido una resolución de *10 bits*, como se puede ver en la *Figuria 1.30*.

```
void setPWM(const int pwm_value) {  
    ledcWrite(1, pwm_value);  
}
```

Figura 34: Función que fija el valor *PWM* en el fichero `actuators.cpp`

La gestión del *PWM* fue un poco problemática. El ESP32 dispone de 8 canales, de 0 a 7, por lo que inicialmente se configuró el canal 7 para controlar el *dimmer*. Esto no funcionó porque el canal 7 ya estaba ocupado por la pantalla, la cual lo necesita para poder definir su nivel de brillo.

Otro aspecto que dio problemas fue la propia implementación que hizo *Espressif* sobre las funciones *ledc*. Ocurrió que durante la fase de desarrollo no se podía utilizar una resolución de 8 bits para una frecuencia de 60 hz, que es la frecuencia a la que trabaja el bombillo UVA. Cuando se procedió a iniciar el microcontrolador con la resolución de 8 bits, este comunicaba por la interfaz serial que el módulo *PWM* no podía ser iniciado con una resolución de 8 bits y una frecuencia tan baja, por lo que había que reajustarla. La solución fue encontrada en la propia documentación^[40], donde se comenta que, para frecuencias más bajas, se debe utilizar valores de resolución más altos y viceversa. En este caso, la resolución que funcionó fue la de *10 bits*, lo que permite especificar *duty-cycles* o valores de trabajo de entre 0 y 1023, siendo este último el valor que representa la mayor potencia.

Por último, en lo referente al botón, se definió dos funciones para controlar cuando se activa y desactiva el brillo de la pantalla: la primera llamada `configButtonGPIO()`, que configura el *GPIO 16* para poder ser utilizado por el botón; la segunda, una función que lee el estado de botón y activa o desactiva el brillo en función de si este ha sido pulsado o no.


```

static void updateScreenBrightnessTask(void *args) {
    int last_state = HIGH;
    int current_state;
    while (1) {
        current_state = digitalRead(BUTTON_GPIO);
        if (last_state == HIGH && current_state == LOW) {
            setDisplayBrightness(0);
            last_state = LOW;
        } else if (last_state == LOW && current_state == LOW) {
            setDisplayBrightness(255);
            last_state = HIGH;
        }
        vTaskDelay(140/portTICK_RATE_MS);
    }
}

```

Figura 35: Función que controla el brillo de la pantalla en el fichero *task.cpp*

3.2.8 Librería WiFi

Como bien indica el nombre de la librería, esta es necesaria para poder gestionar las comunicaciones mediante este medio. Gran parte de su implementación fue explicada en la sección de FreeRTOS, al estar gestionada por un *timer*.

Además de lo explicado en ese apartado, hay que añadir que esta librería se está apoyando también en el uso de la memoria *flash* para poder guardar las credenciales de las redes a las cuales se ha podido conectar correctamente.

```

bool tryWiFi(const char *ssid, const char *password) {
    const ulong start_time = millis();
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED && (millis() - start_time) < wifi_timeout) {
        vTaskDelay(250 / portTICK_RATE_MS);
        Serial.print(".");
    }
    if (WiFi.status() == WL_CONNECTED) {
        Serial.print("\nCONECTADO");
        writeSDFlash(ssid);
        writePasswordFlash(password);
        return true;
    } else {
        Serial.print("\nNO CONECTADO");
        return false;
    }
}

```

Figura 36: Función que prueba una red en el fichero *networking.cpp*

Para finalizar este apartado, hay que mencionar que para que este dispositivo pueda iniciar una comunicación WiFi, no basta solo con usar el *timer* dado que este solo se encarga de realizar la reconexión. Para inicializar una conexión se hace uso de la función *connectWiFi()*, la cual es llamada en el *setup()* del fichero *main.cpp*.

```

void connectWiFi(void) {
  WiFi.disconnect();
  WiFi.mode(WIFI_STA);
  Serial.println("Número de redes disponibles: ");
  Serial.print(WiFi.scanNetworks());
  WiFi.begin(readSSIDFlash().c_str(), readPasswordFlash().c_str());
  while (WiFi.status() != WL_CONNECTED) {
    delay(100);
    Serial.print('.');
  }
  Serial.println("Conexión establecida.");
  Serial.println("IP:");
  Serial.println(WiFi.localIP());
}

```

Figura 37: Función que conecta a una red WiFi, llamada en el fichero *main.cpp*

3.2.9 Librería AsyncMQTTClient

Por último, la librería **AsyncMQTTClient**[\[22\]](#) hace que este microcontrolador se pueda considerar un dispositivo IoT dado que brinda las herramientas necesarias para poder trabajar con el protocolo MQTT.

Para poder empezar a utilizar dicho protocolo en el microcontrolador, es necesario primero realizar una configuración e inicialización previa, que en este caso, se realiza en la función *setup()* dentro del fichero *main.cpp* a través de la función *configMQTT()*.

```

void configMQTT(void) {
  mqtt_reconnect_timer = xTimerCreate("MQTTTimer",
                                      pdMS_TO_TICKS(2000),
                                      pdFALSE,
                                      (void*)0,
                                      reinterpret_cast<TimerCallbackFunction_t>(connectMQTT)
                                      );
  wifi_reconnect_timer = xTimerCreate("WIFITimer",
                                      pdMS_TO_TICKS(5000),
                                      pdFALSE,
                                      (void*)0,
                                      reinterpret_cast<TimerCallbackFunction_t>(wifiReconnect)
                                      );
  mqtt_client.onConnect(onMQTTConnect);
  mqtt_client.onDisconnect(onMQTTDisconnect);
  mqtt_client.onSubscribe(onMQTTSubscribe);
  mqtt_client.onUnsubscribe(onMQTTUnsubscribe);
  mqtt_client.onMessage(onMQTTReceived);
  mqtt_client.onPublish(onMQTTPublish);
  mqtt_client.setServer(MQTT_SERVER, 30000);
}

```

Figura 38: Función definida en el fichero *networking.cpp*

Como se ha podido ver en la figura anterior, se inicializa ambos *timers* (WiFi y MQTT) y se define los procedimientos que debe seguir el microcontrolador cuando se presenta un evento de conexión, suscripción, publicación, etc. además de los parámetros del servidor como la IP y el puerto del bróker.

Para tener un idea de cómo se comportan estos eventos, se muestra a continuación el de *mqtt_client.onSubscribe()*, el cual llama a una función que se encarga de publicar el tópic *iot-esp32/tfg* y definir el nivel de *QoS 0* en el bróker.

```
void subscribeMQTT(void) {
    uint16_t packetIdSub = mqtt_client.subscribe("iot-esp32/tfg", 0);
}
```

Figura 39: Función que suscribe el tópico de MQTT en el fichero `networking.cpp`

En cuanto a los niveles de *QoS* que se puede utilizar, tanto esta librería como MQTT dispone de tres niveles:

- **QoS 0:** Envía como mucho un mensaje, pudiendo no llegar al bróker.
- **QoS 1:** Envía al menos un mensaje hasta que se garantice la entrega. Se puede producir que el bróker reciba el mismo mensaje varias veces.
- **QoS 2:** Garantiza que se envíe solo un mensaje.

Cuando se realiza una publicación de datos al *bróker*, realmente lo que se está transmitiendo es un *json* con los campos de *identificador:valor-sensor*. Esta implementación fue posible gracias a la librería *ArduinoJson*[23].

La librería *AsyncMQTTClient* también ofrece características de seguridad como autenticación por usuario y contraseña, además de poder utilizar encriptación *TLS/SSL*. Sin embargo, por cuestiones de tiempo, no se ha podido implementar dichas características en el sistema. En la sección de líneas futuras de la memoria, se hablará mejor sobre este apartado.

Finalmente, al igual que la comunicación WiFi, esta librería está controlada por otro *timer*. De hecho, este *timer* depende de si se ha podido establecer una comunicación WiFi, es decir, si existe una conexión, el *timer* que gestiona MQTT estará en funcionamiento. Por el contrario, si no existe una conexión WiFi, se detendrá el *timer* de MQTT para dar paso al *timer* del WiFi e intentar recobrar la conexión lo antes posible.

3.3 Configuración del servidor

Una vez mostrados los aspectos más importantes en cuanto al hardware y software por parte del microcontrolador, es hora de examinar la infraestructura que constituye la parte del servidor.

3.3.1 Servidor, Dominio y DNS

Para poder poner a funcionar todas las aplicaciones web, es necesario disponer de un servidor. El servidor para este proyecto ha sido contratado al proveedor *Hetzner*[42]. Es un *VPS* que consta de las siguientes características:

- SO: Debian 11
- CPU: Dos vCPU x86
- RAM: 4GB

- Almacenamiento: 40 GB SSD
- Tráfico: 20 TB
- Firewall: Puertos 30000(MQTT), 80(HTTP) y 443(HTTPS).

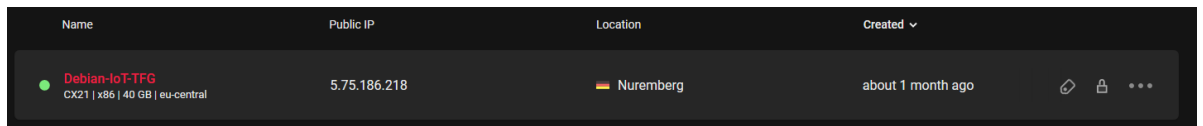


Figura 40: Visualización del servidor en la consola de Hetzner.

Para poder acceder fácilmente al servidor se ha contratado el dominio *tfg-ull.cloud*, al tener un precio bastante económico durante el primer año.

Para finalizar esta sección, se ha utilizado los servidores DNS de *Cloudflare* para gestionar el acceso a las diferentes aplicaciones de las que este sistema está compuesto.

Gestión de DNS para **tfg-ull.cloud**

Permite revisar, agregar y editar registros de DNS. Las ediciones entrarán en vigor una vez guardadas.

Importar y exportar ▼ ⚙️ Configuración de la pantalla del Panel de control

Buscar registros DNS

▼ Agregar filtro Buscar ➕ Agregar registro

Tipo ▲	Nombre	Contenido	Estado de proxy	TTL	Acciones
A	tfg-ull.cloud	5.75.186.218	🚀 Redirigido por proxy	Automático	Editar ▶
CAA	tfg-ull.cloud	0 issuewild letsencrypt.org	Solo DNS	Automático	Editar ▶
CAA	tfg-ull.cloud	0 issue digicert.com	Solo DNS	Automático	Editar ▶
CAA	tfg-ull.cloud	0 issuewild digicert.com	Solo DNS	Automático	Editar ▶
CAA	tfg-ull.cloud	0 issue globalsign.com	Solo DNS	Automático	Editar ▶
CAA	tfg-ull.cloud	0 issue letsencrypt.org	Solo DNS	Automático	Editar ▶
CAA	tfg-ull.cloud	0 issuewild globalsign.com	Solo DNS	Automático	Editar ▶
CAA	tfg-ull.cloud	0 issuewild comodoca.com	Solo DNS	Automático	Editar ▶
CAA	tfg-ull.cloud	0 issue comodoca.com	Solo DNS	Automático	Editar ▶
CNAME	grafana	tfg-ull.cloud	🚀 Redirigido por proxy	Automático	Editar ▶
CNAME	influx	tfg-ull.cloud	🚀 Redirigido por proxy	Automático	Editar ▶
CNAME	nodered	tfg-ull.cloud	🚀 Redirigido por proxy	Automático	Editar ▶
CNAME	portainer	tfg-ull.cloud	🚀 Redirigido por proxy	Automático	Editar ▶
CNAME	traefik	tfg-ull.cloud	🚀 Redirigido por proxy	Automático	Editar ▶

Figura 41: Configuración del DNS dentro de Cloudflare.

3.3.2 IoT Stack – Docker

El principal motivo de utilizar Docker se debe a la flexibilidad que aporta para probar nuevas aplicaciones y su configuración. Otro factor a tener en cuenta es que cuando este proyecto esté finalizado, Docker hace que migrar a otro servidor sea bastante sencillo, ya que, al estar escritos los ficheros de configuración, se puede volver a desplegar el *stack* de aplicaciones ajustando solo unos pocos valores.

En esta sección solo se mostrará los ficheros de configuración. La visualización de las interfaces web se mostrará en el *Capítulo 4 Pruebas realizadas*.

3.3.2.1 Traefik

Traefik es un proyecto con múltiples herramientas, pero para este sistema, solo se está haciendo uso de la funcionalidad *reverse-proxy*. Esta funcionalidad es muy interesante de implementar en el *stack* dado que simplifica la configuración de los accesos web al resto de contenedores. Esto es debido a que *Traefik* está escuchando en todo momento las peticiones realizadas por los puertos *80/443* y las dirige en función del subdominio solicitado. Esto quiere decir que, si en el navegador se escribe <https://portainer.tfg-ull.cloud>, será el propio proxy quien se encargue de atender la petición y redirigirla al contenedor de *Portainer*.

Como se habrá podido observar, la *url* anterior hacía uso de *https*. Esto es otro de los motivos por los que se ha utilizado *Traefik* y se puede ver en la siguiente figura:

```
certificatesResolvers:
  cloudflare:
    acme:
      email: alu0101267760@ull.edu.es
      storage: acme.json
      dnsChallenge:
        provider: cloudflare
        #disablePropagationCheck: true
      resolvers:
        - "1.1.1.1:53"
        - "1.0.0.1:53"
```

Figura 42: Configuración SSL dentro del fichero de configuración *traefik.yml*.

Cuando se crea el contenedor de *Traefik* y se carga el fichero de configuración *traefik.yml*, *Traefik* es capaz de obtener un certificado *SSL* a través de *Cloudflare* y *LetsEncrypt*. Esto es importante ya que hace que todas las comunicaciones que se produzcan entre el navegador y *Traefik* tengan un cifrado. Además, al realizarlo de esta forma, no existe la necesidad de tener que añadir un certificado *SSL* a todos y cada uno de los contenedores, ya que como las comunicaciones entran por el *proxy* ya se ha aplicado el cifrado. Se hace necesario precisar que una vez sale la petición del *proxy* a otro contenedor esta comunicación no está cifrada, pero para este caso, no tiene importancia dado que todas estas comunicaciones se producen dentro del mismo servidor.

Finalmente, para que *Traefik* se pueda comunicar con el resto de los contenedores, y viceversa, es necesario definir una red en Docker. En este caso, se creó una red externa llamada *proxy-network*.

```
iot@Debian-IoT-TFG:~/IoT_Stack_TFG$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
f9ba7953ac11       bridge             bridge             local
c7ee97b8f437       host               host               local
6df821b0e7ed       none               null               local
e662fc42460e       proxy-network      bridge             local
```

Figura 43: Red externa de Docker *proxy-network*.

El resto de los detalles de su funcionamiento se podrá ver en el *Capítulo 4 Pruebas realizadas*. En el anexo de esta misma memoria, estarán disponibles todos los ficheros relacionados con la configuración de este contenedor.

3.3.2.2 Portainer

El motivo de haber integrado *Portainer* en el sistema se debe a que ofrece una interfaz web para controlar y gestionar todos los contenedores configurados en Docker.

```
portainer:
  image: portainer/portainer-ce
  container_name: portainer
  restart: unless-stopped
  security_opt:
    - no-new-privileges:true
  networks:
    - proxy-network
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock:ro
    - ./portainer/data:/data
    - /etc/localtime:/etc/localtime:ro
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.portainer.entrypoints=http"
    - "traefik.http.routers.portainer.rule=Host('portainer.tfg-ull.cloud')"
    - "traefik.http.middlewares.portainer-https-redirect.redirectscheme.scheme=https"
    - "traefik.http.routers.portainer.middlewares=portainer-https-redirect"
    - "traefik.http.routers.portainer-secure.entrypoints=https"
    - "traefik.http.routers.portainer-secure.rule=Host('portainer.tfg-ull.cloud')"
    - "traefik.http.routers.portainer-secure.tls=true"
    - "traefik.http.routers.portainer-secure.service=portainer"
    - "traefik.http.services.portainer.loadbalancer.server.port=9000"
    - "traefik.docker.network=proxy-network"
```

Figura 44: Sección de Portainer dentro del *docker-copose.yml*.

Entre los aspectos más relevantes de la configuración destacan: montar un volumen con el *docker.sock*, lo cual permite a *Portainer* gestionar los contenedores de Docker; la configuración de las *labels*, que configuran la integración con *Traefik*; montar el volumen */data*, para que *Portainer* pueda mantener su configuración a pesar de que el contenedor sea parado; finalmente, añadir la red *proxy-network* para comunicarse con *Traefik*.

El resto de los detalles de su funcionamiento se podrá ver en el *Capítulo 4 Pruebas realizadas*. En el anexo de esta misma memoria, estarán disponible todos los ficheros relacionados con la configuración de este contenedor.

3.3.2.3 Mosquitto

Mosquitto es el bróker seleccionado para gestionar las comunicaciones con el microcontrolador a través del protocolo MQTT.

```

mosquitto:
  image: eclipse-mosquitto
  networks:
    - iot-network
  ports:
    - 30000:1883
    - 30001:9001
  volumes:
    - ./mosquitto/config:/mosquitto/config
    - ./mosquitto/data:/mosquitto/data
    - ./mosquitto/log:/mosquitto/log

```

Figura 45: Sección de Mosquitto dentro del `docker-copose.yml`.

Entre los aspectos más relevantes de la configuración destacan: el uso de una red interna llamada *iot-network*, que se usa para establecer las comunicaciones con *NodeRED*; el montaje de los volúmenes para poder conservar los ficheros de configuración, datos y *logs* en caso de que el contenedor sea parado. *Mosquitto* de momento no está siendo gestionado por *Traefik*.

El resto de los detalles de su funcionamiento se podrá ver en el *Capítulo 4 Pruebas realizadas*. En el anexo de esta misma memoria, estarán disponible todos los ficheros relacionados con la configuración de este contenedor.

3.3.2.4 NodeRED

NodeRED es la aplicación encargada de gestionar las comunicaciones entre *Mosquitto* y la base de datos *Influx*. Todo este manejo de datos se realiza de forma gráfica a través de una interfaz web.

```

nodered:
  image: nodered/node-red
  container_name: nodered
  security_opt:
    - no-new-privileges:true
  networks:
    - proxy-network
    - iot-network
  volumes:
    - ./nodered/data:/data
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.nodered.entrypoints=http"
    - "traefik.http.routers.nodered.rule=Host('nodered.tfg-ull.cloud')"
    - "traefik.http.middlewares.nodered-https-redirect.redirectscheme.scheme=https"
    - "traefik.http.routers.nodered.middlewares=nodered-https-redirect"
    - "traefik.http.routers.nodered-secure.entrypoints=https"
    - "traefik.http.routers.nodered-secure.rule=Host('nodered.tfg-ull.cloud')"
    - "traefik.http.routers.nodered-secure.tls=true"
    - "traefik.http.routers.nodered-secure.service=nodered"
    - "traefik.http.services.nodered.loadbalancer.server.port=1880"
    - "traefik.docker.network=proxy-network"

```

Figura 46: Sección de NodeRED dentro del `docker-copose.yml`.

Entre los aspectos más relevantes de la configuración destacan: el volumen `./nodered/data` para conservar la configuración; las redes *proxy-network* e *iot-network*, para poder comunicarse con *Traefik* y el resto de los contenedores; las *labels* que configuran el acceso web a través del proxy.

El resto de los detalles de su funcionamiento se podrá ver en el *Capítulo 4 Pruebas*

realizadas. En el anexo de esta misma memoria, estarán disponible todos los ficheros relacionados con la configuración de este contenedor.

3.3.2.5 InfluxDB

InfluxDB es una base datos especializada en datos de carácter temporal. Es la responsable de recoger los datos emitidos por *NodeRED* y almacenarlos para luego ser servidos a *Grafana*.

```
influx:
  image: influxdb
  container_name: influxdb
  security_opt:
    - no-new-privileges:true
  networks:
    - proxy-network
    - iot-network
  volumes:
    - ./influxdb/data:/var/lib/influxdb2
    - ./influxdb/config/etc/influxdb2
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.influx.entrypoints=http"
    - "traefik.http.routers.influx.rule=Host('influx.tfg-ull.cloud')"
    - "traefik.http.middlewares.influx-https-redirect.redirectscheme.scheme=https"
    - "traefik.http.routers.influx.middlewares=influx-https-redirect"
    - "traefik.http.routers.influx-secure.entrypoints=https"
    - "traefik.http.routers.influx-secure.rule=Host('influx.tfg-ull.cloud')"
    - "traefik.http.routers.influx-secure.tls=true"
    - "traefik.http.routers.influx-secure.service=influx"
    - "traefik.http.services.influx.loadbalancer.server.port=8086"
    - "traefik.docker.network=proxy-network"
```

Figura 47: Sección de *InfluxDB* dentro del *docker-copose.yml*.

Entre los aspectos más relevantes de la configuración destacan: los volúmenes para conservar la configuración y los datos; las redes *proxy-network* e *iot-network*, para poder comunicarse con *Traefik* y el resto de los contenedores; las *labels* que configuran el acceso web a través del proxy.

El resto de los detalles de su funcionamiento se podrá ver en el *Capítulo 4 Pruebas realizadas*. En el anexo de esta misma memoria, estarán disponible todos los ficheros relacionados con la configuración de este contenedor.

3.3.2.6 Grafana

Finalmente, *Grafana*, la aplicación responsable de recoger los datos almacenados en *InfluxDB* y dibujar una serie de gráficas en función a esos datos.

```
grafana:
  image: grafana/grafana
  container_name: grafana
  security_opt:
    - no-new-privileges:true
  user: "1000"
  networks:
    - proxy-network
    - iot-network
  volumes:
    - ./grafana/data:/var/lib/grafana
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.grafana.entrypoints=http"
    - "traefik.http.routers.grafana.rule=Host('grafana.tfg-ull.cloud')"
    - "traefik.http.middlewares.grafana-https-redirect.redirectscheme.scheme=https"
    - "traefik.http.routers.grafana.middlewares=grafana-https-redirect"
    - "traefik.http.routers.grafana-secure.entrypoints=https"
    - "traefik.http.routers.grafana-secure.rule=Host('grafana.tfg-ull.cloud')"
    - "traefik.http.routers.grafana-secure.tls=true"
    - "traefik.http.routers.grafana-secure.service=grafana"
    - "traefik.http.services.grafana.loadbalancer.server.port=3000"
    - "traefik.docker.network=proxy-network"
```

Figura 48: Sección de *InfluxDB* dentro del *docker-copose.yml*.

Entre los aspectos más relevantes de la configuración destacan: el volumen */grafana/data* para conservar la configuración y los datos; las redes *proxy-network* e *iot-network*, para poder comunicarse con *Traefik* y el resto de los contenedores; las *labels* que configuran el acceso web a través del proxy.

El resto de los detalles de su funcionamiento se podrá ver en el *Capítulo 4 Pruebas realizadas*. En el anexo de esta misma memoria, estarán disponible todos los ficheros relacionados con la configuración de este contenedor.

3.4 Diseño de la carcasa

Para almacenar toda la electrónica de una manera razonable, se diseñó una carcasa en Fusion360 y se imprimió utilizando una impresora 3D.

Dicha carcasa ha sido diseñada de la siguiente manera:

- Se diseñó de forma modular, con una base fija, un sistema de paneles fácilmente intercambiables y diversos agujeros para poder introducir con calor unas roscas que permiten atornillar y fijar los distintos componentes.
- El chasis sostiene todos los paneles, una *PCB* perforada que distribuye las conexiones de los sensores, el *buck-converter*, una clavija *jack* hembra para poder conectar la fuente de alimentación externa y unas clavijas RJ9 hembra, para conectar los sensores preparados con su variante macho.
- El panel frontal alberga la pantalla y el botón que apaga-enciende la pantalla.
- El panel derecho alberga los relés, el regulador de potencia y el reloj DS3231.
- El panel trasero permite instalar tres enchufes para poder conectar ahí los distintos bombillos, una entrada de un cable de 220V para alimentar a los tres enchufes y también tiene un recorte para dejar pasar las clavijas RJ9 hembra.
- El panel izquierdo tiene espacio para instalar un ventilador de 60x60 mm y permite atornillar una regleta para facilitar las conexiones de la línea de 220V.
- El panel superior cierra la caja, aportándole mayor estabilidad y protegiendo el acceso a los cables.

3.4.1 Fusion360

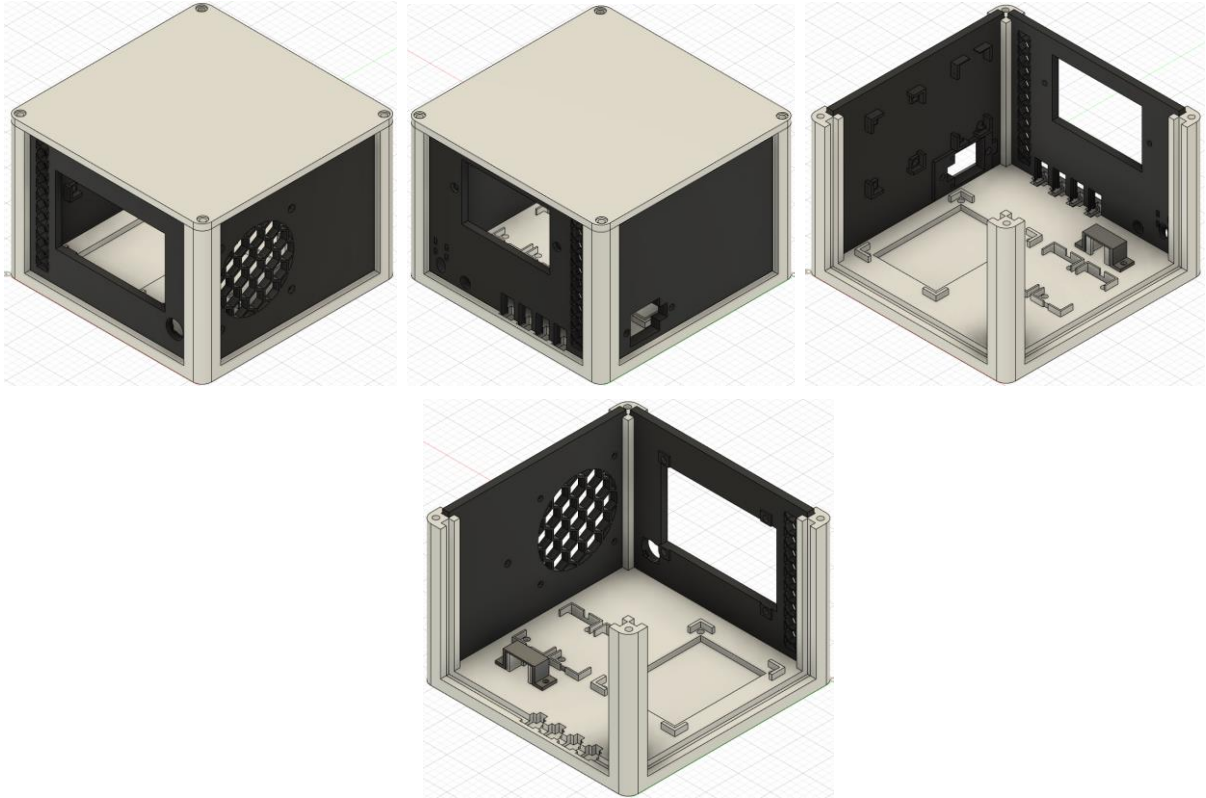


Figura 49: Visualización en distintos ángulos de la carcasa.

3.4.2 Impresión del modelo

La carcasa fue impresa con una boquilla de 0.4 mm, altura de capa de 0.3 y PLA-850 de la marca *Sakata*.

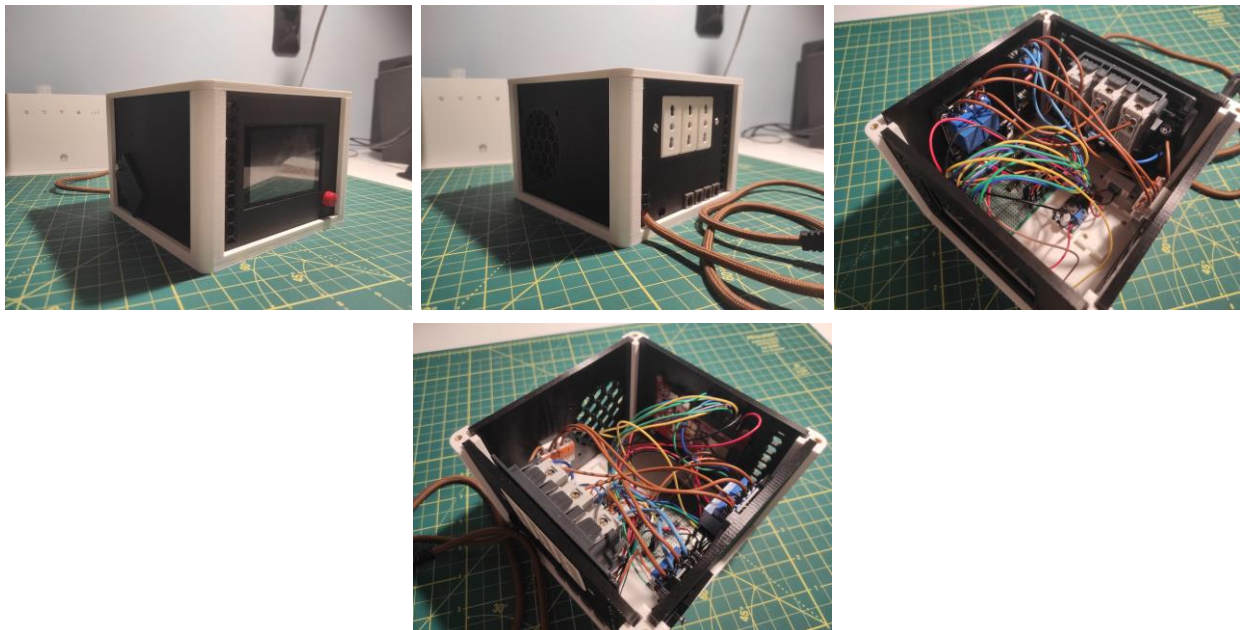


Figura 50: Visualización en distintos ángulos de la carcasa impresa.

Capítulo 4 Funcionamiento

Habiendo analizado los aspectos más importantes en cuanto a la implementación, es hora de ver finalmente como el termostato se comporta e interactúa con *Mosquitto*, el bróker MQTT alojado en el servidor.

4.1 Diagrama de funcionamiento

En el siguiente diagrama se puede observar de forma general cómo se realiza todas las interacciones que involucran el termostato, el bróker y el resto de las aplicaciones en el *stack*.

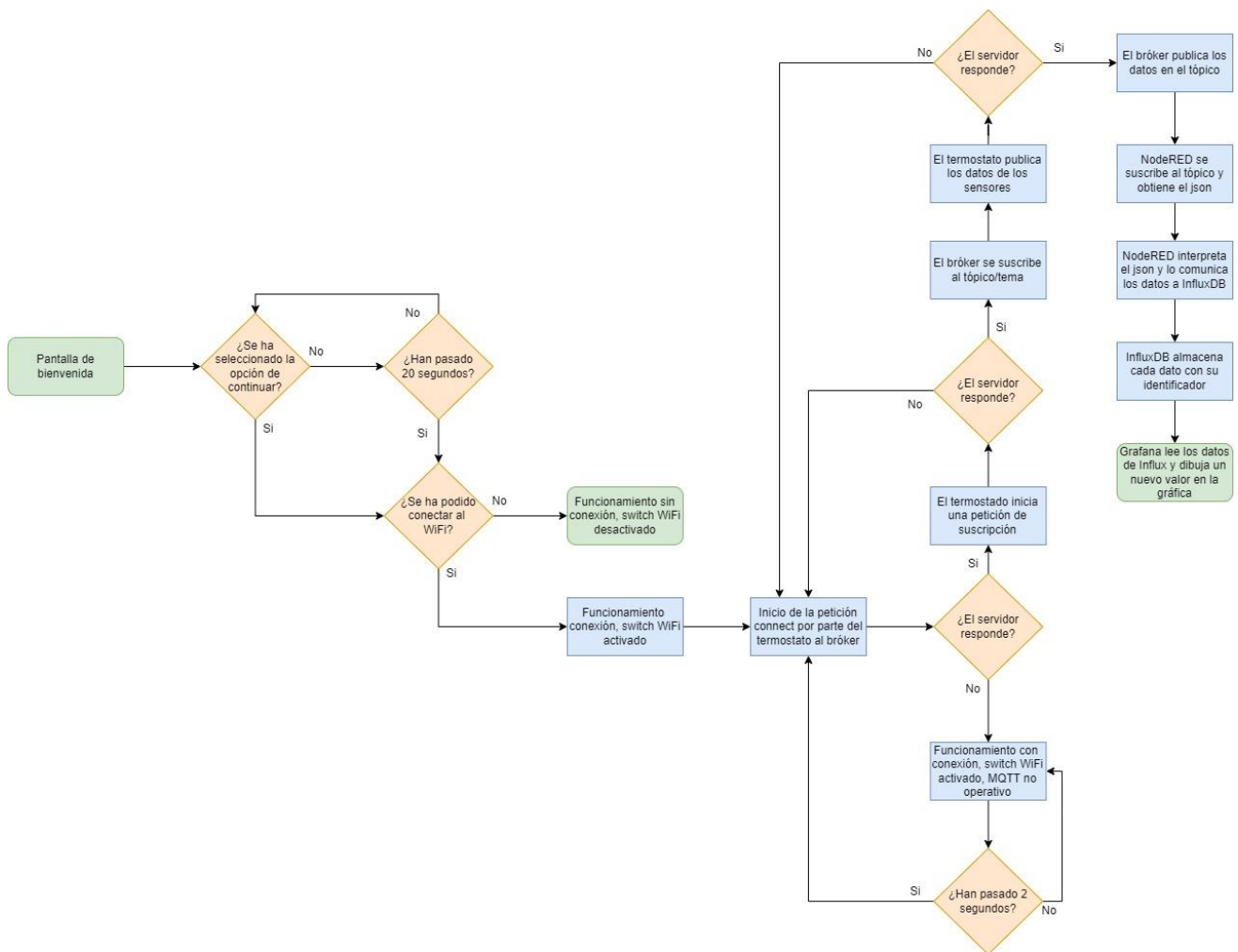
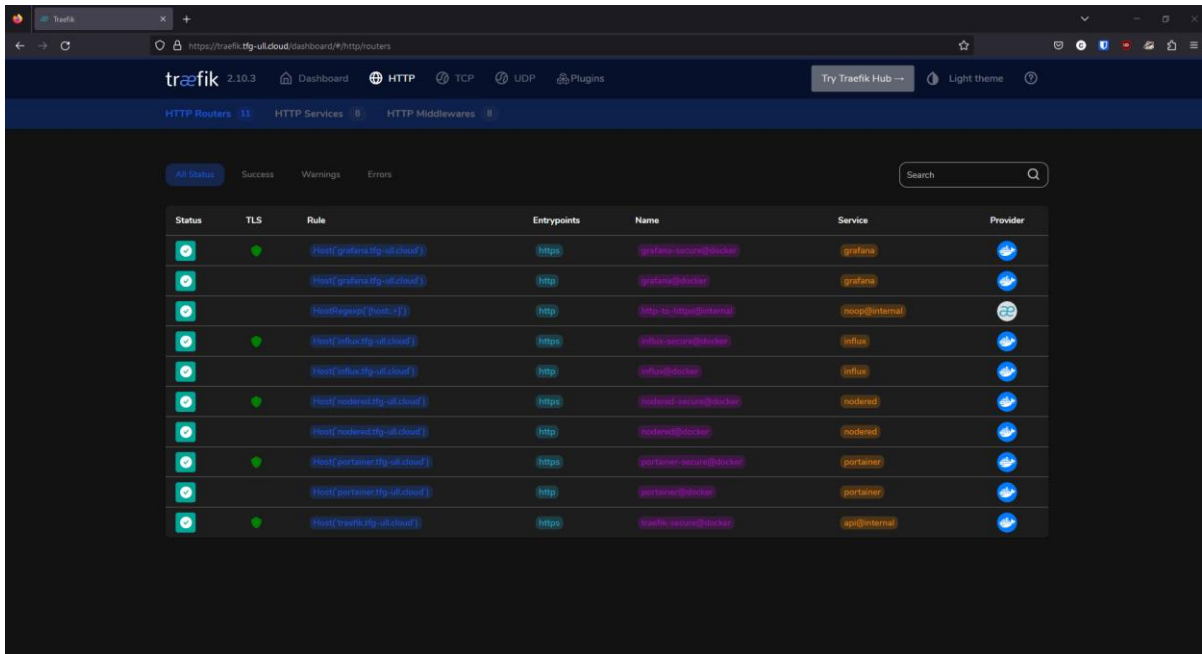


Figura 51: Visualización en distintos ángulos de la carcasa impresa.

4.2 Pruebas realizadas

4.2.1 Visualización de conexiones en Traefik

En la siguiente figura se puede observar cómo *Traefik* ha establecido las rutas *http* y *https* al resto de contenedores. Por defecto, hace una redirección *https*, protegidos con certificado *SSL*, como notifica el símbolo del escudo verde.



Status	TLS	Rule	Entrypoints	Name	Service	Provider
🟢	🟢	Host[grafana.tfg-ul.cloud]	https	grafana-secure@docker	grafana	🟢
🟢	🟢	Host[grafana.tfg-ul.cloud]	http	grafana@docker	grafana	🟢
🟢	🟢	Host[ncop.tfg-ul.cloud]	http	ncop-to-internal	ncop@internal	🟢
🟢	🟢	Host[infux.tfg-ul.cloud]	https	infux-secure@docker	infux	🟢
🟢	🟢	Host[infux.tfg-ul.cloud]	http	infux@docker	infux	🟢
🟢	🟢	Host[nodered.tfg-ul.cloud]	https	nodered-secure@docker	nodered	🟢
🟢	🟢	Host[nodered.tfg-ul.cloud]	http	nodered@docker	nodered	🟢
🟢	🟢	Host[portainer.tfg-ul.cloud]	https	portainer-secure@docker	portainer	🟢
🟢	🟢	Host[portainer.tfg-ul.cloud]	http	portainer@docker	portainer	🟢
🟢	🟢	Host[traefik.tfg-ul.cloud]	https	traefik-secure@docker	traefik@internal	🟢

Figura 52: Visualización de las rutas creadas en el dashboard de Traefik.

4.2.2 Conexión con el bróker *Mosquitto*

Conectando el termostato al serial podemos observar que es capaz de obtener una *IP*, conectarse al servidor MQTT con *QoS 0*, suscribir el tópic “*iot-esp32/tfg*” y mostrar uno de los valores de temperatura escrito en el *json*.

```
PROBLEMAS 7 SALIDA TERMINAL COMENTARIOS

Conectando al servidor MQTT...
.Conexión establecida.
IP:
192.168.16.5
Setup done
Conectado al servidor MQTT!
Suscrito a QoS 0, id del paquete: 1
Subscripción correcta!
  packetId: 1
  qos: 0
Topico recibido:
iot-esp32/tfg
: 24.70
Topico recibido:
iot-esp32/tfg
: 24.70
```

Figura 53: Visualización por serial del comportamiento del termostato.

Si entramos a ver los *logs* del contenedor de *Mosquitto* dentro de *Portainer*, podremos ver como las diferentes conexiones se van produciendo, tanto por parte del termostato (ESP32) como por parte de *NodeRED*, que podrá obtener los datos del tópic.

```
2023-07-11T11:20:31.143557129Z 1689074431: mosquitto version 2.0.15 starting
2023-07-11T11:20:31.143586732Z 1689074431: Config loaded from /mosquitto/config/mosquitto.conf.
2023-07-11T11:20:31.143591722Z 1689074431: Opening ipv4 listen socket on port 1883.
2023-07-11T11:20:31.143595986Z 1689074431: Opening ipv6 listen socket on port 1883.
2023-07-11T11:20:31.143601753Z 1689074431: mosquitto version 2.0.15 running
2023-07-11T11:20:31.860752683Z 1689074431: New connection from [REDACTED] on port 1883.
2023-07-11T11:20:31.866424951Z 1689074431: New client connected from [REDACTED] as esp32-[REDACTED] (p2, c1, k15).
2023-07-11T11:20:34.349142864Z 1689074434: New connection from 172.25.0.4:57606 on port 1883.
2023-07-11T11:20:34.349742245Z 1689074434: New client connected from 172.25.0.4:57606 as Node-RED (p2, c1, k60).
```

Figura 54: Visualización de los logs de Mosquitto en Portainer.

4.2.3 Visualización de datos en NodeRED

Como se mencionó con anterioridad, los diversos flujos de datos, transformaciones y conexiones se realizan a través de la propia interfaz gráfica web de *NodeRED*. En la siguiente figura se puede observar el primer nodo “ESP32-TFG”, que es la conexión con *Mosquitto*; un nodo “json”, el cual se encarga de recibir el mensaje del bróker; un nodo “debug”, que permite visualizar el contenido del *json* en el panel de la izquierda; seis nodos intermediarios que filtran cada campo del *json* para luego pasarlo a la base de datos *Influx*; finalmente, otros seis nodos que tienen la configuración de acceso a la base de datos para poder escribir la información de los sensores.

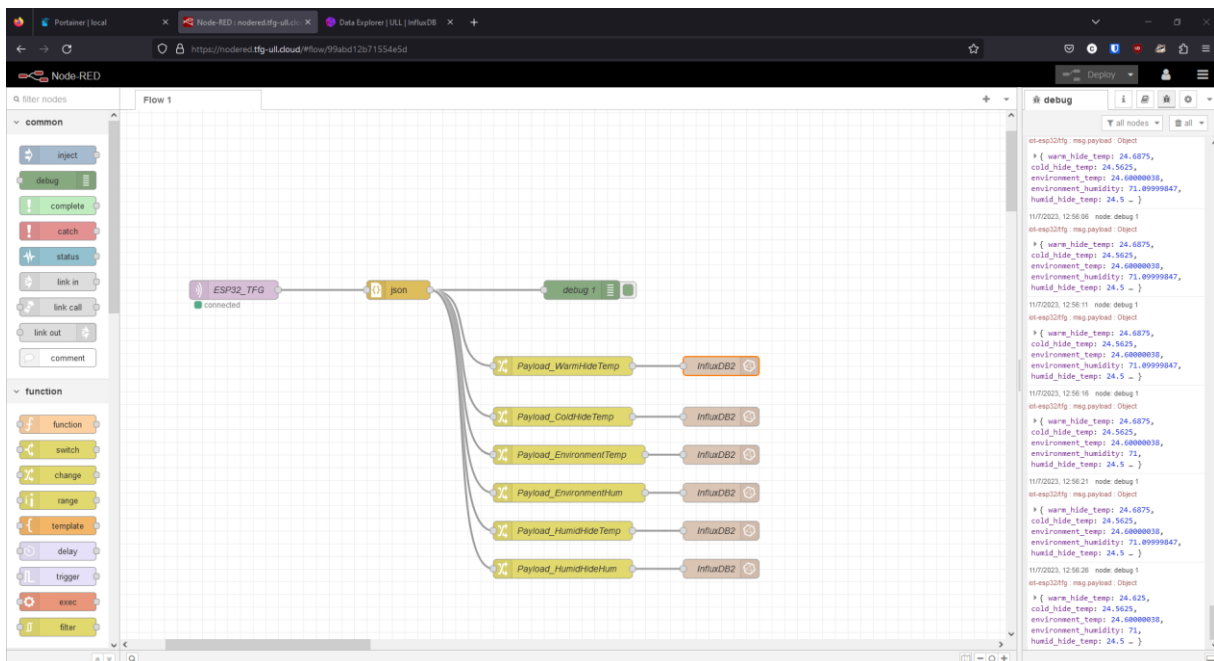


Figura 55: Visualización de los nodos en la interfaz de NodeRED.

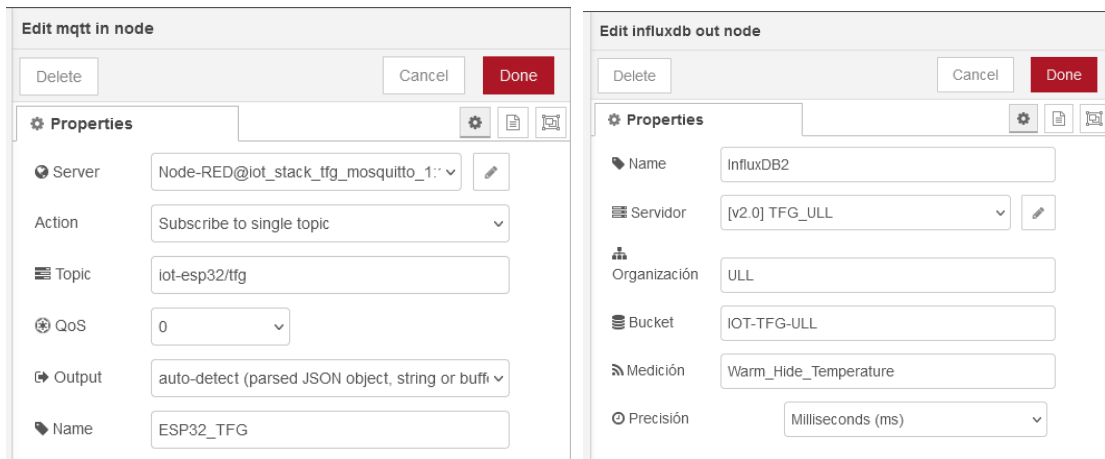


Figura 1.52: Configuración de los nodos de Mosquitto e Influx en NodeRED.

4.2.4 Visualización de conexiones en InfluxDB

InfluxDB a partir de la versión 2 empezó a ofrecer una interfaz web con varias características, entre ellas, visualización por gráficas de los datos, gestión, etc.

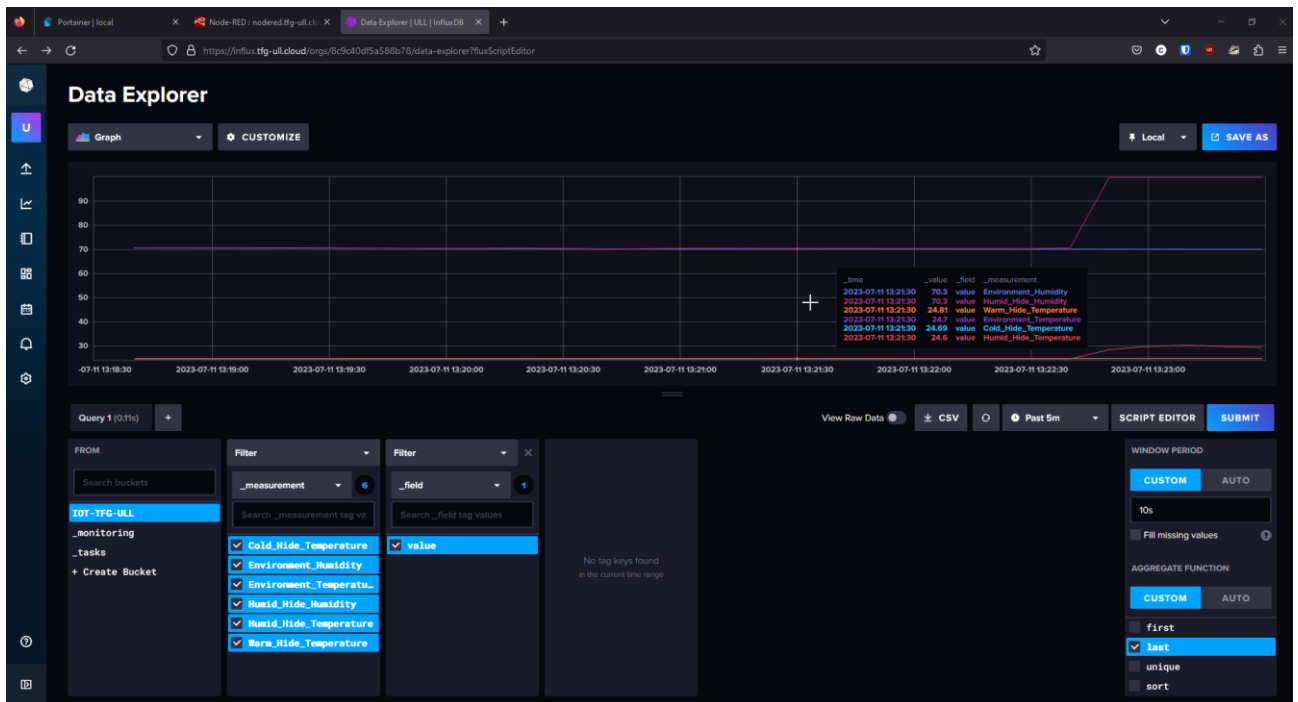


Figura 56: Datos recibidos desde NodeRED, visualizados en InfluxDB.

4.2.5 Visualización de conexiones en Grafana

Finalmente, en la siguiente figura se puede ver a *Grafana* en acción, recibiendo los datos de *InfluxDB*, configurado previamente como “*data source*”. *Grafana* cada cinco segundos realiza una consulta a la base de datos para obtener un valor y de ahí dibuja un punto en una determinada gráfica.

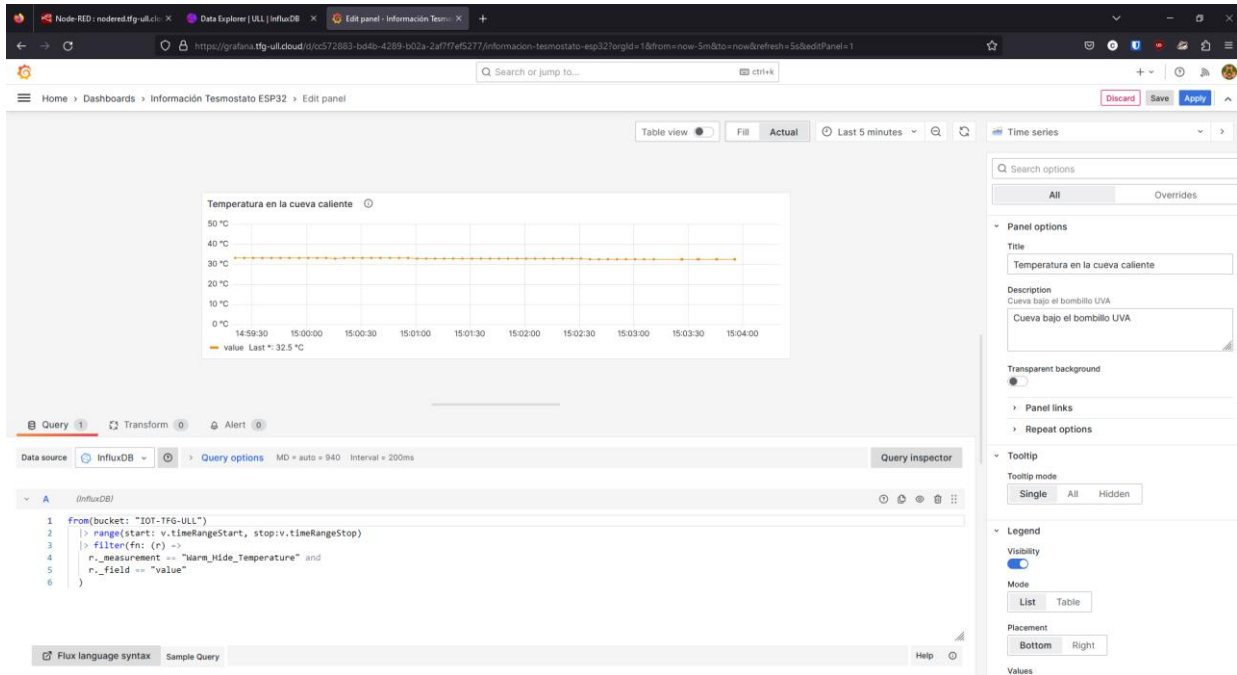


Figura 57: Configuración de la query a InfluxDB en Grafana.

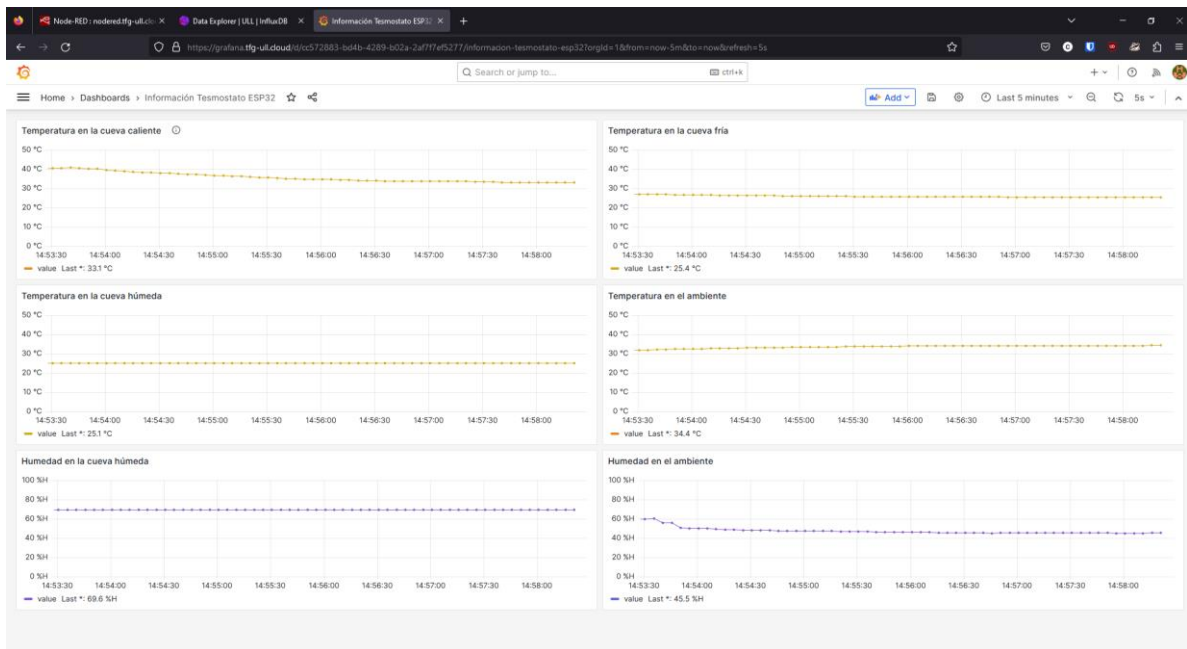


Figura 58: Visualización de todos los sensores en el Dashboard de Grafana.

Capítulo 5 Conclusiones y líneas futuras

Como se habrá podido observar a lo largo de toda la memoria, este proyecto ha puesto a prueba conocimientos de diferentes áreas, ya que se ha tenido que interactuar con electrónica, programación, redes, aplicaciones web, impresión 3D, etc. Todo esto ha derivado en que se ha podido analizar, estudiar y comprender como funciona la programación de un microcontrolador y el internet de las cosas.

En cuanto a los resultados obtenidos, se ha podido observar que este sistema es capaz de ofrecer: una interfaz gráfica en el termóstato, controlar un sistema de luces por tiempo y uno de ellos, además, por temperatura; conexión WiFi; conexión al servidor MQTT; publicación y almacenamiento de los datos en el servidor; y finalmente, generación de gráficas en tiempo real en base a los datos almacenados.

Pero este proyecto no puede quedarse aquí, debe seguir evolucionando. De querer desplegar este sistema en un entorno de producción, sería imperativo terminar de cifrar todas las comunicaciones entre el *ESP32* y el bróker *MQTT*, dado que, por motivos de tiempo, no se ha podido implementar el cifrado *SSL* entre estos dos puntos. Tanto la librería *AsyncMQTTClient* como *Mosquitto* soportan *TLS/SSL*, así que la posibilidad está ahí.

Además de ir solucionando los posibles *bugs* que vayan surgiendo a medida que se utilice este dispositivo, otra característica a añadir sería la de las alertas de *Grafana*, que probablemente será el siguiente paso por su sencillez.

En definitiva, a pesar de que sea el final de esta memoria, el ciclo de vida del termostato no ha hecho más que comenzar, con la esperanza de que algún día, la mayoría de su electrónica pueda ser integrada en una sola *PCB*.

Capítulo 6 Summary and Conclusions

As can be observed throughout the entire report, this project has tested knowledge from different areas, as it has required interaction with various electronics, programming, networks, web applications, 3D printing, and more. All of this has resulted in the ability to analyze, study, and understand how microcontroller programming and the Internet of Things function.

Regarding the obtained results, it has been observed that this system is capable of providing: a graphical interface on the thermostat, time-based control of a lighting system, and temperature-based control of another lighting system. It also includes WiFi connectivity, connection to the MQTT server, data publication and storage on the server, and finally, real-time graph generation based on the stored data.

However, this project cannot remain stagnant; it must continue to evolve. If this system were to be deployed in a production environment, it would be imperative to complete the encryption of all communications between the ESP32 and the MQTT broker. Due to time constraints, SSL encryption between these two points could not be implemented. Both the AsyncMQTTClient library and Mosquitto support TLS/SSL, so the possibility is there.

In addition to addressing any potential bugs that may arise during the usage of this device, another feature to consider adding is Grafana alerts, which will likely be the next step due to its ease of implementation.

In conclusion, although this report marks the end, the life cycle of the thermostat is just beginning, with the hope that one day, most of its electronics can be integrated into a single PCB.

Capítulo 7 Presupuesto

Ítem	Unidades	Precio Unitario	Precio total
MaTouch ESP32-S3 SPI	1	33,74 euros	33,74 euros
Relé Arduino de dos canales	1	1,36 euros	1,36 euros
Regulador de potencia por PWM	1	6.82 euros	6.82 euros
Conector hembra RJ9	50	2.78 euros	2.78 euros
Conector macho RJ9	50	7.87 euros	7.87 euros
Sensor AM2301	2	3,66 euros	7,32 euros
Tuercas de calor M3	100	5,31 euros	5,31 euros
Tornillos M3 x 6	100	1,35 euros	1,35 euros
Cables dupont	40	1,97 euros	1,97 euros
Fuente de alimentación 9V	1	3,11 euros	3,11 euros
Sensor DS18B20	2	1,12 euros	2,24 euros
PCB perforada 6x8cm	1	0.68 euros	0.68 euros
Conector Jack 5.5x2.1 hembra	5	1,15 euros	1,15 euros
RTC DS3231	1	1,78 euros	1,78 euros
Sakata PLA-850	1	18,90 euros	18,90 euros
Enchufe	3	23,34 euros	23,34 euros
Caja soporte para tres enchufes	1	1,47 euros	1,47 euros
Prolongador de cable con enchufe	1	7,99 euros	7,99 euros
Cables de 0.75mm, 10 m	1	7,79 euros	7,79 euros
Botón arduino	25	1,41 euros	1,41 euros
Servidor (meses)	2	6,26 euros	12,52 euros
Dominio <i>tfg-ull.cloud</i> (año)	1	1, 41 euros	1,41 euros
Mano de obra (horas)	200	20 euros	4000 euros
-	-	TOTAL	4.152,31 euros

Tabla 1 Presupuesto

Capítulo 8 Anexos

8.1 Código del termostato

- main.cpp

```
/**
 * @mainpage Documentación TFG: Dispositivo IoT a través del protocolo MQTT
 * @section intro_sec Introducción
 * En esta documentación se muestra como funciona el código de una aplicación \n
 * para un dispositivo IoT, concretamente, un termostato que controla las \n
 * condiciones climáticas en un terrario, usando una placa de desarrollo \n
 * que integra el microcontrolador ESP32-S3. \n
 * Documentación de la placa de desarrollo: \n
 * https://github.com/Makerfabs/Makerfabs-ESP32-S3-SPI-TFT-with-Touch
 */

#include "UI/mcu_ui.h"
#include "RTOS/tasks.h"

/**
 * @brief Fichero principal
 * @file main.cpp
 * @author Jesús Carmelo González Domínguez
 * @date 14-07-2023
 */

/**
 * @brief Encargada de llamar a las funciones que realizan la
 * configuración inicial del microcontrolador.
 * @note Configura el serial, todos los pines GPIO utilizados, inicia la memoria flash,
 * inicia la GUI, crea las tareas del sistema e inicia la comunicación WiFi-MQTT.
 */
void setup(void) {
    Serial.begin(115200);

    configButtonGPIO();
    configRelayGPIO();
    configDimmerGPIO();
    configSensorsGPIO();
    initFlashMemory();
    configLVGLUI();
    initUITask();
    drawStartWiFiMenuScreen();
    createTasks();

    WiFi.onEvent(wifiEvent);
    configMQTT();
    connectWiFi();

    Serial.println( "Setup done" );
}

/**
 * @brief Encargada de llamar a las funciones que realizan la
 * configuración inicial del microcontrolador.
 * @note vTaskDelete(NULL), borra del task handler la función loop().
 */
```

```

*/
void loop() {
    vTaskDelete(NULL);
}

```

- mcu_ui.h

```

/**
 * @brief Fichero que contiene todas las definiciones de las funciones creadas por
 * el autor para interactuar con la librería LVGL.
 * @file mcu_ui.h
 * @author Jesús Carmelo González Domínguez
 * @date 14-07-2023
 */

#ifndef MCU_UI_GUARD
#define MCU_UI_GUARD

#include "lvgl.h"
#include "ui.h"
#include "Devices/ft6236.h"
#include "Devices/display.h"
#include "Devices/networking.h"
#include "vector"

/* Variables */
// Menu management variables
/**
 * @brief Variable que se utiliza para saber si se ha mostrado en
 * pantalla la ventana que otorga la opción de borrar la configuración
 * WiFi en la memoria flash.
 */
extern bool flash_message_box_drawn;
/**
 * @brief Variable que se utiliza para saber si se ha solicitado la ventana
 * de borrado de la configuración WiFi en la memoria flash.
 */
extern bool flash_message_box_requested;
/**
 * @brief Variable que se utiliza para saber si se ha solicitado la ventana
 * de introducción de credenciales WiFi.
 */
extern bool credentials_message_box_drawn;
/**
 * @brief Variable que se utiliza para saber si se mostrado la ventana
 * de introducción de credenciales WiFi.
 */
extern bool credentials_message_box_requested;
/**
 * @brief Variable que contiene el número de redes WiFi encontradas.
 */
extern int current_found_networks;
/**
 * @brief Variable que contiene el último número de redes WiFi encontradas,
 * con el fin de evitar dibujar elementos ya existentes, lo cual hacía que el
 * ESP32 se reiniciara.
 */
extern int last_found_networks;
/**
 * @brief Variable que contiene el ssid de la red wifi seleccionada en el menu.
 */
extern const char *selected_ssid;
/**
 * @brief Variable que contiene la contraseña WiFi escrita en el menú.
 */
extern const char *selected_password;
// Screen variables
/**
 * @brief Variable que contiene el ancho de la pantalla en píxeles.
 */
extern const uint32_t screen_width;
/**
 * @brief Variable que contiene el alto de la pantalla en píxeles.
 */

```

```

extern const uint32_t screen_height;
/**
 * @brief Búffer que se utiliza para escribir la información en la pantalla.
 */
extern Lv_disp_draw_buf_t draw_display_buffer;
/**
 * @brief Búffer que almacena la información de la pantalla .
 */
extern Lv_color_t display_buffer;
/**
 * @brief Instancia de la pantalla LCD.
 */
extern Display display;
/**
 * @brief Instancia del panel táctil.
 */
extern FT6236 ts;

/* Functions */
/**
 * @brief Se encarga de dibujar la pantalla de bienvenida y permite configurar
 * la red WiFi. Si el usuario no introduce ninguna opción en 20 segundos, el
 * sistema proseguirá con la configuración guardada previamente.
 */
void drawStartWiFiMenuScreen(void);
/**
 * @brief Se encarga de dibujar la información en la pantalla con la información
 * de los buffers.
 * @param disp Búffer de la pantalla
 * @param area Área de la pantalla
 * @param color_p Información de color de la pantalla. 5 para rojo, 6 para verde y
 * 5 para azul.
 */
void drawDisplay(Lv_disp_drv_t *disp, const Lv_area_t *area, Lv_color_t *color_p);
/**
 * @brief Se encarga de leer los puntos de presión en el panel táctil.
 * @param indev_driver Instancia del driver del panel táctil
 * @param data Información relacionada a las coordenadas x-y del panel.
 */
void readTouchpad(Lv_indev_drv_t *indev_driver, Lv_indev_data_t *data);
/**
 * @brief Se encarga iniciar y configurar el driver de la LCD y el panel táctil,
 * conectando estos a su vez con la librería LVGL.
 */
void configLVGLUI(void);
/**
 * @brief Fija el brillo que utilizará la pantalla para el resto de la sesión.
 * @param value Valor de brillo que va de 0 a 255.
 */
void setDisplayBrightness(const uint8_t value);
/**
 * @brief Muestra la ventana que permite al usuario borrar la configuración
 * WiFi de la memoria Flash.
 */
void drawCleanFlashMessageBox(void);
/**
 * @brief Lee un evento y cierra la ventana de gestión WiFi si se cumple.
 * @param e Tipo de evento registrado, es decir, si el botón ha sido pulsado o no.
 */
void uiEventCloseWiFiWindow( Lv_event_t *e);
/**
 * @brief Lee un evento y borra la configuración WiFi si se cumple.
 * @param e Tipo de evento registrado, es decir, si el botón sido pulsado o no.
 */
void uiEventCleanFlash( Lv_event_t *e);
/**
 * @brief Lee un evento y escribe la configuración WiFi en la flash si se cumple.
 * @param e Tipo de evento registrado, es decir, si el botón ha sido pulsado o no.
 */
void uiEventWriteWiFiCredentials( Lv_event_t * e);
/**
 * @brief Muestra la configuración WiFi en la pantalla de bienvenida.
 * @param data de tipo WifiScanData, es una matriz de strings, la cual

```

```

* contiene toda la información obtenida durante el escaneo de las redes WiFi.
*/
void drawWiFiMenu(WifiScanData data);

#endif

```

- mcu_ui.cpp

```

#include "UI/mcu_ui.h"
#include "UI/ui_helpers.h"

/* Variables */
// Menu management variables
bool flash_message_box_drawn = false;
bool flash_message_box_requested = false;
bool credentials_message_box_drawn = false;
bool credentials_message_box_requested = false;
int current_found_networks = 0;
int last_found_networks = 0;
const char *selected_ssid = "";
const char *selected_password = "";
// Screen variables
const uint32_t screen_width = 480;
const uint32_t screen_height = 320;
Lv_disp_draw_buf_t draw_buffer;
Lv_color_t buffer[screen_width * screen_height / 2];
Display display;
FT6236 ts = FT6236();

/* Functions */
void drawStartWiFiMenuScreen(void) {
    wifi_data = getWiFiSSIDs();
    String ssid = readSSIDFlash();
    lv_label_set_text(ui_loadingscreenssid, ssid.c_str());
    const unsigned long start_time = millis();
    while (lv_scr_act() == ui_loadingscreen && (millis() - start_time) < 60000) {
        drawWiFiMenu(wifi_data);
    }
    lv_obj_clean(ui_loadingscreen);
    _ui_screen_change( ui_screen1, LV_SCR_LOAD_ANIM_NONE, 0, 0);
}

void drawDisplay(Lv_disp_drv_t *disp, const Lv_area_t *area, Lv_color_t *color_p) {
    uint32_t w = ( area->x2 - area->x1 + 1 );
    uint32_t h = ( area->y2 - area->y1 + 1 );
    display.startWrite();
    display.setAddrWindow( area->x1, area->y1, w, h );
    display.writePixels((lgfx::rgb565_t *)&color_p->full, w * h);
    display.endWrite();
    lv_disp_flush_ready( disp );
}

void readTouchpad(Lv_indev_drv_t * indev_driver, Lv_indev_data_t * data) {
    if(ts.touched()){
        data->state = LV_INDEV_STATE_PR;
        TS_Point p = ts.getPoint();
        data->point.x = p.y;
        data->point.y = display.height() - p.x;
        Serial.printf("x-%d,y-%d\n", data->point.x, data->point.y);
    } else {
        data->state = LV_INDEV_STATE_REL;
    }
}

void configLVGLUI(void) {
    display.begin();
    //display.setRotation(3);
    display.setRotation(1);
    display.setBrightness(255);
    if(!ts.begin(5, SDA_FT6236, SCL_FT6236)){
        Serial.println("Unable to start the capacitive touch Screen.");
    }
}

```

```

}
lv_init();
lv_disp_draw_buf_init(&draw_buffer, buffer, NULL, screen_width * screen_height / 2);
/*Initialize the display*/
static lv_disp_drv_t disp_drv;
lv_disp_drv_init(&disp_drv);
/*Change the following line to your display resolution*/
disp_drv.hor_res = screen_width;
disp_drv.ver_res = screen_height;
disp_drv.flush_cb = drawDisplay;
disp_drv.draw_buf = &draw_buffer;
lv_disp_drv_register(&disp_drv);
/*Initialize the (dummy) input device driver*/
static lv_indev_drv_t indev_drv;
lv_indev_drv_init(&indev_drv);
indev_drv.type = LV_INDEV_TYPE_POINTER;
indev_drv.read_cb = readTouchpad;
lv_indev_drv_register(&indev_drv);
ui_init();
//lv_disp_set_rotation(lv_disp_get_default(), LV_DISP_ROT_180);
}

void setDisplayBrightness(const uint8_t value) {
    display.setBrightness(value);
}

// WiFi Start Screen Menu Management
void drawCleanFlashMessageBox(void) {
    const char text[] = { "Por favor, confirme el procedimiento." };
    static const char* options[] = {"Aceptar", "Rechazar", ""};
    if (ui_reportwifimessagebox != NULL) {
        if (!flash_message_box_drawn) {
            ui_reportwifimessagebox = lv_msgbox_create(NULL, "Borrar red actual", text, options, false);
            lv_obj_center(ui_reportwifimessagebox);
            flash_message_box_drawn = true;
        }
        if (lv_msgbox_get_active_btn(ui_reportwifimessagebox) == 0) {
            writeSSIDFlash("");
            writePasswordFlash("");
            flash_message_box_requested = false;
            flash_message_box_drawn = false;
            lv_msgbox_close(ui_reportwifimessagebox);
        } else if (lv_msgbox_get_active_btn(ui_reportwifimessagebox) == 1) {
            flash_message_box_requested = false;
            flash_message_box_drawn = false;
            lv_msgbox_close(ui_reportwifimessagebox);
        }
    }
}

void drawCredentialsMessageBox(void) {
    if (ui_reportwifimessagebox != NULL) {
        if (!credentials_message_box_drawn) {
            ui_wifikeyboard = lv_keyboard_create(ui_loadingscreen);
            vTaskDelay(50);
            ui_wifipasswordarea = lv_textarea_create(ui_loadingscreen);
            lv_obj_set_size(ui_wifipasswordarea, 480, 160);
            lv_textarea_set_placeholder_text(ui_wifipasswordarea, "Escriba la clave...");
            selected_password = lv_textarea_get_text(ui_wifipasswordarea);
            lv_keyboard_set_textarea(ui_wifikeyboard, ui_wifipasswordarea);
            credentials_message_box_drawn = true;
        }
        if (lv_keyboard_get_selected_btn(ui_wifikeyboard) == 39) {
            if (tryWiFi(selected_ssid, selected_password)) {
                Serial.print("\n CONEXION CORRECTA ");
                const char* message = "Las claves introducidas han sido correctas. Se han guardado en la memoria interna.";
                ui_reportwifimessagebox = lv_msgbox_create(NULL, "Claves correctas", message, NULL, true);
                lv_obj_center(ui_reportwifimessagebox);
                lv_label_set_text(ui_loadingscreenssid, selected_ssid);
            } else {

```

```

- Serial.print("\n CONEXION ERRONEA ");
- const char* message = "Las claves introducidas han sido incorrectas. Introduzcalas de nuevo.";
- ui_reportwifimessagebox = lv_msgbox_create(NULL, "Claves incorrectas", message, NULL, true);
- lv_obj_center(ui_reportwifimessagebox);
- }
- credentials_message_box_requested = false;
- credentials_message_box_drawn = false;
- lv_obj_del_async(ui_wifipasswordarea);
- lv_obj_del_async(ui_wifikeyboard);
- }
- }
- }
- // WiFi menu events
- void uiEventCloseWiFiWindow( Lv_event_t * e ) {
-     Lv_event_code_t event_code = lv_event_get_code(e);
-     Lv_obj_t * target = lv_event_get_target(e);
-     if (event_code == LV_EVENT_CLICKED) {
-         _ui_screen_change( ui_screen1, LV_SCR_LOAD_ANIM_NONE, 0, 0);
-     }
- }
- void uiEventCleanFlash( Lv_event_t * e ) {
-     Lv_event_code_t event_code = lv_event_get_code(e);
-     Lv_obj_t * target = lv_event_get_target(e);
-     if (event_code == LV_EVENT_CLICKED) {
-         flash_message_box_requested = true;
-         lv_label_set_text(ui_loadingscreenssid, "");
-     }
- }
- void uiEventWriteWiFiCredentials( Lv_event_t * e ) {
-     Lv_event_code_t event_code = lv_event_get_code(e); Lv_obj_t * target = lv_event_get_target(e);
-     if (event_code == LV_EVENT_CLICKED) {
-         selected_ssid = lv_list_get_btn_text(ui_wifilistoptions, target);
-         credentials_message_box_requested = true;
-     }
- }
- // WiFi menu
- void drawWiFiMenu(WifiScanData wifi_data) {
-     current_found_networks = wifi_data.size();
-     if (ui_wifilist != NULL) {
-         if (current_found_networks != last_found_networks) {
-             lv_obj_clean(ui_wifilist);
-             lv_list_add_text(ui_wifilist, "Opciones");
-             ui_wifilistoptions = lv_list_add_btn(ui_wifilist, LV_SYMBOL_TRASH, "Borrar red actual");
-             lv_obj_add_event_cb(ui_wifilistoptions, uiEventCleanFlash, LV_EVENT_ALL, NULL);
-             ui_wifilistoptions = lv_list_add_btn(ui_wifilist, LV_SYMBOL_CLOSE, "Continuar con los ajustes
- actuales");
-             lv_obj_add_event_cb(ui_wifilistoptions, uiEventCloseWiFiWindow, LV_EVENT_ALL, NULL);
-             lv_list_add_text(ui_wifilist, "Redes disponibles");
-             for (int i = 0; i < wifi_data.size(); i++) {
-                 ui_wifilistoptions = lv_list_add_btn(ui_wifilist, LV_SYMBOL_WIFI, wifi_data[i][0].c_str());
-                 lv_obj_add_event_cb(ui_wifilistoptions, uiEventWriteWiFiCredentials, LV_EVENT_ALL, NULL);
-             }
-             last_found_networks = current_found_networks;
-         }
-         if (flash_message_box_requested) {
-             drawCleanFlashMessageBox();
-         } else if (credentials_message_box_requested) {
-             drawCredentialsMessageBox();
-         }
-     }
- }
- }

```

```

- task.h
- /**
- * @brief Fichero que define las tareas que ejecuta el sistema.
- *
- * @file tasks.h

```



```

- * @author Jesús Carmelo González Domínguez
- * @date 14-07-2023
- */
-
- #ifndef TASKS_GUARD
- #define TASKS_GUARD
-
- #include "UI/mcu_ui.h"
- #include "Devices/sensor.h"
- #include "Devices/rtc.h"
- #include "Devices/actuators.h"
- #include "Devices/networking.h"
-
- // Handlers
- /**
- * @brief Controlador de la tarea que ejecuta la GUI.
- */
- extern TaskHandle_t lvglHandler;
- /**
- * @brief Controlador de la tarea que ejecuta el encendido/apagado de la pantalla.
- */
- extern TaskHandle_t updateBrightnessTaskHandler;
- /**
- * @brief Controlador de la tarea que escribe la información de los sensores
- * en la GUI.
- */
- extern TaskHandle_t writeSensorsDataUITaskHandler;
- /**
- * @brief Controlador de la tarea que escribe el tiempo reportado por el
- * reloj en la GUI.
- */
- extern TaskHandle_t runClockUITaskHandler;
- /**
- * @brief Controlador de la tarea que muestra el tiempo de actividad del
- * bombillo UVA en la GUI.
- */
- extern TaskHandle_t updateActiveTimeUVATaskHandler;
- /**
- * @brief Controlador de la tarea que muestra el tiempo de actividad del
- * bombillo UVB en la GUI.
- */
- extern TaskHandle_t updateActiveTimeUVBTaskHandler;
- /**
- * @brief Controlador de la tarea que muestra el tiempo de actividad del
- * bombillo Plants en la GUI.
- */
- extern TaskHandle_t updateActiveTimePlantsTaskHandler;
- /**
- * @brief Controlador de la tarea que fija la temperatura máxima a la que
- * puede llegar el bombillo UVA.
- */
- extern TaskHandle_t setMaxTemperatureUVATaskHandler;
- /**
- * @brief Controlador de la tarea controla el sistema de luces.
- */
- extern TaskHandle_t lightSystemTaskHandler;
- /**
- * @brief Controlador de la tarea que se encarga de publicar los datos de
- * los sensores a Mosquitto, el bróker MQTT.
- */
- extern TaskHandle_t publishMQTTBrokerTaskHandler;/**
-
- /* Functions */
- /**
- * @brief Crea la tarea encargada de lanzar el supervisor o handler de LVGL
- */
- void initUITask(void);
- /**
- * @brief Crea todas las tareas del sistema encargadas de leer la temperatura,
- * comunicar datos, actualizar el reloj de la pantalla, etc.

```

```

- */
- void createTasks(void);
- /**
-  * @brief Calcula cuanto tiempo estará activo un bombillo en función de las horas
-  * especificadas en los rollers de tiempo, con el fin de mostrar esa información
-  * en la GUI.
-  * @param hours_remaining Número de horas en las que el bombillo estará encendido
-  * @param minutes_remaining Número de minutos en las que el bombillo estará encendido
-  * @param start_hour Hora a la que se activará el bombillo
-  * @param start_minute Minuto en el que se activará el bombillo
-  * @param finish_hour Hora a la que se desactivará el bombillo
-  * @param finish_minute Minuto en el que se desactivará el bombillo
-  *
-  */
- void getTimeDifference(char* hours_remaining, char* minutes_remaining,
-                       const char* start_hour, const char* start_minute,
-                       const char* finish_hour, const char* finish_minute);
- /**
-  * @brief Obtiene la string de un roller que representa la hora para poder obtener
-  * por separado la hora y los minutos. Todos los parámetros son punteros de arrays de chars.
-  * @param roller_time Contiene la hora del roller.
-  * @param hour Especifica solo la parte de la hora obtenida del roller_time
-  * @param minutes Especifica solo la parte de los minutos obtenidos del roller time
-  *
-  */
- void getUIRollerTime(const char* roller_time, char* hour, char* minutes);
-
- // Tasks functions
- /**
-  * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
-  * que controla la GUI del dispositivo
-  * @param args Esta especificación hace posible que la función puede ser llamada
-  * por su definición a la hora de crear la tarea en freeRTOS.
-  *
-  */
- static void runUI (void* args);
- /**
-  * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
-  * que controla el brillo de la pantalla.
-  * @param args Esta especificación hace posible que la función puede ser llamada
-  * por su definición a la hora de crear la tarea en freeRTOS.
-  *
-  */
- static void updateScreenBrightnessTask(void* args);
- /**
-  * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
-  * que escribe los datos de los sensores en la pantalla.
-  * @param args Esta especificación hace posible que la función puede ser llamada
-  * por su definición a la hora de crear la tarea en freeRTOS.
-  *
-  */
- static void writeSensorsDataUITask(void* args) ;
- /**
-  * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
-  * que escribe la hora en la pantalla.
-  * @param args Esta especificación hace posible que la función puede ser llamada
-  * por su definición a la hora de crear la tarea en freeRTOS.
-  *
-  */
- static void runClockUITask(void* args) ;
- /**
-  * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
-  * que lee los tiempos de actividad del bombillo UVA.
-  * @param args Esta especificación hace posible que la función puede ser llamada
-  * por su definición a la hora de crear la tarea en freeRTOS.
-  *
-  */
- static void updateActiveTimeUVATask(void* args);
- /**
-  * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
-  * que lee los tiempos de actividad del bombillo UVB.

```

```

- * @param args Esta especificación hace posible que la función puede ser llamada
- * por su definición a la hora de crear la tarea en freeRTOS.
- *
- */
- static void updateActiveTimeUVBTask(void* args);
- /**
- * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
- * que lee los tiempos de actividad del bombillo Plants.
- * @param args Esta especificación hace posible que la función puede ser llamada
- * por su definición a la hora de crear la tarea en freeRTOS.
- *
- */
- static void updateActiveTimePlantsTask(void* args);
- /**
- * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
- * que controla la temperatura máxima del bombillo UVA.
- * @param args Esta especificación hace posible que la función puede ser llamada
- * por su definición a la hora de crear la tarea en freeRTOS.
- *
- */
- static void setMaxTemperatureUVATask(void* args);
- /**
- * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
- * que controla el sistema de luces.
- * @param args Esta especificación hace posible que la función puede ser llamada
- * por su definición a la hora de crear la tarea en freeRTOS.
- *
- */
- static void controllightsSystemTask(void* args);
- /**
- * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
- * que dibuja la pantalla de bienvenida y así poder gestionar las redes WiFi.
- * @param args Esta especificación hace posible que la función puede ser llamada
- * por su definición a la hora de crear la tarea en freeRTOS.
- *
- */
- static void drawWiFiMenuTask(void* args);
- /**
- * @brief Función llamada por createTasks(), se encarga de gestionar la tarea
- * que publica los datos de los sensores a Mosquitto, el bróker de MQTT.
- * @param args Esta especificación hace posible que la función puede ser llamada
- * por su definición a la hora de crear la tarea en freeRTOS.
- *
- */
- static void publishMQTTBrokerTask(void* args);
-
- #endif

```

- task.cpp

```

- #include "RTOS/tasks.h"
-
- // Handlers
- TaskHandle_t lvglHandler = NULL;
- TaskHandle_t updateBrightnessTaskHandler = NULL;
- TaskHandle_t writeSensorsDataUITaskHandler = NULL;
- TaskHandle_t runClockUITaskHandler = NULL;
- TaskHandle_t updateActiveTimeUVATaskHandler = NULL;
- TaskHandle_t updateActiveTimeUVBTaskHandler = NULL;
- TaskHandle_t updateActiveTimePlantsTaskHandler = NULL;
- TaskHandle_t setMaxTemperatureUVATaskHandler = NULL;
- TaskHandle_t lightSystemTaskHandler = NULL;
- TaskHandle_t publishMQTTBrokerTaskHandler = NULL;
-
- /* Functions */
- void initUITask(void) {
-     xTaskCreatePinnedToCore(runUI , "UITask",
-                             4096, NULL, 3, &lvglHandler, 1);
- }

```

```

- void createTasks(void) {
-     xTaskCreatePinnedToCore(updateScreenBrightnessTask, "UpdateScreenBrightnessTask",
-                             2048,NULL,1,&updateBrightnessTaskHandler,1);
-     xTaskCreatePinnedToCore(writeSensorsDataUITask,"UpdateDataUITask",
-                             2048,NULL,3,&writeSensorsDataUITaskHandler,1);
-     xTaskCreatePinnedToCore(runClockUITask,"RunClockUITask",
-                             2048,NULL,3,&runClockUITaskHandler,1);
-     xTaskCreatePinnedToCore(updateActiveTimeUVATask,"UpdateActiveTimeUVA",
-                             2048,NULL,3,&updateActiveTimeUVATaskHandler,1);
-     xTaskCreatePinnedToCore(updateActiveTimeUVBTask,"UpdateActiveTimeUVB",
-                             2048,NULL,3,&updateActiveTimeUVBTaskHandler,1);
-     xTaskCreatePinnedToCore(updateActiveTimePlantsTask,"UpdateActiveTimePlants",
-                             2048,NULL,3,&updateActiveTimePlantsTaskHandler,1);
-     xTaskCreatePinnedToCore(setMaxTemperatureUVATask,"SetMaxTempUVA",
-                             2048,NULL,3,&setMaxTemperatureUVATaskHandler,1);
-     xTaskCreatePinnedToCore(controlLightsSystemTask,"ControlLightsSystem",
-                             2048,NULL,3,&lightSystemTaskHandler,1);
-     xTaskCreatePinnedToCore(publishMQTTBrokerTask,"ConnectMQTT",
-                             4096,NULL,3,&publishMQTTBrokerTaskHandler,0);
- }
-
- // Calculates the reamaining time between "on hour" and "off hour"
- void getTimeDifference(char* hours_remaining, char* minutes_remaining,
-                       const char* start_hour, const char* start_minute,
-                       const char* finish_hour, const char* finish_minute) {
-     int hour_difference = 0, minute_difference = 0;
-     int on_hour = atoi(start_hour);
-     int on_minute = atoi(start_minute);
-     int off_hour = atoi(finish_hour);
-     int off_minute = atoi(finish_minute);
-
-     if (on_hour == off_hour) {
-         if (on_minute > off_minute) {
-             hour_difference = 23;
-             minute_difference = 60 - (on_minute - off_minute);
-         } else if (on_minute == off_minute) {
-             minute_difference = 0;
-         } else {
-             minute_difference = off_minute - on_minute;
-         }
-     } else if (on_hour > off_hour ) {
-         hour_difference = 24 - (on_hour - off_hour);
-         if (on_minute > off_minute) {
-             minute_difference = 60 - (on_minute - off_minute);
-         } else if (on_minute == off_minute) {
-             minute_difference = 0;
-         } else {
-             minute_difference = off_minute - on_minute;
-         }
-     } else if (off_hour > on_hour) {
-         hour_difference = off_hour - on_hour;
-         if (on_minute > off_minute) {
-             hour_difference--;
-             minute_difference = 60 - (on_minute - off_minute);
-         } else if (on_minute == off_minute) {
-             minute_difference = 0;
-         } else {
-             minute_difference = off_minute - on_minute;
-         }
-     }
-     sprintf(hours_remaining, "%02d", hour_difference);
-     sprintf(minutes_remaining, "%02d", minute_difference);
- }
-
- void getUIRollerTime(const char* roller_time, char* hour, char* minutes) {
-     hour[0] = roller_time[0];
-     hour[1] = roller_time[1];
-     minutes[0] = roller_time[3];
-     minutes[1] = roller_time[4];
- }

```

```

- /* Tasks */
- static void runUI (void *args) {
-     while (1) {
-         lv_timer_handler();
-         vTaskDelay(5 / portTICK_RATE_MS);
-     }
- }

- // Brightness control
- static void updateScreenBrightnessTask(void *args) {
-     int last_state = HIGH;
-     int current_state;
-     while (1) {
-         current_state = digitalRead(BUTTON_GPIO);
-         if (last_state == HIGH && current_state == LOW ) {
-             setDisplayBrightness(0);
-             last_state = LOW;
-         } else if (last_state == LOW && current_state == LOW) {
-             setDisplayBrightness(255);
-             last_state = HIGH;
-         }
-         vTaskDelay(140/portTICK_RATE_MS);
-     }
- }

- // Read&Write temps/humidity UI
- static void writeSensorsDataUITask(void *args) {
-     float value;
-     char formatted_value[8];
-     prepareSensors();
-     while (1) {
-         warm_hide_temp = readTemperatures(0);
-         cold_hide_temp = readTemperatures(1);
-         environment_temp = readEnvironmentTemp();
-         environment_hum = readEnvironmentHum();
-         humid_hide_temp = readHumidHideTemp();
-         humid_hide_hum = readHumidHideHum();
-         // Primer sensor DS18B20 - Cueva caliente
-         dtostrf(warm_hide_temp , 5 , 2 , formatted_value);
-         lv_label_set_text(ui_warmtemp, formatted_value);
-         // Segundo sensor DS18B20 - Cueva fría
-         dtostrf(cold_hide_temp , 5 , 2 , formatted_value);
-         lv_label_set_text(ui_coldtemp , formatted_value);
-         // Primer sensor AM2301 - Cueva húmeda
-         dtostrf(humid_hide_temp, 5 , 2 , formatted_value);
-         lv_label_set_text(ui_moisttemp , formatted_value);
-         dtostrf(humid_hide_hum, 5 , 2 , formatted_value);
-         lv_label_set_text(ui_moistrelativehum , formatted_value);
-         // Segundo sensor AM2301 - Entorno
-         dtostrf(environment_temp , 5 , 2 , formatted_value);
-         lv_label_set_text( ui_environmenttemp, formatted_value);
-         dtostrf(environment_hum, 5 , 2 , formatted_value);
-         lv_label_set_text( ui_environmentrelativehum, formatted_value);

-         vTaskDelay(100 / portTICK_RATE_MS);
-     }
- }

- // Updates screens clock
- static void runClockUITask(void *args) {
-     configRTC();
-     char rtc_time[8];
-     char format[] = "hh:mm";
-     while (1) {
-         strcpy(rtc_time, format);
-         reportTime().toString(rtc_time);
-         lv_label_set_text(ui_clock , rtc_time);
-         vTaskDelay(1000 / portTICK_RATE_MS);
-     }
- }

```

```

static void updateActiveTimeUVATask(void *args) {
    char on_roller_uva[10], off_roller_uva[10];
    char hours_remaining [10], minutes_remaining [10];
    char on_hours[3], on_minutes[3], off_hours[3], off_minutes[3];
    while(1) {
        if (ui_screen1 != lv_scr_act() && ui_screen2 != lv_scr_act() &&
            ui_screen4 != lv_scr_act() && ui_screen5 != lv_scr_act()) {

            lv_roller_get_selected_str(ui_onrollerscreen3, on_roller_uva, 10);
            lv_roller_get_selected_str(ui_offrollerscreen3, off_roller_uva, 10);

            getUIRollerTime(on_roller_uva, on_hours, on_minutes);
            getUIRollerTime(off_roller_uva, off_hours, off_minutes);

            getTimeDifference(hours_remaining, minutes_remaining,
                on_hours, on_minutes,
                off_hours, off_minutes);
            lv_label_set_text(ui_hours3 , hours_remaining);
            lv_label_set_text(ui_minutes3 , minutes_remaining);
        }

        vTaskDelay(100 / portTICK_RATE_MS);
    }
}

static void updateActiveTimeUVBTask(void *args) {
    char on_roller_uvb[10], off_roller_uvb[10], hours_remaining [10];
    char minutes_remaining [10];
    char on_hour[3], on_minutes[3], off_hours[3], off_minutes[3];
    while(1) {
        if (ui_screen1 != lv_scr_act() && ui_screen2 != lv_scr_act() &&
            ui_screen3 != lv_scr_act() && ui_screen5 != lv_scr_act()) {

            lv_roller_get_selected_str(ui_onrollerscreen4, on_roller_uvb, 10);
            lv_roller_get_selected_str(ui_offrollerscreen4, off_roller_uvb, 10);
            on_hour[0] = on_roller_uvb[0];
            on_hour[1] = on_roller_uvb[1];
            off_hours[0] = off_roller_uvb[0];
            off_hours[1] = off_roller_uvb[1];
            on_minutes[0] = on_roller_uvb[3];
            on_minutes[1] = on_roller_uvb[4];
            off_minutes[0] = off_roller_uvb[3];
            off_minutes[1] = off_roller_uvb[4];
            getTimeDifference(hours_remaining, minutes_remaining,
                on_hour, on_minutes,
                off_hours, off_minutes);
            lv_label_set_text(ui_hours4 , hours_remaining);
            lv_label_set_text(ui_minutes4 , minutes_remaining);
        }
        vTaskDelay(100 / portTICK_RATE_MS);
    }
}

static void updateActiveTimePlantsTask(void *args) {
    char on_roller_plants[10], off_roller_plants[10], hours_remaining [10];
    char minutes_remaining [10];
    char on_hour[3], on_minutes[3], off_hours[3], off_minutes[3];
    while(1) {
        if (ui_screen1 != lv_scr_act() && ui_screen2 != lv_scr_act() &&
            ui_screen4 != lv_scr_act() && ui_screen3 != lv_scr_act()) {

            lv_roller_get_selected_str(ui_onrollerscreen5, on_roller_plants, 10);
            lv_roller_get_selected_str(ui_offrollerscreen5, off_roller_plants, 10);
            on_hour[0] = on_roller_plants[0];
            on_hour[1] = on_roller_plants[1];
            off_hours[0] = off_roller_plants[0];
            off_hours[1] = off_roller_plants[1];
            on_minutes[0] = on_roller_plants[3];
            on_minutes[1] = on_roller_plants[4];
            off_minutes[0] = off_roller_plants[3];
            off_minutes[1] = off_roller_plants[4];
        }
    }
}

```

```

-         getTimeDifference(hours_remaining, minutes_remaining,
-                           on_hour, on_minutes,
-                           off_hours, off_minutes);
-         lv_label_set_text(ui_hours5 , hours_remaining);
-         lv_label_set_text(ui_minutes5 , minutes_remaining);
-     }
-     vTaskDelay(100 / portTICK_RATE_MS);
- }
- }

// Sets the maximum temperature that the UVA bulb would target
static void setMaxTemperatureUVATask(void *args) {
-     char max_temp_slider_value[8];
-     int value;
-     while(1) {
-         if (ui_screen1 != lv_scr_act() && ui_screen2 != lv_scr_act() &&
-             ui_screen4 != lv_scr_act() && ui_screen5 != lv_scr_act()) {

-             value = lv_slider_get_value(ui_tempslider3);
-             sprintf(max_temp_slider_value, "%02d", value);
-             lv_label_set_text(ui_targettemp3, max_temp_slider_value);
-         }
-         vTaskDelay(100 / portTICK_RATE_MS);
-     }
- }

// Lights handling
static void controllLightsSystemTask(void *args) {
-     char on_roller_uva[10], off_roller_uva[10] ;
-     char on_roller_uvb[10], off_roller_uvb[10] ;
-     char on_roller_plants[10], off_roller_plants[10] ;
-     char on_hours_uva[3], on_minutes_uva[3], off_hours_uva[3], off_minutes_uva[3];
-     char on_hours_uvb[3], on_minutes_uvb[3], off_hours_uvb[3], off_minutes_uvb[3];
-     char on_hour_plants[3], on_minutes_plants[3], off_hours_plants[3], off_minutes_plants[3];
-     digitalWrite(UVB_RELAY_GPIO, LOW);
-     digitalWrite(PLANTS_RELAY_GPIO, LOW);
-     while (1) {
-         lv_roller_get_selected_str(ui_onrollerscreen3, on_roller_uva, 10);
-         lv_roller_get_selected_str(ui_offrollerscreen3, off_roller_uva, 10);
-         getUIRollerTime(on_roller_uva, on_hours_uva, on_minutes_uva);
-         getUIRollerTime(off_roller_uva, off_hours_uva, off_minutes_uva);
-         controlUVALight(on_hours_uva, on_minutes_uva, off_hours_uva, off_minutes_uva);

-         lv_roller_get_selected_str(ui_onrollerscreen4, on_roller_uvb, 10);
-         lv_roller_get_selected_str(ui_offrollerscreen4, off_roller_uvb, 10);
-         getUIRollerTime(on_roller_uvb, on_hours_uvb, on_minutes_uvb);
-         getUIRollerTime(off_roller_uvb, off_hours_uvb, off_minutes_uvb);
-         controlUVBLight(on_hours_uvb, on_minutes_uvb, off_hours_uvb, off_minutes_uvb);

-         lv_roller_get_selected_str(ui_onrollerscreen5, on_roller_plants, 10);
-         lv_roller_get_selected_str(ui_offrollerscreen5, off_roller_plants, 10);
-         getUIRollerTime(on_roller_plants, on_hour_plants, on_minutes_plants);
-         getUIRollerTime(off_roller_plants, off_hours_plants, off_minutes_plants);
-         controlPlantsLight(on_hour_plants, on_minutes_plants, off_hours_plants, off_minutes_plants);

-         vTaskDelay(1000 / portTICK_RATE_MS);
-     }
- }

static void publishMQTTBrokerTask(void *args) {
-     while (1) {
-         String publish_data = "";
-         StaticJsonDocument<300> json_data;
-         json_data["warm_hide_temp"] = warm_hide_temp;
-         json_data["cold_hide_temp"] = cold_hide_temp;
-         json_data["environment_temp"] = environment_temp;
-         json_data["environment_humidity"] = environment_hum;
-         json_data["humid_hide_temp"] = humid_hide_temp;
-         json_data["humid_hide_humidity"] = humid_hide_hum;
-         serializeJson(json_data , publish_data);

```

```

- mqtt_client.publish("iot-esp32/tfg", 0, true, (char*)publish_data.c_str());
- vTaskDelay(5000 / portTICK_RATE_MS);
- }
- }

```

- ui.h

```

- /**
-  * @brief Fichero generado por SquareLine Studio 1.2.3, contiene
-  * la definición de todos los elementos involucrados en la GUI del
-  * microcontrolador, como widgets, pantallas, eventos, etc.
-  * @file ui.h
-  * @author Jesús Carmelo González Domínguez
-  * @date 14-07-2023
-  */
-
- // This file was generated by SquareLine Studio
- // SquareLine Studio version: SquareLine Studio 1.2.3
- // LVGL version: 8.3.4
- // Project name: SquareLine_Project
-
- #ifndef _SQUARELINE_PROJECT_UI_H
- #define _SQUARELINE_PROJECT_UI_H
-
- #ifdef __cplusplus
- extern "C" {
- #endif
-
- #if defined __has_include
- #if __has_include("lvgl.h")
- #include "lvgl.h"
- #elif __has_include("lvgl/lvgl.h")
- #include "lvgl/lvgl.h"
- #else
- #include "lvgl.h"
- #endif
- #else
- #include "lvgl.h"
- #endif
-
- #include "ui_comp.h"
- #include "ui_comp_hook.h"
- #include "ui_events.h"
- extern lv_obj_t *ui_loadingscreen;
- extern lv_obj_t *ui_loadingscreentext;
- extern lv_obj_t *ui_loadingscreenssid;
- extern lv_obj_t *ui_screen1;
- extern lv_obj_t *ui_topbarscreen1;
- extern lv_obj_t *ui_topbartitlescreen1;
- extern lv_obj_t *ui_clock;
- extern lv_obj_t *ui_wifiswitch;
- extern lv_obj_t *ui_wifitext;
- extern lv_obj_t *ui_textmenuscreen1;
- extern lv_obj_t *ui_environmenttemp;
- extern lv_obj_t *ui_warmtemp;
- extern lv_obj_t *ui_moisttemp;
- extern lv_obj_t *ui_coldtemp;
- extern lv_obj_t *ui_environmentrelativehum;
- extern lv_obj_t *ui_moistrelativehum;
- extern lv_obj_t *ui_wifilist;
- extern lv_obj_t *ui_wifilistoptions;
- extern lv_obj_t *ui_reportwifimessagebox;
- extern lv_obj_t *ui_wifikeyboard;
- extern lv_obj_t *ui_wifipasswordarea;
- void ui_event_nextbuttonscreen1( lv_event_t * e);
- void ui_event_settingspopupwifiiwindow( lv_event_t * e);
- extern lv_obj_t *ui_nextbuttonscreen1;
- extern lv_obj_t *ui_nextbuttontextscreen1;
- extern lv_obj_t *ui_screen2;
- extern lv_obj_t *ui_topbarscreen2;
- extern lv_obj_t *ui_topbartitlecreen2;

```



```

- extern Lv_obj_t *ui_textmenuscreen2;
- void ui_event_backbuttonscreen2( Lv_event_t * e);
- extern Lv_obj_t *ui_backbuttonscreen2;
- extern Lv_obj_t *ui_backbuttontextscreen2;
- extern Lv_obj_t *ui_uvaswitch;
- extern Lv_obj_t *ui_uvbswitch;
- extern Lv_obj_t *ui_plantsswitch;
- void ui_event_gouvabutton( Lv_event_t * e);
- extern Lv_obj_t *ui_gouvabutton;
- extern Lv_obj_t *ui_gouvatext;
- void ui_event_gouvbutton( Lv_event_t * e);
- extern Lv_obj_t *ui_gouvbutton;
- extern Lv_obj_t *ui_gouvbtext;
- void ui_event_goplantsbutton( Lv_event_t * e);
- extern Lv_obj_t *ui_goplantsbutton;
- extern Lv_obj_t *ui_goplantstext;
- extern Lv_obj_t *ui_screen3;
- extern Lv_obj_t *ui_textmenuscreen3;
- void ui_event_backbuttonscreen3( Lv_event_t * e);
- extern Lv_obj_t *ui_backbuttonscreen3;
- void ui_event_backbuttontextscreen3( Lv_event_t * e);
- extern Lv_obj_t *ui_backbuttontextscreen3;
- extern Lv_obj_t *ui_titlescreen3;
- extern Lv_obj_t *ui_onrollerscreen3;
- extern Lv_obj_t *ui_offrollerscreen3;
- extern Lv_obj_t *ui_tiempoactivad3;
- extern Lv_obj_t *ui_hours3;
- extern Lv_obj_t *ui_timetext3;
- extern Lv_obj_t *ui_minutes3;
- extern Lv_obj_t *ui_tempslider3;
- extern Lv_obj_t *ui_targettemp3;
- extern Lv_obj_t *ui_centgr3;
- extern Lv_obj_t *ui_screen4;
- extern Lv_obj_t *ui_textmenuscreen4;
- void ui_event_backbuttonscreen4( Lv_event_t * e);
- extern Lv_obj_t *ui_backbuttonscreen4;
- void ui_event_backbuttontextscreen4( Lv_event_t * e);
- extern Lv_obj_t *ui_backbuttontextscreen4;
- extern Lv_obj_t *ui_titlescreen4;
- extern Lv_obj_t *ui_onrollerscreen4;
- extern Lv_obj_t *ui_offrollerscreen4;
- extern Lv_obj_t *ui_tiempoactivad4;
- extern Lv_obj_t *ui_hours4;
- extern Lv_obj_t *ui_timetext4;
- extern Lv_obj_t *ui_minutes4;
- extern Lv_obj_t *ui_screen5;
- extern Lv_obj_t *ui_textmenuscreen5;
- void ui_event_backbuttonscreen5( Lv_event_t * e);
- extern Lv_obj_t *ui_backbuttonscreen5;
- void ui_event_backbuttontextscreen5( Lv_event_t * e);
- extern Lv_obj_t *ui_backbuttontextscreen5;
- extern Lv_obj_t *ui_titlescreen5;
- extern Lv_obj_t *ui_onrollerscreen5;
- extern Lv_obj_t *ui_offrollerscreen5;
- extern Lv_obj_t *ui_tiempoactivad5;
- extern Lv_obj_t *ui_hours5;
- extern Lv_obj_t *ui_timetext5;
- extern Lv_obj_t *ui_minutes5;
- extern Lv_obj_t *ui___initial_actions0;

- void ui_init(void);

- #ifdef __cplusplus
- } /*extern "C"*/
- #endif

- #endif

```

- sensor.h

```
/**
 * @brief Fichero que define las funciones relativas a los sensores.
 * @file sensor.h
 * @author Jesús Carmelo González Domínguez
 * @date 14-07-2023
 */

#ifndef SENSOR_GUARD
#define SENSOR_GUARD

#include <DallasTemperature.h>
#include <DHT.h>

/**
 * @brief Pin que usará el bus de los sensores DS18B20
 */
#define DS18B20s_GPIO 8
/**
 * @brief Pin que usará el primer sensor DHT
 */
#define DHT_GPIO_1 9
/**
 * @brief Pin que usará el segundo sensor DHT
 */
#define DHT_GPIO_2 46
/**
 * @brief Definición del tipo de sensor DHT
 */
#define DHT_TYPE DHT21

/* Variables */
/**
 * @brief Bus OneWire que utilizarán los sensores DS18B20
 */
extern OneWire gpio_bus;
/**
 * @brief Sensores DS18B20
 */
extern DallasTemperature ds18b20_sensors;
/**
 * @brief Direcciones de los sensores DS18B20, para ser indentificados dentro
 * del bus.
 */
extern DeviceAddress ds18b20_address;
/**
 * @brief Primer sensor DHT, midiendo los valores del ambiente.
 */
extern DHT dht_environment;
/**
 * @brief Segundo sensor DHT, midiendo los valores de la llamada cueva húmeda.
 */
extern DHT dht_humid_hide;
/**
 * @brief Valor de la temperatura en la zona iluminada/calentada por el bombillo
 * UVA.
 */
static double warm_hide_temp;
/**
 * @brief Valor de la temperatura en la zona fría, no calentada por ningún
 * bombillo.
 */
static double cold_hide_temp;
/**
 * @brief Valor de la temperatura ambiente
 */
static double environment_temp;
/**
 * @brief Valor de la humedad relativa del ambiente
 */
static double environment_hum;
```

```

- /**
-  * @brief Valor de la temperatura de la llamada cueva húmeda.
-  */
- static double humid_hide_temp;
- /**
-  * @brief Valor de la humedad relativa de la llamada cueva húmeda.
-  */
- static double humid_hide_hum;
-
- /* Functions */
- /**
-  * @brief Configura los pines para funcionar correctamente con los sensores.
-  */
- void configSensorsGPIO(void);
- /**
-  * @brief Inicia los sensores.
-  */
- void prepareSensors(void);
- /**
-  * @brief Comprueba si un sensor DS18B20 está conectado.
-  * @param index Identificador del sensor, puede ser 0 o 1.
-  * @return true si está conectado, false si no.
-  */
- bool isTemperatureSensorConnected(const int index);
- /**
-  * @brief Lee la temperatura de un sensor DS18B20.
-  * @param index Identificador del sensor, puede ser 0 o 1.
-  * @return Valor de la temperatura.
-  */
- double readTemperatures(const int index);
- /**
-  * @brief Lee la temperatura del sensor DHT en la llamada cueva húmeda.
-  * @return Valor de la temperatura.
-  */
- double readHumidHideTemp(void);
- /**
-  * @brief Lee la humedad relativa del sensor DHT en la llamada cueva húmeda.
-  * @return Valor de la humedad relativa.
-  */
- double readHumidHideHum(void);
- /**
-  * @brief Lee la temperatura del sensor DHT ambiente.
-  * @return Valor de la temperatura.
-  */
- double readEnvironmentTemp(void);
- /**
-  * @brief Lee la humedad relativa del sensor DHT ambiente.
-  * @return Valor de la humedad relativa.
-  */
- double readEnvironmentHum(void);
-
- #endif

```

- sensor.cpp

```

- #include "Devices/sensor.h"
-
- /* Variables */
- OneWire gpio_bus(DS18B20s_GPIO);
- DallasTemperature ds18b20_sensors(&gpio_bus);
- DeviceAddress ds18b20_address;
- DHT dht_environment(DHT_GPIO_1, DHT_TYPE);
- DHT dht_humid_hide(DHT_GPIO_2, DHT_TYPE);
-
- /* Functions */
- void configSensorsGPIO(void) {
-     pinMode(DS18B20s_GPIO, INPUT_PULLUP);
-     pinMode(DHT_GPIO_1, INPUT);
-     pinMode(DHT_GPIO_2, INPUT);
- }

```

```

- void prepareSensors(void) {
-   ds18b20_sensors.begin();
-   dht_environment.begin();
-   dht_humid_hide.begin();
- }
-
- bool isTemperatureSensorConnected(const int index) {
-   ds18b20_sensors.getAddress(ds18b20_address , index);
-   if (ds18b20_sensors.isConnected(ds18b20_address)) {
-     return true;
-   } else {
-     return false;
-   }
- }
-
- double readTemperatures(const int index) {
-   ds18b20_sensors.getAddress(ds18b20_address , index);
-   if (ds18b20_sensors.isConnected(ds18b20_address)) {
-     ds18b20_sensors.requestTemperatures();
-     return ds18b20_sensors.getTempCByIndex(index);
-   } else {
-     Serial.print("No se puede conectar con el sensor.");
-     return 99.99;
-   }
- }
-
- double readHumidHideTemp() {
-   double value = dht_humid_hide.readTemperature();
-   if (!isnan(value)) {
-     return value;
-   } else {
-     return 99.99;
-   }
- }
-
- double readHumidHideHum() {
-   double value = dht_humid_hide.readHumidity();
-   if (!isnan(value)) {
-     return value;
-   } else {
-     return 99.99;
-   }
- }
-
- double readEnvironmentTemp() {
-   double value = dht_environment.readTemperature();
-   if (!isnan(value)) {
-     return value;
-   } else {
-     return 99.99;
-   }
- }
-
- double readEnvironmentHum() {
-   double value = dht_environment.readHumidity();
-   if (!isnan(value)) {
-     return value;
-   } else {
-     return 99.99;
-   }
- }
-

```

- rtc.h

```
/**
 * @brief Fichero que define las funciones relacionadas con el reloj.
 *
 * @file rtc.h
 * @author Jesús Carmelo González Domínguez
 * @date 14-07-2023
 */

#ifndef RTC_GUARD
#define RTC_GUARD

#include "RTCLib.h"
#include "Wire.h"

/**
 * @brief Pin SDA del módulo DS3231 bajo el bus I2C
 */
#define RTC_SDA 11

/**
 * @brief Pin SCL del módulo DS3231 bajo el bus I2C
 */
#define RTC_SCL 17

/* Variables */
/**
 * @brief Instancia del reloj DS3231
 */
extern RTC_DS3231 rtc;

/**
 * @brief Instancia del bus I2C que utilizará el reloj.
 * Tiene que ser el segundo bus que ofrece la placa porque el primero ya
 * está siendo utilizado por el panel táctil.
 */
extern TwoWire i2c_rtc;

/* Functions */
/**
 * @brief Configura el bus I2C para el reloj y lo inicia. En caso de que al
 * iniciar el dispositivo el reloj haya detectado que se perdió la energía
 * por parte de la batería, se asignará por defecto la hora de compilación
 * como la hora actual.
 */
void configRTC(void);

/**
 * @brief Informa sobre la hora.
 * @param DateTime Devuelve un objeto que contiene toda la información de un
 * determinado instante de tiempo, como la hora, minutos, etc.
 */
DateTime reportTime(void);

#endif
```

- rtc.cpp

```
#include "Devices/rtc.h"

RTC_DS3231 rtc;
TwoWire i2c_rtc = TwoWire(1);

void configRTC(void) {
    i2c_rtc.begin(RTC_SDA , RTC_SCL, 100000);
    bool status = rtc.begin(&i2c_rtc);
    if (!status) {
        Serial.print("No se ha podido iniciar el reloj");
    } else {
        Serial.print("Se ha podido iniciar el reloj");
    }
    if (rtc.lostPower()) {
```

```

-     rtc.adjust(DateTime(F(__DATE__),F(__TIME__)));
- }
- }
-
- DateTime reportTime(void) {
-     return rtc.now();
- }

```

- networking.h

```

- /**
-  * @brief Fichero que define las funciones encargadas de gestionar la
-  * comunicación via WiFi, el protocolo MQTT, etc.
-  *
-  * @file networking.h
-  * @author Jesús Carmelo González Domínguez
-  * @date 14-07-2023
-  */
-
- #ifndef NETWORKING_GUARD
- #define NETWORKING_GUARD
-
- #include "UI/ui.h"
- #include "WiFi.h"
- #include "vector"
- #include "Preferences.h"
- #include "time.h"
- #include <AsyncMqttClient.h>
- #include <ArduinoJson.h>
-
- /**
-  * @brief Definición de una matriz de strings que contiene la información
-  * obtenida al realizar el escaneo de redes WiFi.
-  */
- typedef std::vector<std::vector<String>> WifiScanData;
-
- /* Variables */
-
- /**
-  * @brief Tiempo máximo permitido para intentar conectarse a una red WiFi.
-  */
- extern const ulong wifi_timeout;
- /**
-  * @brief Objeto que contiene los datos de los credenciales WiFi.
-  */
- extern Preferences device_data;
- /**
-  * @brief Matriz de strings que contiene la información obtenida al realizar
-  * el escaneo de redes WiFi.
-  */
- extern WifiScanData wifi_data;
- /**
-  * @brief Usado principalmente para debugging, devuelve la cadena de datos
-  * recibida por el servidor.
-  */
- extern String payload;
- /**
-  * @brief Objeto que contiene la IP bróker MQTT
-  */
- extern IPAddress MQTT_SERVER;
- /**
-  * @brief Instancia del protocolo MQTT en el ESP32
-  */
- extern AsyncMqttClient mqtt_client;
- // Timers
- /**
-  * @brief Temporizador utilizado para controlar la llamada de la función que
-  * reconecta con el bróker MQTT.
-  */
- extern TimerHandle_t mqtt_reconnect_timer;

```

```

- /**
-  * @brief Temporizador utilizado para controlar la llamada de la función que
-  * reconecta con el WiFi.
-  */
- extern TimerHandle_t wifi_reconnect_timer;
-
- // Functions
- /**
-  * @brief Escanea las redes WiFi disponibles.
-  * @return Devuelve una matriz de strings con la información de todas las redes
-  * obtenidas.
-  */
- WifiScanData getWiFiSSIDs(void);
- /**
-  * @brief Inicializa la memoria flash
-  */
- void initFlashMemory(void);
- /**
-  * @brief Escribe el SSID en la memoria flash.
-  */
- void writeSSIDFlash(const char *ssid);
- /**
-  * @brief Escribe el password en la memoria flash.
-  */
- void writePasswordFlash(const char *password);
- /**
-  * @brief Lee de la memoria flash el SSID.
-  * @return Devuelve el SSID.
-  */
- const String readSSIDFlash(void);
- /**
-  * @brief Lee de la memoria flash el password.
-  * @return Devuelve el password.
-  */
- const String readPasswordFlash(void);
- /**
-  * @brief Función que prueba los credenciales introducidos para comprobar si
-  * se ha podido conectar con éxito. Si es así, escribe los credenciales en
-  * la memoria flash.
-  * @param ssid El SSID que identifica la red WiFi.
-  * @param password El password que autoriza el acceso a la red WiFi.
-  * @return Devuelve true si se ha podido conectar, false si no.
-  */
- bool tryWiFi(const char *ssid, const char *password);
- /**
-  * @brief Función llamada en el main.cpp, se utiliza para iniciar
-  * los timers que empezarán a funcionar en caso de que el WiFi esté conectado o no.
-  * - Si se detecta que se ha obtenido una IP, se llama a la función connectMQTT()
-  * para establecer una conexión con el bróker MQTT. Además, se activa un switch en
-  * la GUI para notificar que el WiFi está conectado, se para el timer de reconexión
-  * wifi_reconnect_timer y se activa el timer para reconectar con el bróker MQTT.
-  * - Si se detecta que no se ha desconectado del WiFi, se para el timer de reconexión
-  * al bróker MQTT y se activa el timer de reconexión WiFi para intentar recuperar
-  * la conexión. Además, se desactiva un switch gráfica para notificar al usuario que
-  * el WiFi no está en funcionamiento.
-  * @param event El SSID que identifica la red WiFi.
-  */
- void wifiEvent(WiFiEvent_t event);
- /**
-  * @brief Función llamada en el main.cpp, se utiliza para establecer una
-  * conexión WiFi, con los datos almacenados en la memoria flash.
-  */
- void connectWiFi(void);
- /**
-  * @brief Función llamada por el timer wifi_reconnect_timer que intenta restablecer
-  * la conexión previa.
-  */
- void wifiReconnect(void);
- /**
-  * @brief Función que crea los timers mqtt_reconnect_timer, wifi_reconnect_timer y fija
-  * todos los comportamientos a realizar cuando se conecta, desconecta, suscribe a un tema,

```

```

- * publica, etc. Finalmente, configura el bróker MQTT con su IP y puerto de destino.
- */
- void configMQTT(void);
- /**
-  * @brief Conecta con el bróker MQTT.
-  */
- void connectMQTT();
- /**
-  * @brief En caso de establecer una conexión con el bróker MQTT, intenta suscribirse
-  * a un tópic con subscribeMQTT().
-  * @param status true si se ha conectado, false si no.
-  */
- void onMQTTConnect(bool status);
- /**
-  * @brief En caso de perder la conexión con el bróker MQTT, comprueba primero si el
-  * WiFi está conectado y si es así, inicia el timer mqtt_reconnect_timer.
-  * @param reason Motivo de la desconexión, que pueden ser los siguientes:
-  * - TCP_DISCONNECTED
-  * - MQTT_UNACCEPTABLE_PROTOCOL_VERSION
-  * - MQTT_IDENTIFIER_REJECTED
-  * - MQTT_SERVER_UNAVAILABLE
-  * - MQTT_MALFORMED_CREDENTIALS
-  * - MQTT_NOT_AUTHORIZED
-  * - ESP8266_NOT_ENOUGH_SPACE
-  * - TLS_BAD_FINGERPRINT
-  */
- void onMQTTDisconnect(AsyncMqttClientDisconnectReason reason);
- /**
-  * @brief Para debugging. Cuando se suscribe correctamente a un tema publica
-  * información relevante por el serial.
-  * @param packet_id Identificador del paquete enviado
-  * @param qos Nivel de calidad de servicio establecido
-  */
- void onMQTTSubscribe(uint16_t packet_id, uint8_t qos);
- /**
-  * @brief Para debugging. Cuando no se puede suscribir a un tema publica
-  * información relevante por el serial.
-  * @param packet_id Identificador del paquete enviado
-  */
- void onMQTTUnsubscribe(uint16_t packet_id);
- /**
-  * @brief Para debugging. Cuando el mensaje llega correctamente al bróker MQTT, se
-  * publica información relevante por el serial en relación al conteido enviado, en este
-  * caso, la información de temperatura/humedad de los sensores.
-  * @param topic Tópico
-  * @param payload Mensaje enviado correctamente al servidor
-  * @param properties Parámetro específico de la librería.
-  * @param size_Len Logitud total del mensaje
-  * @param index Parámetro específico de la librería.
-  * @param total Parámetro específico de la librería.
-  */
- void onMQTTReceived(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len,
- size_t index, size_t total);
- /**
-  * @brief Para debugging. Una vez publicada la información correctamente en el servidor,
-  * muestra información relevante por el serial.
-  * @param packet_id Tópico
-  */
- void onMQTTPublish(uint16_t packet_id);
- /**
-  * @brief Se suscribe a un tema para luego ser gestionado por el bróker MQTT.
-  * @param packet_id Tópico
-  */
- void subscribeMQTT(void);
- /**
-  * @brief Obtiene el contenido del payload enviado al bróker MQTT, iterando el payload
-  * como un array de chars y devolviendo la string formada tras las iteraciones.
-  * @param data Contenido del mensaje o payload
-  * @param Len Longitud del mensaje o payload
-  * @return Devuelve como string el contenido del mensaje o payload.
-  */
- */

```



```
- String getPayloadContent(char* data, size_t Len);
- #endif
```

```
- networking.cpp
```

```
#include "Devices/networking.h"

/* Variables */
const ulong wifi_timeout = 10000;
Preferences device_data;
WifiScanData wifi_data;
String payload;
IPAddress MQTT_SERVER(5 , 75 , 186, 218);
AsyncMqttClient mqtt_client;
// Timers
TimerHandle_t mqtt_reconnect_timer;
TimerHandle_t wifi_reconnect_timer;

/* Functions */
WifiScanData getWiFiSSIDs(void) {
    WifiScanData wifi_data;
    int networks_number = WiFi.scanNetworks();
    wifi_data.resize(networks_number);
    for (int i = 0; i < wifi_data.size(); i++) {
        wifi_data[i].resize(2);
        wifi_data[i][0] = WiFi.SSID(i);
        if (WiFi.RSSI(i) <= (-67)) {
            wifi_data[i][1] = "Señal mala";
        } else if (WiFi.RSSI(i) > (-66) && WiFi.RSSI(i) <= (-50)) {
            wifi_data[i][1] = "Señal media";
        } else {
            wifi_data[i][1] = "Señal buena";
        }
    }
    return wifi_data;
}

void initFlashMemory(void) {
    device_data.begin("credentials", false);
}

void writeSSIDFlash(const char *ssid) {
    device_data.putString("SSID", ssid);
}

void writePasswordFlash(const char *password) {
    device_data.putString("PASSWORD", password);
}

const String readSSIDFlash(void) {
    const String ssid = device_data.getString("SSID", "");
    return ssid;
}

const String readPasswordFlash(void) {
    const String password = device_data.getString("PASSWORD", "");
    return password;
}

bool tryWiFi(const char *ssid, const char *password) {
    const ulong start_time = millis();
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED && (millis() - start_time) < wifi_timeout) {
        vTaskDelay(250 / portTICK_RATE_MS);
        Serial.print(".");
    }
    if(WiFi.status() == WL_CONNECTED) {
```

```

Serial.print("\nCONECTADO");
writeSSIDFlash(ssid);
writePasswordFlash(password);
return true;
} else {
Serial.print("\nNO CONECTADO");
return false;
}
}

void wifiEvent(WiFiEvent_t event) {
switch (event) {
case SYSTEM_EVENT_STA_GOT_IP:
Serial.println("Evento: Se ha obtenido IP");
lv_obj_add_state(ui_wifiswitch, LV_STATE_CHECKED);
connectMQTT();
xTimerStop(wifi_reconnect_timer, 0);
break;
case SYSTEM_EVENT_STA_DISCONNECTED:
Serial.println("Se ha desconectado del WiFi.");
xTimerStop(mqtt_reconnect_timer, 0);
xTimerStart(wifi_reconnect_timer, 0);
lv_obj_clear_state(ui_wifiswitch, LV_STATE_CHECKED);
break;
}
}

void connectWiFi(void) {
Serial.println("Entrando al WiFi.");
WiFi.disconnect();
WiFi.mode(WIFI_STA);
Serial.println("Número de redes disponibles: ");
Serial.print(WiFi.scanNetworks());
WiFi.begin(readSSIDFlash().c_str(), readPasswordFlash().c_str());
while (WiFi.status() != WL_CONNECTED) {
delay(100);
Serial.print('.');
}
Serial.println("Conexión establecida.");
Serial.println("IP:");
Serial.println(WiFi.localIP());
}

void wifiReconnect(void) {
WiFi.getAutoReconnect();
while (WiFi.status() != WL_CONNECTED) {
delay(100);
Serial.print('.');
}
}

void configMQTT(void) {
mqtt_reconnect_timer = xTimerCreate("MQTTTimer",
pdMS_TO_TICKS(2000),
pdFALSE,
(void*)0,
reinterpret_cast<TimerCallbackFunction_t>(connectMQTT)
);
wifi_reconnect_timer = xTimerCreate("WiFiTimer",
pdMS_TO_TICKS(5000),
pdFALSE,
(void*)0,
reinterpret_cast<TimerCallbackFunction_t>(wifiReconnect)
);

mqtt_client.onConnect(onMQTTConnect);
mqtt_client.onDisconnect(onMQTTDisconnect);
mqtt_client.onSubscribe(onMQTTSubscribe);
mqtt_client.onUnsubscribe(onMQTTUnsubscribe);
mqtt_client.onMessage(onMQTTReceived);
mqtt_client.onPublish(onMQTTPublish);
}

```

```

- mqtt_client.setServer( MQTT_SERVER , 30000);
- }
-
- void connectMQTT() {
-     Serial.println("Conectando al servidor MQTT..");
-     mqtt_client.connect();
- }
-
- void onMQTTConnect(bool status) {
-     Serial.println("Conectado al servidor MQTT!");
-     subscribeMQTT();
- }
-
- void onMQTTDisconnect(AsyncMqttClientDisconnectReason reason) {
-     Serial.println("Desconectado del servidor MQTT.");
-     if(WiFi.isConnected()) {
-         xTimerStart(mqtt_reconnect_timer, 0);
-     }
- }
-
- void onMQTTSubscribe(uint16_t packet_id, uint8_t qos) {
-     Serial.println("Suscripción correcta!");
-     Serial.print("  packetId: ");
-     Serial.println(packet_id);
-     Serial.print("  qos: ");
-     Serial.println(qos);
- }
-
- void onMQTTUnsubscribe(uint16_t packet_id) {
-     Serial.println("Unsuscripción correcta.");
-     Serial.print("  packetId: ");
-     Serial.println(packet_id);
- }
-
- void onMQTTReceived(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len,
- size_t index, size_t total) {
-     Serial.println("Topico recibido: ");
-     Serial.println(topic);
-     Serial.print(" : ");
-
-     String content = getPayloadContent(payload, len);
-
-     StaticJsonDocument<300> received;
-     DeserializationError error = deserializeJson(received, content);
-     if (error) {
-         Serial.println("ERROR!");
-         return;
-     }
-
-     float data = received["environment_temp"];
-     Serial.print(data);
-     Serial.println();
- }
-
- void onMQTTPublish(uint16_t packet_id) {
-     Serial.println("Publicación reconocida: ");
-     Serial.print("  packetId: ");
-     Serial.println(packet_id);
- }
-
- void subscribeMQTT(void) {
-     uint16_t packetIdSub = mqtt_client.subscribe("iot-esp32/tfg", 0);
-     Serial.print("Suscrito a QoS 0, id del paquete: ");
-     Serial.println(packetIdSub);
- }
-
- String getPayloadContent(char* data, size_t len) {
-     String content = "";
-     for(size_t i = 0; i < len; i++) {
-         content.concat(data[i]);
-     }
- }

```

```

-     return content;
- }
-

```

- ft6236.h

```

- /**
-  * @brief Fichero que contiene la clase que configura el driver del panel táctil. Todos los
-  * valores de este fichero son estáticos para funcionar con el modelo FT6236 de Focaltech.
-  * Lo único que se ha añadido han sido los pines del bus I2C, que se comentan más abajo.
-  * Documentación de la librería:
-  * https://github.com/DustinWatts/FT6236/tree/main
-  * @file ft6236.h
-  * @author Jesús Carmelo González Domínguez
-  * @date 14-07-2023
-  */
-
- #ifndef FT6236_GUARD
- #define FT6236_GUARD
-
- #include "Arduino.h"
- #include <Wire.h>
-
- #define FT6236_ADDR 0x38 // I2C address
- #define FT6236_G_FT5201ID 0xA8 // FocalTech's panel ID
- #define FT6236_REG_NUMTOUCHES 0x02 // Number of touch points
-
- #define FT6236_NUM_X 0x33 // Touch X position
- #define FT6236_NUM_Y 0x34 // Touch Y position
-
- #define FT6236_REG_MODE 0x00 // Device mode, either WORKING or FACTORY
- #define FT6236_REG_CALIBRATE 0x02 // Calibrate mode
- #define FT6236_REG_WORKMODE 0x00 // Work mode
- #define FT6236_REG_FACTORYMODE 0x40 // Factory mode
- #define FT6236_REG_THRESHOLD 0x80 // Threshold for touch detection
- #define FT6236_REG_POINTRATE 0x88 // Point rate
- #define FT6236_REG_FIRMVERS 0xA6 // Firmware version
- #define FT6236_REG_CHIPIID 0xA3 // Chip selecting
- #define FT6236_REG_VENDID 0xA8 // FocalTech's panel ID
-
- #define FT6236_VENDID 0x11 // FocalTech's panel ID
- #define FT6206_CHIPIID 0x06 // FT6206 ID
- #define FT6236_CHIPIID 0x36 // FT6236 ID
- #define FT6236U_CHIPIID 0x64 // FT6236U ID
-
- #define FT6236_DEFAULT_THRESHOLD 128 // Default threshold for touch detection
-
- /**
-  * @brief Pin SDA asignado al panel táctil. Se obtuvo mirando los schematics de la placa
-  * de desarrollo:
-  * https://github.com/Makerfabs/Makerfabs-ESP32-S3-SPI-TFT-with-Touch/blob/main/hardware/ESP32-S3%20TFT%20Touch%20v1.1\(3.5'%20ili9488\).PDF
-  */
- #define SDA_FT6236 38
- /**
-  * @brief Pin SCL asignado al panel táctil. Se obtuvo mirando los schematics de la placa
-  * de desarrollo:
-  * https://github.com/Makerfabs/Makerfabs-ESP32-S3-SPI-TFT-with-Touch/blob/main/hardware/ESP32-S3%20TFT%20Touch%20v1.1\(3.5'%20ili9488\).PDF
-  */
- #define SCL_FT6236 39
-
- class TS_Point
- {
- public:
-     TS_Point(void);
-     TS_Point(int16_t x, int16_t y, int16_t z);
-

```

```

-     bool operator==(TS_Point);
-     bool operator!=(TS_Point);
-
-     int16_t x;
-     int16_t y;
-     int16_t z;
- };
-
- class FT6236
- {
- public:
-     FT6236(void);
-     void debug(void);
-     boolean begin(uint8_t thresh = FT6236_DEFAULT_THRESHOLD, int8_t sda = -1, int8_t scl = -1);
-     uint8_t touched(void);
-     TS_Point getPoint(uint8_t n = 0);
-
- private:
-     void writeRegister8(uint8_t reg, uint8_t val);
-     uint8_t readRegister8(uint8_t reg);
-
-     void readData(void);
-     uint8_t touches;
-     uint16_t touchX[2], touchY[2], touchID[2];
- };
-
- #endif

```

- ft6236.cpp

```

- /**
-  * @brief Fichero que contiene la clase que configura el driver del panel táctil. Todos los
-  * valores de este fichero son estáticos para funcionar con el modelo FT6236 de Focaltech.
-  * Lo único que se ha añadido han sido los pines del bus I2C, que se comentan más abajo.
-  * Documentación de la librería:
-  * https://github.com/DustinWatts/FT6236/tree/main
-  * @file ft6236.h
-  * @author Jesús Carmelo González Domínguez
-  * @date 14-07-2023
-  */
-
- #ifndef FT6236_GUARD
- #define FT6236_GUARD
-
- #include "Arduino.h"
- #include <Wire.h>
-
- #define FT6236_ADDR 0x38 // I2C address
- #define FT6236_G_FT5201ID 0xA8 // FocalTech's panel ID
- #define FT6236_REG_NUMTOUCHES 0x02 // Number of touch points
-
- #define FT6236_NUM_X 0x33 // Touch X position
- #define FT6236_NUM_Y 0x34 // Touch Y position
-
- #define FT6236_REG_MODE 0x00 // Device mode, either WORKING or FACTORY
- #define FT6236_REG_CALIBRATE 0x02 // Calibrate mode
- #define FT6236_REG_WORKMODE 0x00 // Work mode
- #define FT6236_REG_FACTORYMODE 0x40 // Factory mode
- #define FT6236_REG_THRESHOLD 0x80 // Threshold for touch detection
- #define FT6236_REG_POINTRATE 0x88 // Point rate
- #define FT6236_REG_FIRMVERS 0xA6 // Firmware version
- #define FT6236_REG_CHIPID 0xA3 // Chip selecting
- #define FT6236_REG_VENDID 0xA8 // FocalTech's panel ID
-
- #define FT6236_VENDID 0x11 // FocalTech's panel ID
- #define FT6206_CHIPID 0x06 // FT6206 ID
- #define FT6236_CHIPID 0x36 // FT6236 ID
- #define FT6236U_CHIPID 0x64 // FT6236U ID
-
- #define FT6236_DEFAULT_THRESHOLD 128 // Default threshold for touch detection

```

```

- /**
-  * @brief Pin SDA asignado al panel táctil. Se obtuvo mirando los schematics de la placa
-  * de desarrollo:
-  * https://github.com/Makerfabs/Makerfabs-ESP32-S3-SPI-TFT-with-Touch/blob/main/hardware/ESP32-
-  * S3%20TFT%20Touch%20v1.1(3.5'%20ili9488).PDF
-  *
-  */
- #define SDA_FT6236 38
- /**
-  * @brief Pin SCL asignado al panel táctil. Se obtuvo mirando los schematics de la placa
-  * de desarrollo:
-  * https://github.com/Makerfabs/Makerfabs-ESP32-S3-SPI-TFT-with-Touch/blob/main/hardware/ESP32-
-  * S3%20TFT%20Touch%20v1.1(3.5'%20ili9488).PDF
-  *
-  */
- #define SCL_FT6236 39
-
- class TS_Point
- {
- public:
-     TS_Point(void);
-     TS_Point(int16_t x, int16_t y, int16_t z);
-
-     bool operator==(TS_Point);
-     bool operator!=(TS_Point);
-
-     int16_t x;
-     int16_t y;
-     int16_t z;
- };
-
- class FT6236
- {
- public:
-     FT6236(void);
-     void debug(void);
-     boolean begin(uint8_t thresh = FT6236_DEFAULT_THRESHOLD, int8_t sda = -1, int8_t scl = -1);
-     uint8_t touched(void);
-     TS_Point getPoint(uint8_t n = 0);
-
- private:
-     void writeRegister8(uint8_t reg, uint8_t val);
-     uint8_t readRegister8(uint8_t reg);
-
-     void readData(void);
-     uint8_t touches;
-     uint16_t touchX[2], touchY[2], touchID[2];
- };
-
- #endif

```

- display.h

```

- /**
-  * @brief Fichero que contiene la clase que configura el driver del panel LCD con
-  * driver ILI988.
-  * Documentación de la librería:
-  * https://github.com/lovyan03/LovyanGFX/tree/master
-  * @file display.h
-  * @author Jesús Carmelo González Domínguez
-  * @date 14-07-2023
-  */
-
- #ifndef DISPLAY_GUARD
- #define DISPLAY_GUARD
-
- #include <LovyanGFX.hpp>
-
- class Display : public lgfx::LGFX_Device {
- // Tipo de panel LCD en uso ILI9488

```

```

- lgfx::Panel_ILI9488    _panel_instance;
- // Tipo de bus con el que se conecta la LCD
- lgfx::Bus_SPI         _bus_instance;
- // Control de brillo de la LCD
- lgfx::Light_PWM      _light_instance;
-
- public:
-   Display(void) {
-     // Configuración del bus SPI, para controlar la LCD
-     {
-       auto cfg          = _bus_instance.config();
-       cfg.spi_host      = SPI3_HOST;
-       cfg.spi_mode      = 0;
-       cfg.freq_write    = 40000000;
-       cfg.freq_read     = 16000000;
-       cfg.spi_3wire     = true;
-       cfg.use_lock      = true;
-       cfg.dma_channel   = 1;
-       cfg.pin_sclk      = 14;
-       cfg.pin_mosi      = 13;
-       cfg.pin_miso      = 12;
-       cfg.pin_dc        = 21;
-       // Carga la configuración para preparar el bus SPI
-       _bus_instance.config(cfg);
-       _panel_instance.setBus(&_bus_instance);
-     }
-     // Configuración del panel LCD
-     {
-       auto cfg          = _panel_instance.config();
-       cfg.pin_cs        = 15;
-       cfg.pin_rst       = -1;
-       cfg.pin_busy      = -1;
-       cfg.panel_width   = 320;
-       cfg.panel_height  = 480;
-       cfg.dummy_read_pixel = 8;
-       cfg.dummy_read_bits = 1;
-       cfg.readable      = true;
-       cfg.invert        = false;
-       cfg.rgb_order     = false;
-       cfg.dlen_16bit    = false;
-       //Ya que el bus está compartido con la SD.
-       cfg.bus_shared    = true;
-       // Carga la configuración para preparar el panel LCD
-       _panel_instance.config(cfg);
-     }
-     // Configuración del control de brillo de la LCD
-     {
-       auto cfg = _light_instance.config();
-       cfg.pin_bl = 48;
-       cfg.invert = false;
-       cfg.freq = 44100;
-       cfg.pwm_channel = 7;
-       // Carga la configuración para preparar el control de brillo
-       _light_instance.config(cfg);
-       _panel_instance.light(&_light_instance);
-     }
-     // Una vez realizados todos la configuración previa, terminamos de crear el objeto con setPanel()
-     setPanel(&_panel_instance);
-   }
- };
- #endif

```

```

- actuators.h
- /**
-  * @brief Fichero que contiene las definiciones de las funciones que controlan
-  * los relés y el dimmer/regulador de potencia.
-  * @file actuators.h
-  * @author Jesús Carmelo González Domínguez
-  * @date 14-07-2023
-  */

```

```

- #ifndef ACTUATORS_GUARD
- #define ACTUATORS_GUARD
-
- #include "Devices/rtc.h"
- #include "Devices/sensor.h"
- #include "UI/mcu_ui.h"
-
- /**
-  * @brief Pin del canal del relé que controla el bombillo UVB.
-  */
- #define UVB_RELAY_GPIO 4
- /**
-  * @brief Pin del canal del relé que controla el bombillo Plants.
-  */
- #define PLANTS_RELAY_GPIO 6
- /**
-  * @brief Pin del regulador de potencia que controla el bombillo UVA.
-  */
- #define DIMMER_GPIO 7
- /**
-  * @brief Pin del botón que apaga/enciende el brillo de la pantalla.
-  */
- #define BUTTON_GPIO 16
-
- /* Functions */
- // BUTTON
- /**
-  * @brief Función que configura el GPIO del botón en modo INPUT_PULLUP,
-  * con lo que se ahorra tener que poner una resistencia en el circuito
-  * electrónico.
-  */
- void configButtonGPIO(void);
-
- // Relays
- /**
-  * @brief Función que configura los GPIO de los relés como salidas.
-  */
- void configRelayGPIO(void);
- /**
-  * @brief Función que activa el relé del bombillo UVB.
-  * @param value Si value = true, se apaga. value= false, se enciende.
-  */
- void switchUVBRelay(const bool value);
- /**
-  * @brief Función que activa el relé del bombillo Plants.
-  * @param value Si value = true, se apaga. value= false, se enciende.
-  */
- void switchPlantsRelay(const bool value);
- /**
-  * @brief Función que controla el comportamiento del bombillo UVA, en
-  * base a una hora de encendido y apagado.
-  * @param on_hour Hora de encendido
-  * @param on_minute Minuto de encendido
-  * @param off_hour Hora de apagado
-  * @param off_minite Minuto de apagado
-  *
-  */
- void controlUVALight(const char* on_hour, const char* on_minute,
-                     const char* off_hour, const char* off_minute);
- /**
-  * @brief Función que controla el comportamiento del bombillo UVB, en
-  * base a una hora de encendido y apagado.
-  * @param on_hour Hora de encendido
-  * @param on_minute Minuto de encendido
-  * @param off_hour Hora de apagado
-  * @param off_minite Minuto de apagado
-  *
-  */
- void controlUVBLight(const char* on_hour, const char* on_minute,
-                     const char* off_hour, const char* off_minute);

```



```

-      /**
-   * @brief Función que controla el comportamiento del bombillo Plants, en
-   * base a una hora de encendido y apagado.
-   * @param on_hour Hora de encendido
-   * @param on_minute Minuto de encendido
-   * @param off_hour Hora de apagado
-   * @param off_minite Minuto de apagado
-   *
-   */
- void controlPlantsLight(const char* on_hour, const char* on_minute,
-                       const char* off_hour, const char* off_minute);
-
- // Dimmer
- /**
-   * @brief Función que configura el GPIO del módulo dimmer/regulador de potencia.
-   * - Se realiza a través de ledc la función creada por Espressif.
-   * - Se ha establecido una frecuencia de 60Hz, al ser esta la frecuencia operativa del
-   *   bombillo.
-   * - Para el ESP32-S3, hay 8 canales PWM, de 0 a 7.
-   * - Se ha escogido el canal 1. No se puede escoger el canal 7, está ocupado por el panel
-   *   LCD.
-   * - Para una frecuencia de 60Hz, es necesario tener una resolución mayor que 8, en este
-   *   caso, 10. Véase la explicación en:
-   *   https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/api-reference/peripherals/ledc.html#ledc-api-configure-timer
-   *
-   */
- void configDimmerGPIO(void);
- /**
-   * @brief Función que fija el valor de la señal PWM del bombillo UVA. Al tener una
-   * resolución de 10 bits, los valores pueden ir desde 0 a 1023.
-   * @param pwmvalue Valor PWM de 0 a 1023.
-   */
- void setPWM(const int pwm_value);
-
- #endif

```

```

- actuators.cpp
- #include "Devices/actuators.h"
- #include "Arduino.h"
-
- // Button
- void configButtonGPIO(void) {
-     pinMode(BUTTON_GPIO, INPUT_PULLUP);
- }
-
- // Relay
- void configRelayGPIO(void) {
-     pinMode(UVB_RELAY_GPIO, OUTPUT);
-     pinMode(PLANTS_RELAY_GPIO, OUTPUT);
- }
-
- void switchUVBRelay(const bool value) {
-     if (value == 1) {
-         digitalWrite(UVB_RELAY_GPIO, HIGH);
-     } else {
-         digitalWrite(UVB_RELAY_GPIO, LOW);
-     }
- }
-
- void switchPlantsRelay(const bool value) {
-     if (value == 1) {
-         digitalWrite(PLANTS_RELAY_GPIO, HIGH);
-     } else {
-         digitalWrite(PLANTS_RELAY_GPIO, LOW);
-     }
- }
-
- void controlUVALight(const char* on_hour, const char* on_minute,
-                    const char* off_hour, const char* off_minute) {

```

```

- DateTime now = rtc.now();
- int up_time = atoi(on_hour) * 100 + atoi(on_minute);
- int current_time = (int)now.hour() * 100 + (int)now.minute();
- int down_time = atoi(off_hour) * 100 + atoi(off_minute);
-
- if( up_time < current_time && current_time < down_time) {
-     setPWM(1020);
-     lv_obj_add_state(ui_uvaswitch, LV_STATE_CHECKED);
-
- } else {
-     setPWM(1);
-     lv_obj_clear_state(ui_uvaswitch, LV_STATE_CHECKED);
- }
-
- }
-
- void controlUVBLight(const char* on_hour, const char* on_minute,
-                     const char* off_hour, const char* off_minute) {
-     DateTime now = rtc.now();
-     int up_time = atoi(on_hour) * 100 + atoi(on_minute);
-     int current_time = (int)now.hour() * 100 + (int)now.minute();
-     int down_time = atoi(off_hour) * 100 + atoi(off_minute);
-
-     if( up_time < current_time && current_time < down_time) {
-         switchUVBRelay(false);
-         lv_obj_add_state(ui_uvbswitch, LV_STATE_CHECKED);
-     } else {
-         switchUVBRelay(true);
-         lv_obj_clear_state(ui_uvbswitch, LV_STATE_CHECKED);
-     }
-
- }
-
- void controlPlantsLight(const char* on_hour, const char* on_minute,
-                         const char* off_hour, const char* off_minute) {
-     DateTime now = rtc.now();
-     int up_time = atoi(on_hour) * 100 + atoi(on_minute);
-     int current_time = (int)now.hour() * 100 + (int)now.minute();
-     int down_time = atoi(off_hour) * 100 + atoi(off_minute);
-
-     if( up_time < current_time && current_time < down_time) {
-         switchPlantsRelay(false);
-         lv_obj_add_state(ui_plantsswitch, LV_STATE_CHECKED);
-     } else {
-         switchPlantsRelay(true);
-         lv_obj_clear_state(ui_plantsswitch, LV_STATE_CHECKED);
-     }
-
- }
-
- // Dimmer
- void configDimmerGPIO(void) {
-     ledcSetup(1,60, 10);
-     ledcAttachPin(DIMMER_GPIO, 1);
- }
-
- void setPWM(const int pwm_value) {
-     ledcWrite(1, pwm_value);
- }

```

8.2 Ficheros de configuración del servidor

- iot-stack – docker-compose.yml

```
version: "3.3"

services:

  mosquito:

    image: eclipse-mosquitto

    networks:

      - iot-network

    ports:

      - 30000:1883

      - 30001:9001

    volumes:

      - ./mosquitto/config:/mosquitto/config

      - ./mosquitto/data:/mosquitto/data

      - ./mosquitto/log:/mosquitto/log

  nodered:

    image: nodered/node-red

    container_name: nodered

    #user: "1000"

    security_opt:

      - no-new-privileges:true

    networks:

      - proxy-network

      - iot-network

    volumes:

      - ./nodered/data:/data

    labels:

      - "traefik.enable=true"

      - "traefik.http.routers.nodered.entrypoints=http"

      - "traefik.http.routers.nodered.rule=Host(`nodered.tfg-ull.cloud`)"

      - "traefik.http.middlewares.nodered-https-redirect.redirectscheme.scheme=https"

      - "traefik.http.routers.nodered.middlewares=nodered-https-redirect"

      - "traefik.http.routers.nodered-secure.entrypoints=https"

      - "traefik.http.routers.nodered-secure.rule=Host(`nodered.tfg-ull.cloud`)"

      - "traefik.http.routers.nodered-secure.tls=true"

      - "traefik.http.routers.nodered-secure.service=nodered"

      - "traefik.http.services.nodered.loadbalancer.server.port=1880"
```

- "traefik.docker.network=proxy-network"

influx:

image: influxdb

container_name: influxdb

security_opt:

- no-new-privileges:true

networks:

- proxy-network

- iot-network

volumes:

- ./influxdb/data:/var/lib/influxdb2

- ./influxdb/config:/etc/influxdb2

labels:

- "traefik.enable=true"

- "traefik.http.routers.influx.entrypoints=http"

- "traefik.http.routers.influx.rule=Host(`influx.tfg-ull.cloud`)"

- "traefik.http.middlewares.influx-https-redirect.redirectscheme.scheme=https"

- "traefik.http.routers.influx.middlewares=influx-https-redirect"

- "traefik.http.routers.influx-secure.entrypoints=https"

- "traefik.http.routers.influx-secure.rule=Host(`influx.tfg-ull.cloud`)"

- "traefik.http.routers.influx-secure.tls=true"

- "traefik.http.routers.influx-secure.service=influx"

- "traefik.http.services.influx.loadbalancer.server.port=8086"

- "traefik.docker.network=proxy-network"

grafana:

image: grafana/grafana

container_name: grafana

uid 0 para que el usuario dentro de grafana pueda

escribir dentro del volumen

#probar con uid 1000 que es el mismo que el de iot

security_opt:

- no-new-privileges:true

user: "1000"

networks:

- proxy-network

- iot-network

volumes:

- ./grafana/data:/var/lib/grafana

labels:

- "traefik.enable=true"
- "traefik.http.routers.grafana.entrypoints=http"
- "traefik.http.routers.grafana.rule=Host(`grafana.tfg-ull.cloud`)"
- "traefik.http.middlewares.grafana-https-redirect.redirectscheme.scheme=https"
- "traefik.http.routers.grafana.middlewares=grafana-https-redirect"
- "traefik.http.routers.grafana-secure.entrypoints=https"
- "traefik.http.routers.grafana-secure.rule=Host(`grafana.tfg-ull.cloud`)"
- "traefik.http.routers.grafana-secure.tls=true"
- "traefik.http.routers.grafana-secure.service=grafana"
- "traefik.http.services.grafana.loadbalancer.server.port=3000"
- "traefik.docker.network=proxy-network"

portainer:

image: portainer/portainer-ce

container_name: portainer

restart: unless-stopped

security_opt:

- no-new-privileges:true

networks:

- proxy-network

volumes:

- /var/run/docker.sock:/var/run/docker.sock:ro
- ./portainer/data:/data
- /etc/localtime:/etc/localtime:ro

labels:

- "traefik.enable=true"
- "traefik.http.routers.portainer.entrypoints=http"
- "traefik.http.routers.portainer.rule=Host(`portainer.tfg-ull.cloud`)"
- "traefik.http.middlewares.portainer-https-redirect.redirectscheme.scheme=https"
- "traefik.http.routers.portainer.middlewares=portainer-https-redirect"
- "traefik.http.routers.portainer-secure.entrypoints=https"
- "traefik.http.routers.portainer-secure.rule=Host(`portainer.tfg-ull.cloud`)"
- "traefik.http.routers.portainer-secure.tls=true"
- "traefik.http.routers.portainer-secure.service=portainer"
- "traefik.http.services.portainer.loadbalancer.server.port=9000"
- "traefik.docker.network=proxy-network"

networks:

iot-network:

```
proxy-network:
external: true
```

```
- trafik - docker-compose.yml
```

```
version: '3'
```

```
services:
```

```
traefik:
```

```
image: traefik:latest
```

```
container_name: traefik
```

```
restart: unless-stopped
```

```
security_opt:
```

```
- no-new-privileges:true
```

```
networks:
```

```
- proxy-network
```

```
ports:
```

```
- 80:80
```

```
- 443:443
```

```
environment:
```

```
- CF_API_EMAIL=alu0101267760@ull.edu.es
```

```
##- CF_DNS_API_TOKEN=
```

```
- CF_API_KEY="PONER API-KEY DE CLOUDFLARE"
```

```
# be sure to use the correct one depending on if you are using a token or key
```

```
volumes:
```

```
- /etc/localtime:/etc/localtime:ro
```

```
- /var/run/docker.sock:/var/run/docker.sock:ro
```

```
- /home/iot/IoT_Stack_TFG/traefik/data/traefik.yml:/traefik.yml:ro
```

```
- /home/iot/IoT_Stack_TFG/traefik/data/acme.json:/acme.json
```

```
- /home/iot/IoT_Stack_TFG/traefik/data/config.yml:/config.yml:ro
```

```
labels:
```

```
- "traefik.enable=true"
```

```
- "traefik.http.routers.traefik.entrypoints=http"
```

```
- "traefik.http.routers.traefik.rule=Host(`traefik.tfg-ull.cloud`)"
```

```
- "traefik.http.middlewares.traefik-auth.basicauth.users="PONER USER:PASSWORD"
```

```
- "traefik.http.middlewares.traefik-https-redirect.redirectscheme.scheme=https"
```

```
- "traefik.http.middlewares.sslheader.headers.customrequestheaders.X-Forwarded-Proto=https"
```

```
- "traefik.http.routers.traefik.middlewares=traefik-https-redirect"
```

```
- "traefik.http.routers.traefik-secure.entrypoints=https"
```

```
- "traefik.http.routers.traefik-secure.rule=Host(`traefik.tfg-ull.cloud`)"
```

- "traefik.http.routers.traefik-secure.middlewares=traefik-auth"
- "traefik.http.routers.traefik-secure.tls=true"
- "traefik.http.routers.traefik-secure.tls.certresolver=cloudflare"
- "traefik.http.routers.traefik-secure.tls.domains[0].main=tfg-ull.cloud"
- "traefik.http.routers.traefik-secure.tls.domains[0].sans=*.tfg-ull.cloud"
- "traefik.http.routers.traefik-secure.service=api@internal"

networks:

 proxy-network:

external: true

Bibliografía

- [1] <https://trends.google.es/trends/explore?date=all&q=IoT&hl=es>
- [2] <https://www.geeksforgeeks.org/3-layer-iot-architecture/>
- [3] <https://www.mdpi.com/2078-2489/12/2/87>
- [4] <https://es.wikipedia.org/wiki/Wifi>
- [5] <https://es.wikipedia.org/wiki/Bluetooth>
- [6] [https://es.wikipedia.org/wiki/LTE_\(telecomunicaciones\)](https://es.wikipedia.org/wiki/LTE_(telecomunicaciones))
- [7] <https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/>
- [8] <https://es.wikipedia.org/wiki/Zigbee>
- [9] <https://es.wikipedia.org/wiki/LoRa>
- [10] <https://es.wikipedia.org/wiki/Ethernet>
- [11] <https://es.wikipedia.org/wiki/LoRaWAN>
- [12] https://es.wikipedia.org/wiki/Advanced_Message_Queueing_Protocol
- [13] https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto
- [14] <https://es.wikipedia.org/wiki/CoAP>
- [15] <https://www.dds-foundation.org/what-is-dds-3/>
- [16] <https://www.paessler.com/es/it-explained/mqtt>
- [17] <https://github.com/lovyan03/LovyanGFX>
- [18] <https://github.com/DustinWatts/FT6236/tree/main>
- [19] <https://docs.lvgl.io/8.3/index.html>
- [20] <https://www.freertos.org/a00106.html>
- [21] <https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/src/WiFi.h>
- [22] <https://github.com/marvinroger/async-mqtt-client>
- [23] <https://github.com/bblanchon/ArduinoJson>
- [24] <https://github.com/adafruit/RTCLib>
- [25] <https://github.com/milesburton/Arduino-Temperature-Control-Library/tree/master>
- [26] <https://github.com/adafruit/DHT-sensor-library>
- [27] <https://www.elecrow.com/blog/why-is-it-so-hard-to-buy-raspberry-pi-these-days.html>
- [28] <https://www.portainer.io/>

- [29] <https://doc.traefik.io/traefik/>
- [30] <https://mosquitto.org/>
- [31] <https://nodered.org/>
- [32] <https://www.influxdata.com/>
- [33] <https://grafana.com/>
- [34] <https://github.com/Makerfabs/Makerfabs-ESP32-S3-SPI-TFT-with-Touch/tree/main/hardware>
- [35] <https://platformio.org/>
- [36] <https://www.digikey.com/en/maker/projects/introduction-to-rtos-solution-to-part-4-memory-management/6d4dfcaa1ff84f57a2098da8e6401d9c>
- [37] <https://squareline.io/>
- [38] <https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/>
- [39] <http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/>
- [40] <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/ledc.html>
- [41] <https://www.hetzner.com/cloud>

