

CARRERA DE ESPECIALIZACIÓN EN INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

Sistema de monitoreo y control de ambientes a distancia

Autor:
Ing. César Javier Fanelli

Director:
Ing. Fernando Lichtschein (FIUBA)

Codirector:
Ing. María Celeste Corominas (FIUBA)

Jurados:
Esp. Ing. Pedro Rosito (FIUBA)
Ing. Marcelo Edgardo Romeo (UNSAM)
Mg. Lic. Leopoldo Zimperz (FIUBA)

*Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires,
entre febrero de 2023 y diciembre de 2023.*

Resumen

En la presente memoria se describe el diseño de un prototipo de sistema integral de domótica de índole académica. El propósito es aplicar los conocimientos adquiridos en el transcurso del posgrado. En esta primera etapa se implementan dos tipos de nodos y se agregarán a futuro nuevos elementos con otras funcionalidades.

En el desarrollo del trabajo se utilizaron conocimientos de diseño de páginas web, desarrollo de código backend para el servidor, implementación de bases de datos, desarrollo de sistemas embebidos, ciberseguridad y diseño electrónico para el hardware adicional.

Agradecimientos

Agradezco a docentes, alumnos y no docentes que conforman el posgrado, así como a los directores que me acompañaron en el desarrollo de este trabajo. En especial agradezco a mi esposa e hija, quienes fueron mi apoyo en todo momento y sin quienes no habría podido lograr la hazaña de emprender este viaje arduo y satisfactorio.

Índice general

Resumen	I
1. Introducción general	1
1.1. Hogares inteligentes	1
1.1.1. Aplicaciones comunes	1
1.2. Motivación	2
1.3. Estado del arte	2
1.4. Objetivos y alcance	3
2. Introducción específica	5
2.1. Protocolos de comunicación	5
2.1.1. Modelo OSI	5
2.1.2. Protocolo HTTP	6
2.1.3. Protocolo MQTT	7
Calidad de servicio	8
Protocolos SSL y TLS	8
2.2. Componentes de hardware	8
2.2.1. Raspberry Pi	8
Especificaciones técnicas de la Raspberry Pi 400	9
2.2.2. Sistema en chip	10
Familia ESP32 [13]	10
2.2.3. Especificaciones de los sensores	11
2.2.4. Especificaciones de los actuadores	12
2.3. Herramientas de software	13
2.3.1. Visual studio code [21]	13
2.3.2. Marco de desarrollo ESP	13
2.3.3. Lenguajes JavaScript y TypeScript	13
2.3.4. Angular y Ionic	14
2.3.5. Bases de datos relacionales MySQL y MariaDB	14
2.3.6. Contenedores con Docker	15
3. Diseño e implementación	17
3.1. Arquitectura del sistema	17
3.2. Modelo de datos	18
3.2.1. Tabla Dispositivos	19
3.2.2. Tabla Usuarios	20
3.2.3. Tabla Mediciones	20
3.3. Desarrollo del frontend	21
3.3.1. Rutas y páginas destacadas	23
Pantalla de <i>login</i>	23
Pantalla principal <i>home</i>	24
Pantalla de creación de dispositivo nuevo	24

Pantalla de configuración de modo	25
Pantallas de mediciones y gráfico	26
3.4. Desarrollo del backend	28
3.4.1. Conexión a la base de datos	28
3.4.2. Configuración de la conexión <i>mqtt</i>	29
3.4.3. Configuración del envío de mails	30
3.4.4. API y rutas	30
3.5. Nodos, sensores y actuadores	31
3.5.1. Características de los nodos	31
3.5.2. Menús y pantallas	33
3.5.3. Modelo de programación	34
3.6. Comunicación del sistema	36
3.6.1. <i>Topics MQTT</i> utilizados	36
4. Ensayos y resultados	37
4.1. Banco de pruebas	37
4.2. Metodología empleada	39
4.2.1. Pruebas del frontend	39
4.2.2. Pruebas del backend	40
4.2.3. Pruebas de los nodos	41
4.3. Resultados finales	42
4.4. Comparación con el estado del arte	45
5. Conclusiones	47
5.1. Resultados obtenidos	47
5.2. Trabajo futuro	47
A. Ejemplo de creación de menú en ESP32	49
Bibliografía	53

Índice de figuras

1.1. Ejemplo de sistema de domótica	2
1.2. Matter y Home Assistant	3
1.3. Esquema básico del sistema desarrollado	4
2.1. Modelo OSI	5
2.2. Arquitectura MQTT	7
2.3. Raspberry Pi 4 y Raspberry Pi 400	9
2.4. Módulos de la familia ESP32	11
2.5. Sensores utilizados	12
2.6. Actuadores utilizados	13
2.7. Contenedor Docker	15
3.1. Arquitectura cliente-servidor	17
3.2. Configuración de Docker de la base de datos	19
3.3. Tabla de dispositivos	19
3.4. Tabla de usuarios	20
3.5. Tabla de mediciones	21
3.6. Configuración de Docker de Ionic	21
3.7. Estructura de archivos de página	22
3.8. Pantalla de login	23
3.9. Pantalla de home	24
3.10. Pantalla de creación de dispositivo nuevo	24
3.11. Pantalla de configuración de modo	26
3.12. Pantalla de medición gráfica y estado	26
3.13. Pantalla de mediciones del dispositivo de temperatura	27
3.14. Pantalla de presentación gráfica de mediciones	27
3.15. Ubicación de los certificados SSL	31
3.16. Diagrama en bloques del nodo de temperatura	32
3.17. Diagrama en bloques del nodo de dimerización	32
3.18. Placa del módulo principal	32
3.19. Pantallas principales del nodo	33
4.1. Componentes del nodo de temperatura	37
4.2. Componentes del nodo de dimerización	38
4.3. Componentes del servidor	38
4.4. Pantalla de aplicación y consola del frontend	40
4.5. Mensaje por consola del servidor en el backend	41
4.6. Pantalla de MQTDX	41
4.7. Dimmer en modo automático funcionando	42
4.8. Pantalla de configuración del dimmer en modo automático	43
4.9. Pantalla de configuración de dimmer	43
4.10. Valores de dispositivo nuevo al agregar dispositivo	44
4.11. Valores de usuario modificados	44

4.12. Envío de alarma de medición	45
---	----

Índice de tablas

1.1. Comparativa entre las distintas opciones del mercado nacional	3
2.1. Especificaciones de Raspberry Pi 400	10
2.2. Especificaciones técnicas del módulo ESP32	11
3.1. Rutas de la aplicación	23
4.1. Comparativa entre las distintas opciones - Estado del arte	46

Dedicado a mi hija Sofía.

Capítulo 1

Introducción general

1.1. Hogares inteligentes

La domótica es la aplicación de la tecnología a la automatización del hogar, que se utiliza para controlar y gestionar diferentes sistemas y dispositivos, con el fin de aportar seguridad, bienestar y confort. Estos sistemas pueden abarcar la iluminación, la calefacción, el aire acondicionado, los sistemas de seguridad y cámaras de vigilancia, los sistemas de entretenimiento y otros dispositivos domésticos [1].

Este sector, como muchos otros que utilizan tecnología ha estado creciendo, al punto que algunas de las viviendas modernas son concebidas como hogares inteligentes desde su edificación, llegando a tener edificios completos con este tipo de soluciones instaladas.

1.1.1. Aplicaciones comunes

Los servicios que ofrece la domótica se pueden agrupar según cinco aspectos o ámbitos principales [2]:

- Programación y ahorro energético: el ahorro energético se puede lograr de muchas maneras. En muchos casos no es necesario sustituir los aparatos o sistemas del hogar por otros que consuman menos energía, sino una gestión eficiente de los mismos.
- Confort: conlleva todas las actuaciones que se puedan llevar a cabo que mejoren la comodidad en una vivienda. Dichas actuaciones pueden ser de carácter tanto pasivo, como activo o mixtas.
- Seguridad: consiste en una red encargada de proteger tanto los bienes patrimoniales, como la seguridad personal y la vida.
- Comunicaciones: son los sistemas o infraestructuras de comunicaciones que posee el hogar.
- Accesibilidad: bajo este mecanismo se incluyen las aplicaciones o instalaciones de control remoto del entorno que favorecen la autonomía personal de personas con limitaciones funcionales, o discapacidad.

El presente trabajo puede encuadrarse dentro de la programación y ahorro energético, confort y accesibilidad. En la figura 1.1 puede verse una imagen con los distintos tipos de dispositivos que pueden estar conectados a un sistema de domótica.

¹Imagen tomada de: <https://intelligy.com/blog/2018/03/12/conoce-la-domotica/>

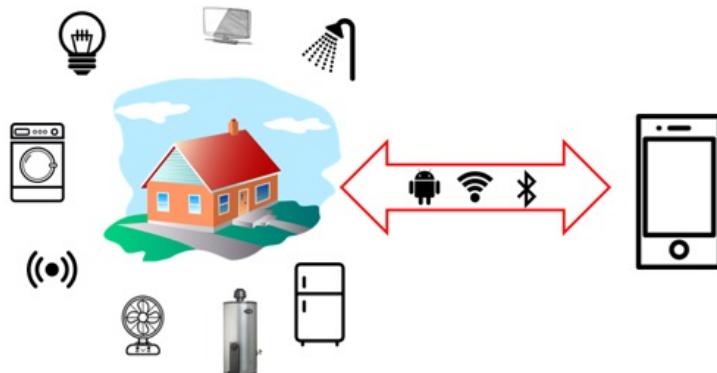


FIGURA 1.1. Ejemplo de sistema de domótica.¹

1.2. Motivación

El uso de la tecnología para facilitar y mejorar la vida de las personas se está implementando en todo el mundo en diversos ámbitos, creando soluciones complejas e innovadoras. Los electrodomésticos e instalaciones se fabrican con posibilidades de conexión y capacidades cada vez más amplias, abarcando funcionalidades complejas que aportan al bienestar y confort de las personas.

Este proyecto nace como mejora y actualización de la tesis de grado de ingeniería, en la cual se creó un sistema similar que carecía de conectividad a Internet y utilizando tecnología que al día de hoy es obsoleta. Es por este motivo que se ha decidido hacer un proyecto académico con esta temática, pudiendo crear un sistema integral con una página web, base de datos que almacene mediciones y utilizando hardware actualizado.

1.3. Estado del arte

En el mercado actual existe una gran cantidad de sistemas de domótica, algunos privados o bajo licencias propietarias y otros con licencias de código abierto, nacionales e internacionales. Aquellas soluciones de código abierto tienen los repositorios públicos en GitHub al alcance de todos para ser descargados, implementados y hasta modificados.

Del mercado internacional, podemos nombrar 2 importantes variantes:

- Matter: es un estándar de conectividad de código abierto creado por la *Connectivity standards alliance*. Es de los más importantes a nivel global y participan empresas como Amazon, Apple, Google, Huawei entre muchas otras. Ha crecido y tomado mucha fuerza durante el corto tiempo de vida que tiene el proyecto [3].
- Home assistant: software de automatización de hogares de código abierto que prioriza el control local y la privacidad. Desarrollado por una comunidad mundial de entusiastas de autodidactas y profesionales con un sentido propio [4].

En la figura 1.2 se observa un kit de Matter a la izquierda y la pantalla de la aplicación de Home Assistant a la derecha.

FIGURA 1.2. Matter y Home Assistant.²

A nivel nacional, existen empresas que se dedican al desarrollo de sistemas de domótica de forma privada:

- Domotic [5].
- Commax [6].
- Reactor [7].

Todos estos sistemas utilizan distintos tipos de conexión como Zigbee o Wi-Fi, mensajes entre dispositivos como HTTP, MQTT y otros, y pueden alojar sus plataformas en un servidor local o en la nube.

Sin embargo, pocos de estos sistemas tienen la posibilidad de funcionar con mandos desde el lugar o sin conexión con el servidor, es decir, que sean independientes y controlables de forma local. Este proyecto tiene tal posibilidad, ya que la página web es una forma más de cambiar los parámetros de los nodos y está diseñado para decidir si cada terminal posee comandos o no.

En la tabla 1.1 pueden verse las principales características de las empresas nacionales con soluciones similares.

TABLA 1.1. Comparativa entre las distintas opciones.

Empresa	Productos	Compatibilidad
Domotic	Catálogo amplio	No especifica
Commax	Catálogo amplio	Smart things y Apple Home Kit
Reactor	2 alternativas	Google Assistant e IFTTT

1.4. Objetivos y alcance

El objetivo de este trabajo es aplicar todos los conocimientos adquiridos a lo largo del posgrado en un área de interés particular, que es la domótica. En especial, el principal propósito es desarrollar un prototipo que sirva como punto de partida para un sistema más amplio y avanzado, con la particularidad de que en un futuro cercano se pueda llegar a analizar la opción de integrarlo con sistemas como *Matter* o *Home Assistant*.

En este desarrollo se incluyó:

²Imágenes tomadas de: <https://csa-iot.org/> y <https://www.home-assistant.io/>

- El esquema de conexiones de las placas de ESP32 con los módulos de display, encoder, driver LED y driver de calefacción.
- El diseño en una placa de desarrollo del circuito del driver de corriente continua de la iluminación LED.
- El diseño en una placa de desarrollo del circuito del driver de corriente alterna de la calefacción.
- La programación de un firmware de la placa ESP32 para la comunicación y manejo de conexiones.
- La creación del software frontend, backend y la base de datos que almacena toda la información dentro del servidor.
- Las conexiones, instalación y configuración del servidor montado en una Raspberry Pi.

En la figura 1.3 se puede ver un esquema básico del sistema desarrollado.

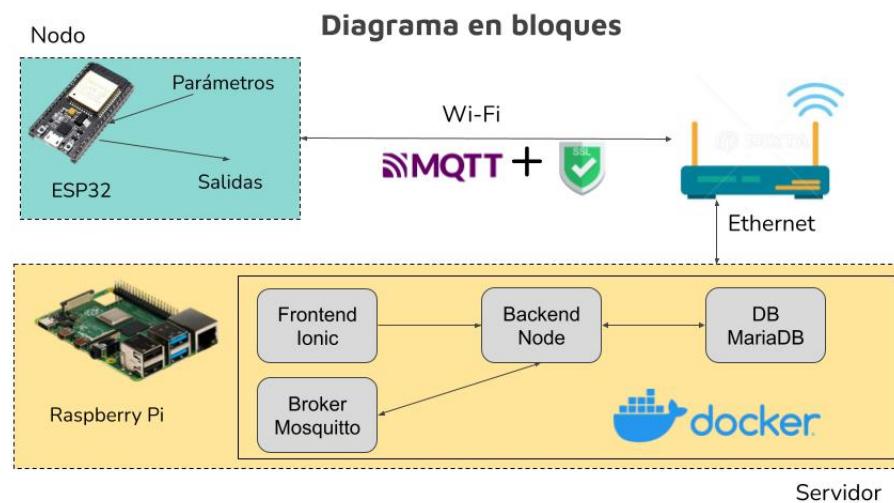


FIGURA 1.3. Esquema básico del sistema desarrollado.

Capítulo 2

Introducción específica

En el presente capítulo se introducen las tecnologías y herramientas de hardware y software utilizados en el desarrollo del trabajo.

2.1. Protocolos de comunicación

2.1.1. Modelo OSI

El modelo de interconexión de sistemas abiertos, conocido como modelo OSI (en inglés, *Open Systems Interconnection*), es un modelo de referencia para los protocolos de la red. Define un estándar que tiene por objetivo interconectar sistemas de distinta procedencia para que estos puedan intercambiar información sin ningún tipo de impedimentos. Está conformado por 7 capas o niveles de abstracción. Cada uno de estos niveles tiene sus propias funciones para que en conjunto sean capaces de poder alcanzar su objetivo final. Precisamente esta separación en niveles hace posible la intercomunicación de protocolos distintos al concentrar funciones específicas en cada nivel de operación [8].

En la figura 2.1 pueden verse de forma gráfica las distintas capas del modelo OSI.

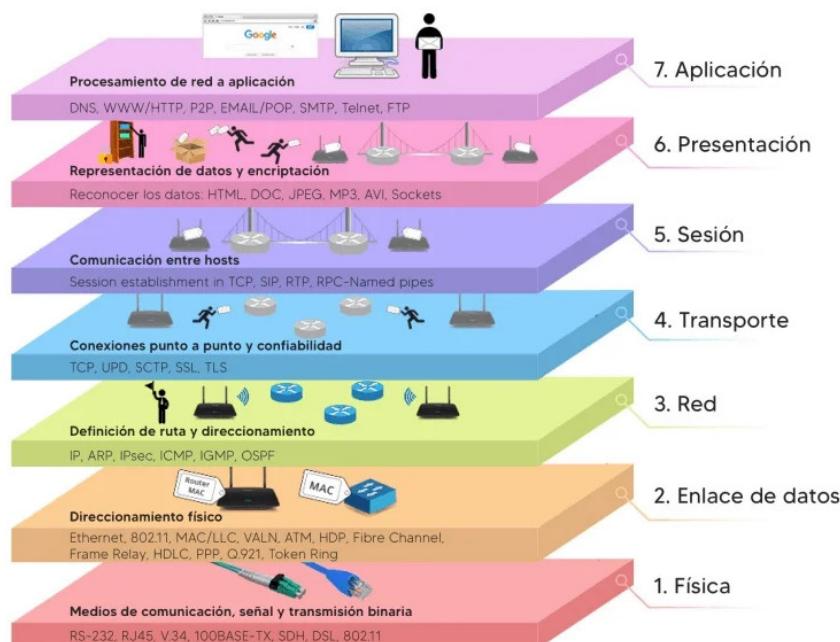


FIGURA 2.1. Modelo OSI.¹

A continuación se presentan en detalle las funciones de cada capa:

- Capa 1 o física: define todas las especificaciones eléctricas y físicas de los dispositivos.
- Capa 2 o de enlace de datos: proporciona direccionamiento físico y procedimientos de acceso a medios.
- Capa 3 o de red: se encarga del direccionamiento lógico y el dominio del enrutamiento.
- Capa 4 o de transporte: proporciona transporte confiable y control del flujo a través de la red.
- Capa 5 o de sesión: establece, administra y finaliza las conexiones entre las aplicaciones locales y las remotas.
- Capa 6 de presentación: transforma el formato de los datos y proporciona una interfaz estándar para la capa de aplicación.
- Capa 7 o de aplicación: responsable de los servicios de red para las aplicaciones.

2.1.2. Protocolo HTTP

El protocolo de transferencia de hipertexto, conocido como HTTP (en inglés *Hypertext Transfer Protocol*), es el protocolo de comunicación que permite las transferencias de información a través de archivos como XML y HTML entre otros. Está orientado a transacciones y sigue el esquema petición y respuesta entre un cliente y un servidor. El cliente realiza una petición al servidor enviando un mensaje con un formato determinado. El servidor le envía un mensaje de respuesta con la información solicitada o un mensaje de error [9].

Los mensajes HTTP son en texto plano y tienen la siguiente estructura:

- Línea inicial.
- Cabecera.
- Cuerpo.

En la línea inicial, se envían las peticiones con el método hacia el servidor o las respuestas con el código devueltas hacia el navegador. Los métodos más comunes, y utilizados en el proyecto, son:

- GET: solicita una representación del recurso especificado.
- POST: envía datos para que sean procesados por el recurso identificado en la URL de la línea de petición.
- PUT: envía datos al servidor. La diferencia con POST es que este está orientado a la creación de nuevos contenidos, mientras que PUT está orientado a la actualización de los mismos.
- DELETE: Borra el recurso especificado.

¹Imagen tomada de: <https://platzi.com/clases/2225-redes/35587-modelo-osi/>

2.1.3. Protocolo MQTT

El protocolo de transporte de telemetría de colas de mensajes, conocido como MQTT (en inglés *Message Queue Telemetry Transport*) es uno de los más utilizados y difundidos en IoT (Internet de las cosas, o en inglés *Internet of Things*). Es un protocolo de capa de aplicación, posee una topología de estrella, y sus mensajes se transmiten como colas de publicación/suscripción.

El nodo central de su topología se llama *broker* y es al cual se conectan los clientes remotos. El *broker* se encarga de gestionar la red, recibir todos los mensajes de los clientes y redirigirlos hacia los clientes de destino. Un cliente es cualquier dispositivo que pueda interactuar con el *broker* para enviar y recibir mensajes. Puede ser un sensor de IoT en campo o una aplicación en un centro de datos que procesa datos provenientes de los sensores. Cualquier cliente puede publicar o suscribirse a *topics* (temas) para acceder a la información. Un *topic* se representa mediante una cadena de texto con una estructura jerárquica. Cada jerarquía se separa con el carácter '/'.

La finalidad de MQTT es minimizar el uso recursos en los dispositivos (CPU, RAM, ROM), ser confiable y ofrecer distintos niveles calidad del servicio. Consumo un ancho de banda relativamente bajo y mantiene una conexión continua entre el broker y los clientes. Una de las grandes ventajas de MQTT es que permite desacoplar los nodos que producen la información de aquellos que la van a utilizar, tanto a nivel de datos como de aplicaciones. Un nodo publica y se pueden suscribir una cantidad de nodos independiente a cada *topic*, eso simplifica la arquitectura y la hace más flexible. Generalmente se suele usar este protocolo como nexo entre los dispositivos de campo con elementos típicos de software como servidores webs, bases de datos o herramientas de análisis. En la figura 2.2 se puede ver una arquitectura básica de conexión MQTT [10].

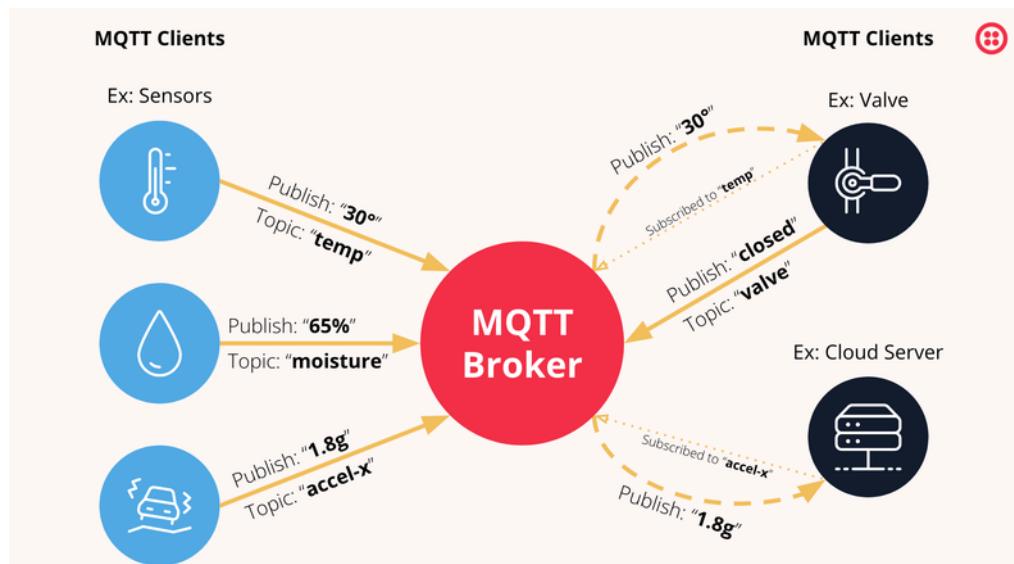


FIGURA 2.2. Arquitectura MQTT.²

Para reforzar la seguridad en la comunicación mediante MQTT, se pueden proteger los datos mediante el uso de usuario y contraseña, o con cifrado de capa

²Imagen tomada de: <https://www.twilio.com/blog/what-is-mqtt>

de sockets seguros, conocidos como SSL (en inglés *Secure Sockets Layer*). La capa de transporte segura (en inglés *Transport Layer Security* o TLS), es una versión actualizada y más segura de SSL, y es por esto que en la actualidad se refiere a SSL/TLS como sinónimos.

Calidad de servicio

MQTT está diseñado para ser simple y minimizar el ancho de banda, lo que lo convierte en una buena opción para muchos escenarios, aunque esta simplicidad hace que la interpretación de los mensajes dependa completamente del diseñador del sistema. Para mitigar este tipo de inconvenientes se soportan distintos niveles de calidad de servicio.

Estos niveles determinan cómo se entrega cada mensaje y es necesario especificar un QoS (siglas en inglés para *quality of service*) para cada *topic* MQTT. Es importante elegir el valor de QoS adecuado para cada mensaje que está determinado de la siguiente manera:

- QoS 0 (a lo sumo una vez): los mensajes se mandan sin tener en cuenta si llegan o no. Puede haber pérdida de mensajes. No se hacen retransmisiones.
- QoS 1 (al menos una vez): se asegura que los mensajes lleguen pero se pueden producir duplicados. El receptor recibe el mensaje por lo menos una vez. Si el receptor no confirma la recepción del mensaje o se pierde en el camino se reenvía el mensaje hasta que recibe por lo menos una confirmación.
- QoS 2 (exactamente una vez): se asegura que el mensaje llegue exactamente una vez. Eso incrementa la sobrecarga en la comunicación pero es la mejor opción cuando la duplicación de un mensaje no es aceptable.

Protocolos SSL y TLS

SSL y TLS son protocolos criptográficos, que proporcionan comunicaciones seguras por una red. Se usa criptografía asimétrica para autenticar a la contraparte con quien se está comunicando y para intercambiar una llave simétrica, iniciando una sesión entre las partes intervinientes. Esta sesión es luego usada para cifrar el flujo de datos entre las partes. Esto permite la confidencialidad de la información transmitida y la autenticación de los mensajes.

Antes de que un cliente y el servidor pueden empezar a intercambiar información protegida por TLS, deben intercambiar en forma segura o acordar una clave de cifrado y una clave para usar cuando se cifren los datos. Existen varios métodos utilizados para el intercambio y acuerdo de claves [11].

2.2. Componentes de hardware

2.2.1. Raspberry Pi

Las Raspberry Pi son computadoras de placa simple (en inglés *Single Board Computer* o su sigla SBC). Actualmente en el mercado existen distintas marcas y variantes de este tipo de placas, siendo elegidas tanto como computadoras de escritorio como para algunas aplicaciones específicas. A continuación, se describen las principales características de la familia Raspberry Pi:

- Posee distintos puertos, permitiendo conectarla a periféricos como pueden ser una pantalla, un medio de almacenamiento e incluso cualquier dispositivo que tenga interfaz USB.
- Contiene un procesador gráfico integrado, lo que permite la reproducción de vídeo, incluso en alta definición.
- Permite la conexión a la red a través del puerto de Ethernet y algunos modelos permiten conexión Wi-Fi y Bluetooth.
- Consta de una ranura microSD que permite instalar y ejecutar sistemas operativos a través de una tarjeta de memoria.

El diseño de la Raspberry Pi fue evolucionando con el correr del tiempo. En la actualidad se encuentran los modelos Raspberry Pi 4 y Raspberry Pi 400, siendo los más modernos y robustos lanzados por la marca. La familia de Raspberry Pi 4 cuenta con 4 modelos de placa con distinta capacidad de memoria RAM, yendo desde 1 GB hasta 8 GB. La Raspberry Pi 400 es una variante de la Raspberry Pi 4. Además, existen otros modelos más pequeños diseñados para otro tipo de aplicaciones.

En la figura 2.3 se pueden ver los modelos Raspberry Pi 4 y Raspberry Pi 400.



FIGURA 2.3. Raspberry Pi 4 y Raspberry Pi 400.³

Algunas de las funciones y aplicaciones más comunes de este tipo de computadoras se describen a continuación:

- Navegar en la red, utilizar aplicaciones de oficina para la edición de documentos y emplearla como si fuese una computadora de escritorio.
- Crear un centro multimedia y ver los archivos guardados en su memoria.
- Se puede utilizar como un servidor privado dentro de una red local.
- Conectar los puertos del microprocesador desde un conector de 40 pines a distintos circuitos y dispositivos.

Especificaciones técnicas de la Raspberry Pi 400

El sistema operativo del servidor está instalado y se ejecuta desde un disco de estado sólido por USB. Este tipo de discos tienen una mayor capacidad de escrituras y lecturas que una memoria microSD, lo que resulta favorable al momento

³Imagen tomada de: <https://all3dp.com/2/raspberry-pi-400-vs-raspberry-pi-4-differences/>

de hacer modificaciones y pruebas de ejecución de software. En la versión final del sistema todo el software estará instalado en una tarjeta microSD para que todo el conjunto sea lo más pequeño posible.

En la tabla 2.1 pueden verse las especificaciones técnicas de hardware más importantes del modelo Raspberry Pi 400.

TABLA 2.1. Especificaciones de Raspberry Pi 400.

Procesador	Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8 GHz
Memoria	4GB LPDDR4-3200
Conectividad	2.4 GHz and 5.0 GHz 802.11b/g/n/ac wireless LAN Bluetooth 5.0, BLE Gigabit Ethernet
Alimentación	5 V DC vía USB-C

2.2.2. Sistema en chip

Un sistema en chip o SoC (del inglés *System on a Chip*) es aquel dispositivo que posee integrados todos o gran parte de los módulos que componen un sistema informático o electrónico en un único circuito integrado o chip. El diseño de estos sistemas puede estar basado en circuitos de señal digital, señal analógica y a menudo módulos o sistemas de radiofrecuencia. Un ámbito común de aplicación de la tecnología SoC son los sistemas embebidos.

Un SoC estándar está constituido por [12]:

- Un microcontrolador con el núcleo de la CPU. Algunos son construidos con microproycesadores dotados de varios núcleos.
- Módulos de memoria ROM (memoria de sólo lectura), RAM (memoria de acceso aleatorio), EEPROM (memoria de sólo lectura programable y borrable electrónicamente) y Flash (memorias de acceso muy rápido).
- Generadores de frecuencia fija.
- Componentes periféricos como contadores, temporizadores y relojes en tiempo real o RTC (en inglés *Real Time Clock*).
- Controladores de comunicación con interfaces externas normalmente estándar como USB, Ethernet, UART, o SPI.
- Controladores de interfaces analógicas, incluyendo conversores analógico a digital (ADC) y digital a analógico (DAC).
- Reguladores de tensión y circuitos de gestión eficaz de la energía.

Familia ESP32 [13]

ESP32 es la denominación de una familia de chips SoC de bajo costo y consumo de energía, con tecnología Wi-Fi y Bluetooth de modo dual integrada. Emplea un microproycesador Tensilica Xtensa LX6 en sus variantes de simple y doble núcleo

e incluye interruptores de antena, balun de radiofrecuencia, amplificador de potencia, amplificador receptor de bajo ruido, filtros y módulos de administración de energía. El ESP32 fue creado y desarrollado por Espressif Systems.

En la figura 2.4 se pueden ver algunos de los módulos de desarrollo que contienen ESP32.

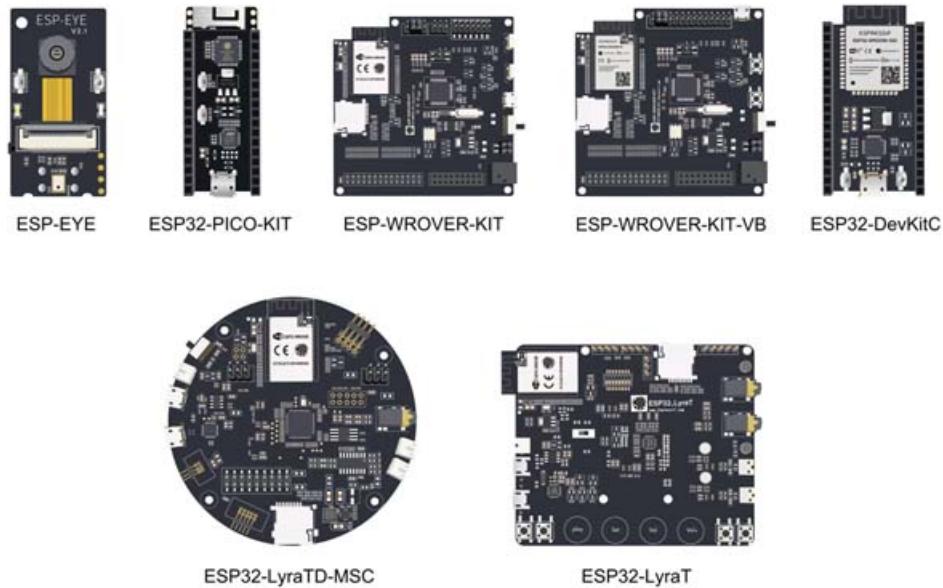


FIGURA 2.4. Módulos de la familia ESP32.⁴

En la tabla 2.2 pueden verse las características de hardware más importantes del microcontrolador ESP32, utilizado para el desarrollo de los dispositivos.

TABLA 2.2. Especificaciones técnicas del módulo ESP32 [14].

Característica	ESP32
Núcleo	Xtensa® dual-core 32-bit LX6 @240 MHz
Flash	0 MB, 2 MB o 4 MB (dependiendo la versión)
Protocolo Wi-Fi	802.11 b/g/n, 2.4 GHz

2.2.3. Especificaciones de los sensores

A continuación, se listan sensores utilizados con sus principales características:

- DHT22: sensor de temperatura y humedad relativa ambiente [15].
 - Rango de temperatura: -40 a 80 grados Celsius.
 - Resolución: 0,1 grado Celsius.
 - Comunicación: serie, bus de 1 conductor, 40 bits por trama.
- KY-040: encoder rotativo con interruptor. 20 pulsos por vuelta [16].

⁴Imagen tomada de: <https://www.electrodaddy.com/esp32/>

En la figura 2.5 se muestra la imagen del encoder a la izquierda y del sensor de temperatura a la derecha con sus correspondientes esquemas de conexión.

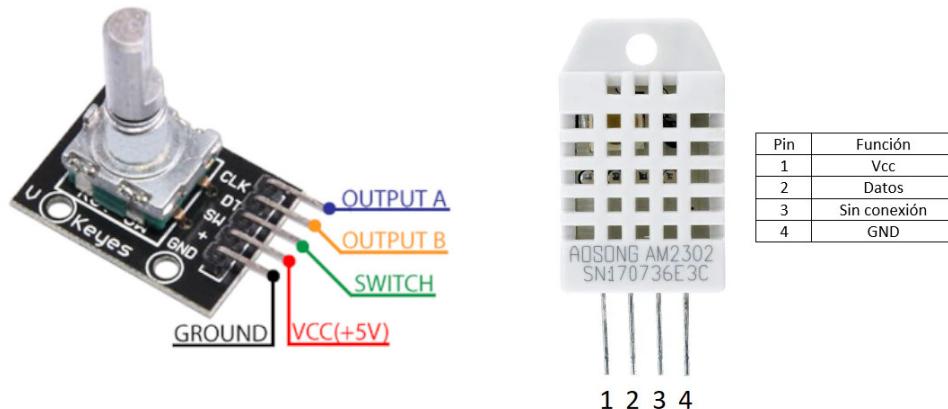


FIGURA 2.5. Sensores utilizados.

2.2.4. Especificaciones de los actuadores

- SSD1306: display OLED 1,2 pulgadas [17].
 - Resolución: 128x64 píxeles.
 - Interfaz: I2C.
- Control de potencia para 220 V: módulo con aislación y salida de triac. Se utilizó un triac BT137 [18] y un optoacoplador MOC3041 [19].
 - Tipo: encendido - apagado (on - off).
 - Carga máxima: 8 A.
 - Tipo de aislación: optoacoplador.
 - Tensión máxima de aislación: 7500 V AC pico, 1 segundo de duración.
- Control de intensidad de luz: módulo de control para iluminación LED de corriente continua implementado con modulación por ancho de pulso (PWM). Se utilizó un transistor BC337 [20].
 - Tipo: PWM y encendido - apagado (on - off).
 - Carga máxima: 800 mA CC.
 - Tensión de alimentación: 5 a 24 V CC.

En la figura 2.6 se muestra la imagen del circuito de potencia para la calefacción montado en una caja estanca a la izquierda, y del control por ancho de pulso para la dimerización a la derecha con sus correspondientes esquemas de conexión. Ambos circuitos fueron montados sobre una placa perforada de desarrollo para hacer las pruebas de funcionamiento.

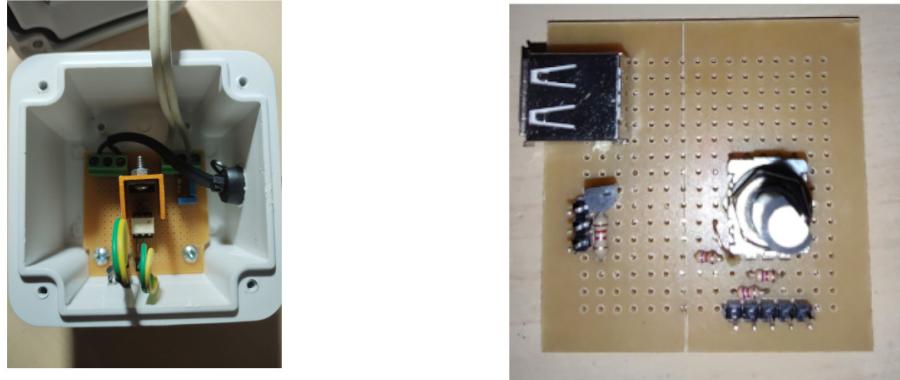


FIGURA 2.6. Actuadores utilizados.

2.3. Herramientas de software

2.3.1. Visual Studio Code [21]

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para múltiples sistemas operativos. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. Es personalizable y tiene la posibilidad de instalar extensiones para agregar lenguajes, depuradores y herramientas para el desarrollo de código. Es gratuito y de código abierto, aunque la descarga oficial está bajo software privativo e incluye características personalizadas por Microsoft.

Algunos de los lenguajes de programación que soporta son: C, C++, Dockerfile, Git-commit, HTML, JSON, Java, JavaScript, PHP, Python, Ruby, Rust, SQL, Shell script, TypeScript y Visual Basic, entre otros.

2.3.2. Marco de desarrollo ESP

El marco de desarrollo de ESP, o ESP-IDF (en inglés *ESP IoT Development Framework*) es un entorno completo de programación para desarrollar sistemas embedidos para dispositivos ESP. Es desarrollado por Espressif y se puede descargar como una extensión de Visual studio code. El lenguaje de programación es C e incluye herramientas para cargar el código desarrollado al chip y depurar el programa en tiempo real.

El lenguaje C es un lenguaje de programación de propósito general de tipos de datos estáticos, débilmente tipado. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a bajo nivel. Uno de los objetivos de diseño del lenguaje C es que solo sean necesarias unas pocas instrucciones en lenguaje máquina para traducir cada elemento del lenguaje, sin que haga falta un soporte intenso en tiempo de ejecución [22].

2.3.3. Lenguajes JavaScript y TypeScript

JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz

de usuario y páginas web dinámicas, aunque también se utiliza del lado del servidor. Todos los navegadores modernos interpretan el código en este lenguaje integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del modelo de objeto de documento, o DOM (en inglés *Document Object Model*). Es el único lenguaje de programación que entienden de forma nativa los navegadores [23].

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, su antecesor, que esencialmente añade tipos estáticos y objetos basados en clases. Es usado para desarrollar aplicaciones JavaScript que se ejecutarán en el lado del cliente o del servidor, o extensiones para programas. Extiende la sintaxis de su antecesor, por tanto cualquier código en el lenguaje original existente debería funcionar sin problemas. Está pensado para grandes proyectos, los cuales a través de un compilador de TypeScript se traducen a código JavaScript [24].

2.3.4. Angular y Ionic

Angular es un marco de desarrollo (framework en inglés) de ingeniería de software de código abierto mantenido por Google, que sirve para desarrollar aplicaciones web de estilo aplicación de una sola página (en inglés *single page application* o SPA) y aplicación web progresiva (en inglés *Progressive Web App* o PWA). Sirve tanto para versiones móviles como de escritorio. Ofrece soluciones robustas, escalables y optimizadas para lograr un estilo de codificación homogéneo y de gran modularidad. Su desarrollo se realiza por medio de TypeScript o JavaScript. En este último se ofrecen diversas herramientas adicionales al lenguaje como tipado estático o decoradores [25].

El marco de desarrollo Ionic es un kit de desarrollo de software (en inglés *Software Development Kit* o SDK) de frontend de código abierto para desarrollar aplicaciones híbridas basado en tecnologías web (HTML, CSS y JS). Es decir, un framework que nos permite desarrollar aplicaciones multiplataforma desde una única base de código. Posee la capacidad de integrarse con otros marcos populares como Angular, React y Vue. Su principal característica es que permite desarrollar y desplegar aplicaciones híbridas, que funcionan en múltiples plataformas, como iOS nativo, Android, escritorio y la web (como una aplicación web progresiva) con una única base de código [26].

2.3.5. Bases de datos relacionales MySQL y MariaDB

Una base de datos relacional es un conjunto de una o más tablas estructuradas en registros (filas) y campos (columnas), que se vinculan entre sí por un campo en común. MySQL es un sistema de administración de bases de datos relacionales de código abierto. Se considera como la base de datos de código abierto más utilizada en el mundo. Posee cuatro funciones básicas que se conocen con la sigla CRUD: *create* (crear), *read* (leer), *update* (actualizar) y *delete* (borrar). Estas funciones son las que se aplican a los nuevos registros que se quieran crear y a los ya existentes que se deseen leer, actualizar y borrar. Este tipo de bases de datos posee una arquitectura cliente-servidor, siendo el cliente el que hace las solicitudes de datos y el servidor el que posee dichos datos y responde a dicha solicitud.

MariaDB es una versión modificada de MySQL. Fue creada por el equipo de desarrollo original de MySQL debido a problemas de licencia y distribución después

de que Oracle Corporation adquiriera MySQL. MariaDB adopta los archivos de definición de tablas y datos de MySQL y también usa protocolos de cliente, API de cliente, puertos y sockets idénticos. Con ello se pretende que los usuarios de MySQL puedan cambiar a MariaDB sin problemas [27].

2.3.6. Contenedores con Docker

Los contenedores son una forma de virtualización del sistema operativo. Un solo contenedor se puede usar para ejecutar cualquier aplicación, desde un microservicio o un proceso de software a una aplicación de mayor tamaño. Dentro de un contenedor se encuentran todos los ejecutables, el código binario, las bibliotecas y los archivos de configuración necesarios. Sin embargo, en comparación con los métodos de virtualización de máquinas o servidores, los contenedores no contienen imágenes del sistema operativo. Esto los hace más ligeros y portátiles, con una sobrecarga significativamente menor. En implementaciones de aplicaciones de mayor tamaño, se pueden poner en marcha varios contenedores como uno o varios clústeres de contenedores. Estos clústeres se pueden gestionar mediante un orquestador de contenedores, como Kubernetes o Docker Compose [28].

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Usar Docker para crear y gestionar contenedores puede simplificar la creación de sistemas altamente distribuidos, permitiendo que múltiples aplicaciones, las tareas de los trabajadores y otros procesos funcionen de forma autónoma en una única máquina física o en varias máquinas virtuales [29]. En la figura 2.7 se puede ver de forma gráfica el funcionamiento de un contenedor Docker.

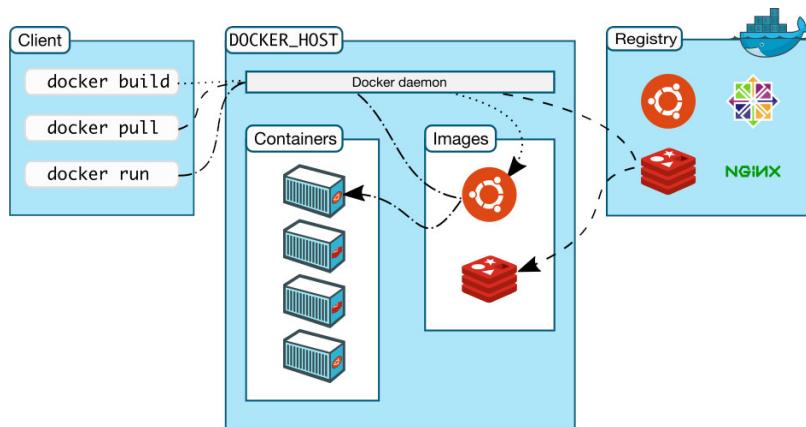


FIGURA 2.7. Contenedor Docker.⁵

Docker Compose es una herramienta para definir y ejecutar aplicaciones Docker de varios contenedores. Se utiliza un archivo YAML para configurar los servicios de su aplicación, y luego, con un solo comando, se crean e inician todos los servicios desde su configuración. Tiene comandos para gestionar todo el ciclo de vida de una aplicación con los que se pueden [30]:

- Iniciar, detener y reconstruir servicios.

⁵Imagen tomada de: <https://algodaily.com/lessons/what-is-a-container-a-docker-tutorial>

- Ver el estado de los servicios en ejecución.
- Transmitir la salida del registro de los servicios en ejecución.
- Ejecutar un comando único en un servicio.

Capítulo 3

Diseño e implementación

En el presente capítulo se presentan los detalles del diseño y los criterios adoptados para el desarrollo del trabajo junto con los pasos seguidos para su implementación.

3.1. Arquitectura del sistema

El sistema tiene una configuración de arquitectura del tipo cliente-servidor. Está constituido por dos nodos y un servidor, los cuales están conectados a la red local y se comunican a través del protocolo MQTT. El servidor recibe los parámetros actuales de estado y cambios desde cada dispositivo, los procesa y almacena en la base de datos. También envía mensajes hacia los dispositivos para cambiar el estado de las salidas o el parámetro que se desee cambiar. Los usuarios pueden consultar y modificar el estado de los dispositivos desde un navegador web móvil o desde una computadora.

En la figura 3.1 se puede observar la arquitectura cliente-servidor del sistema implementado en el trabajo.

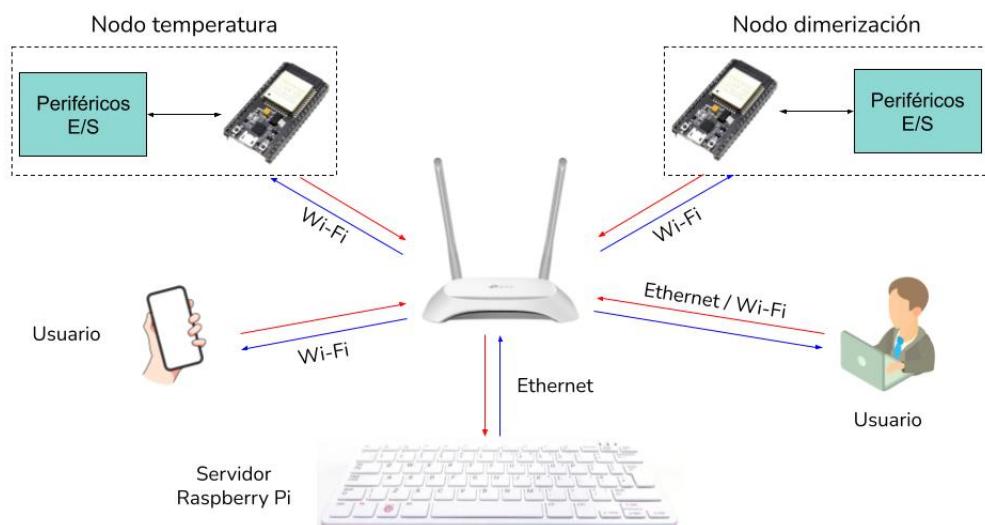


FIGURA 3.1. Arquitectura cliente-servidor.¹

¹Imagen tomada de: <https://www.bujarra.com/raspberry-pi-servidor-vpn-con-pptp/>

Uno de los nodos tiene como función sensar y controlar la temperatura de un recinto, y el otro controlar la iluminación. Originalmente, el proyecto estaba pensado para que un solo nodo implemente estas dos funciones, pero durante el desarrollo se optó por la implementación separada. Este cambio se basó en la idea de modularizar los nodos y que sus funciones sean específicas. De esta forma, es más amigable para el usuario visualizar y modificar el estado en pantalla e implementar el alta de nuevos dispositivos en el sistema.

El servidor está montado sobre una Raspberry Pi 400 con un sistema operativo Raspbian con interfaz gráfica. Este sistema operativo es la versión oficial ofrecida por la fundación Raspberry Pi y está basado en Debian versión 11 (*bullseye*). En esta etapa de desarrollo se optó por una Raspberry Pi 400 por razones de costos y practicidad en el desarrollo y pruebas, aunque tiene un mayor volumen que los otros modelos de la familia. Cuando se ofrezca una solución definitiva, está previsto que se implemente en una placa con un formato más pequeño, como cualquiera de las Raspberry Pi 4. La conexión del servidor a la red local se realiza a través de cable Ethernet.

3.2. Modelo de datos

En esta sección, se describen las diferentes tablas dentro de la base de datos de tipo relacional MariaDB llamada *Domotica*. Con el objetivo de mostrar una representación visual fácil de comprender, se muestran las imágenes representadas en la página phpMyadmin. Las tablas que componen esta base de datos son las siguientes:

- Dispositivos.
- Usuarios.
- Mediciones.

Debe tenerse en cuenta que todo el contenido que se ejecuta del lado del servidor se encuentra dentro de un contenedor Docker. Este contenido corresponde a las imágenes de los servicios de Ionic, MariaDB, phpMyAdmin, backend con Node y Mosquitto. En la figura 3.2 se observa el contenido del archivo *docker-compose.yml*, que corresponde a la configuración de los servicios de MariaDB y phpMyAdmin.

```

mariadb:
  image: tobi312/rpi-mariadb:10.6-alpine
  hostname: mariadb
  environment:
    MARIADB_ROOT_PASSWORD: userpass
    MARIADB_DATABASE: Domotica
    MARIADB_USER: mysql
    MARIADB_PASSWORD: userpass2
  container_name: mariadb
  restart: unless-stopped
  volumes:
    - ./db/dumps:/docker-entrypoint-initdb.d
    - ./db/data:/var/lib/mysql
  networks:
    - app-fullstack-net
  ports:
    - "3306:3306"

phpmyadmin:
  image: phpmyadmin
  environment:
    PMA_HOST: mariadb
    PMA_PORT: 3306
    MARIADB_ROOT_PASSWORD: userpass
  container_name: phpmyadmin
  networks:
    - app-fullstack-net
  depends_on:
    - mariadb
  ports:
    - "8001:80"

```

FIGURA 3.2. Configuración de Docker de la base de datos.

3.2.1. Tabla Dispositivos

Esta tabla contiene los datos de los dispositivos dados de alta en el sistema. Estos datos incluyen el ID del dispositivo, su nombre, la ubicación, la dirección MAC, el tipo, el valor de la alarma y el estado de la alarma (0 para desactivada y 1 para activada).

En la figura 3.3 se muestra la tabla cargada con 2 dispositivos en funcionamiento.

dispositivoID	nombre	ubicacion	mac	tipo	alarma	act_al
0028192332001	ESP32+DHT22	Habitación	94:B5:55:2B:FF:64	Temperatura	26	0
0128192332001	ESP32	Sala	B0:A7:32:DD:18:0C	Luz dimmer	0	0

FIGURA 3.3. Tabla de dispositivos.

El ID del dispositivo es un número proporcionado por el fabricante del dispositivo. Este ID consta de 13 dígitos y sigue un patrón específico: los primeros 2 dígitos representan el tipo de dispositivo (el sistema está diseñado para cubrir hasta un total de 100 tipos de dispositivos, aunque en esta etapa de prototipo sólo se han desarrollado 2); 4 dígitos de seguridad fijos, que en este caso son "2819", utilizados para corroboración y seguridad. Finalmente los últimos 5 dígitos corresponden al número de serie del equipo, con 2 para el año, 2 para la semana y 3 para el número de fabricación del equipo en esa semana, en ese orden. Como puede observarse, el sistema está pensado para que en un futuro sea parte de una producción en serie.

Tanto el nombre como la ubicación son campos alfanuméricos elegidos por el usuario para describir al dispositivo. La dirección MAC es enviada por el dispositivo la primera vez que se conecta al sistema y no debe completarla por el

usuario. El tipo corresponde al tipo de sensor y para esta etapa del desarrollo puede ser "Temperatura" o "Luz dimmer".

El valor de la alarma es un campo numérico y solo tiene efecto para dispositivos del tipo de temperatura. Se enviará una notificación por mail cada vez que el valor enviado por el dispositivo sea mayor o igual al configurado en este campo, siempre que esta alarma se encuentre activada en el campo.

3.2.2. Tabla Usuarios

La tabla de usuarios contiene todos los datos relacionados con las personas utilizarán el sistema. En esta etapa inicial está implementado con 3 usuarios, dado que, al ser de uso doméstico, es poco probable que más de 3 personas se registren en una misma casa. No obstante, el sistema es fácilmente escalable para admitir la cantidad de usuarios que se desee mediante la creación de una página de registro de usuarios.

Los valores almacenados en esta tabla son el ID de usuario (de 1 a 3), el nombre de usuario, la contraseña, el nombre completo de la persona, su e-mail y un campo de tipo booleano que se modificará una vez que el usuario haya actualizado su perfil. Es importante destacar que cada vez que un usuario que no ha actualizado sus datos ingrese al sistema, se mostrará un aviso en pantalla para que este actualice sus datos.

En la figura 3.4 puede verse la tabla cargada con los 3 usuarios, de los cuales 2 han actualizado su perfil y el tercero aún tiene los valores por defecto.

userId	user	password	nombre	apellido	email	updated
1	javier	ceiot	Javier	Fanelli	javifanelli@gmail.com	1
2	romina	user	Romina	Carrizo	rominamarcarrizo@gmail.com	1
3	user3	user	Nombre	Apellido	user3@example.com	0

FIGURA 3.4. Tabla de usuarios.

3.2.3. Tabla Mediciones

La tabla de mediciones contiene todos los datos referentes a las mediciones realizadas por los dispositivos, incluyendo la información sobre el modo de funcionamiento seleccionado. Los dispositivos reportan cada 5 minutos estos datos, y se almacenan en esta tabla.

Los datos almacenados en esta tabla son: el ID de la medición (un valor auto-incremental), el ID del dispositivo, el tipo de dispositivo, la fecha y hora de la medición, el valor de la medición, el modo de funcionamiento (manual o automático), el valor de la salida (si es de tipo temperatura es 0 o 100 y corresponde a encendido o apagado, y si es de tipo luz dimmer va de 0 a 100 con saltos de 10), y la hora y minutos de encendido y apagado para el modo automático.

En la figura 3.5 pueden verse algunas de las mediciones dentro de la correspondiente tabla, ordenadas por ID. Cabe aclarar que sólo se muestran algunas ya que los dispositivos siguen reportando datos y llenando la base de datos con información actual.

medicionId	dispositivoId	tipo	fecha	valor	set_point	modo	salida	hon	mon	hoff	moff
1	0128192332001	Luz dimmer	2023-10-01 21:32:50	0	50	Manual	0	20	0	8	0
2	0028192332001	Temperatura	2023-10-01 21:32:56	21	20	Manual	0	20	0	8	0
3	0128192332001	Luz dimmer	2023-10-01 21:37:53	0	50	Manual	0	20	0	8	0
4	0028192332001	Temperatura	2023-10-01 21:38:02	21	20	Manual	0	20	0	8	0
5	0128192332001	Luz dimmer	2023-10-01 21:42:57	0	50	Manual	0	20	0	8	0
6	0028192332001	Temperatura	2023-10-01 21:43:07	21	20	Manual	0	20	0	8	0
7	0128192332001	Luz dimmer	2023-10-01 21:48:00	0	50	Manual	0	20	0	8	0
8	0028192332001	Temperatura	2023-10-01 21:48:13	21	20	Manual	0	20	0	8	0
9	0128192332001	Luz dimmer	2023-10-01 21:53:03	0	50	Manual	0	20	0	8	0
10	0028192332001	Temperatura	2023-10-01 21:53:18	21	20	Manual	0	20	0	8	0
11	0128192332001	Luz dimmer	2023-10-01 21:58:06	0	50	Manual	0	20	0	8	0
12	0028192332001	Temperatura	2023-10-01 21:58:24	21	20	Manual	0	20	0	8	0
13	0128192332001	Luz dimmer	2023-10-01 22:03:09	0	50	Manual	0	20	0	8	0
14	0028192332001	Temperatura	2023-10-01 22:03:30	21	20	Manual	0	20	0	8	0
15	0128192332001	Luz dimmer	2023-10-01 22:08:13	0	50	Manual	0	20	0	8	0
16	0028192332001	Temperatura	2023-10-01 22:08:35	21	20	Manual	0	20	0	8	0
17	0128192332001	Luz dimmer	2023-10-01 22:13:16	0	50	Manual	0	20	0	8	0
18	0028192332001	Temperatura	2023-10-01 22:13:41	21	20	Manual	0	20	0	8	0
19	0128192332001	Luz dimmer	2023-10-01 22:18:19	0	50	Manual	0	20	0	8	0
20	0028192332001	Temperatura	2023-10-01 22:18:46	21	20	Manual	0	20	0	8	0

FIGURA 3.5. Tabla de mediciones.

3.3. Desarrollo del frontend

El frontend del presente trabajo fue desarrollado en el lenguaje TypeScript con Angular como *framework*, integrado con Ionic. El prototipo de la aplicación está diseñado para acceder desde un navegador web tanto desde una computadora como un móvil, pero se optó por usar Ionic para un posterior desarrollo de una aplicación para sistemas operativos móviles. Por esta razón, en esta instancia puede referirse a la aplicación web como una SPA y no como una PWA.

En la figura 3.6 se puede observar un fragmento de código correspondiente a la configuración de Ionic dentro del archivo *docker-compose.yml*.

```
ionic-ui:
  build:
    context: ./src/frontend/dam
    dockerfile: Dockerfile
  ports:
    - "8100:8100"
  container_name: ionic-ui
  volumes:
    - ./src/frontend/dam:/src/frontend/dam
    - ./src/frontend/dam/node_modules
  command: ionic serve --external
```

FIGURA 3.6. Configuración de Docker de Ionic.

La estructura de archivos de cada página está diseñada como se muestra en la imagen 3.7. A modo de ejemplo se muestran los archivos que componen la página *home*.

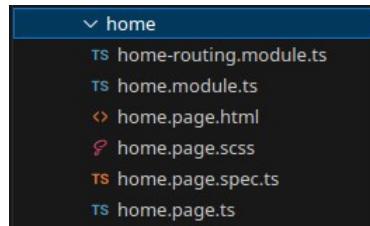


FIGURA 3.7. Estructura de archivos de página.

Además, se utilizaron interfaces para definir las estructuras de datos relacionados con los dispositivos, las mediciones y los usuarios. También se han implementando servicios para el proceso de autenticación y para realizar consultas HTTP al backend, que serán explicadas posteriormente.

El servicio de autenticación implica el ingreso de un ingreso de un usuario y una contraseña en la página de *login*. Estos datos se comparan con los que están almacenados en la base de datos. Si los valores ingresados coinciden con alguno de los registros existentes, el backend genera un *token* que se almacena en el dispositivo que está haciendo la consulta. Esto permite que se puedan acceder a las demás páginas de la aplicación.

En el código 3.1 puede verse el fragmento de la autenticación del archivo *auth.guard.ts*.

```

1 export class AuthGuard {
2   constructor(private _loginService: LoginService, private
3     _router: Router) {}
4   canActivate(
5     route: ActivatedRouteSnapshot,
6     state: RouterStateSnapshot): Observable<boolean | UrlTree> |
7     Promise<boolean | UrlTree> | boolean | UrlTree {
8     if (!this._loginService.logIn) {
9       this._router.navigate(['/login'])
10      return false
11    }
12    return true;
13  }
14 }
```

CÓDIGO 3.1. Autenticación de rutas.

El *route guard* es una característica del Angular Router que permite ejecutar una lógica cuando se solicita una ruta. Basándose en esta lógica, decide si se permite o deniega el acceso al usuario a esa ruta en particular. Comúnmente, es utilizado para verificar si un usuario está logueado o no en el sistema y si tiene la autorización para acceder a una URL determinada.

El *route guard* se puede agregar implementando la interfaz *CanActivate* disponible en `@angular/router` y allí implementar el método *canActivate()* que contendrá la lógica para denegar o permitir el acceso a la ruta [31].

En todas las páginas de la aplicación se utilizan los métodos *ngOnInit* y *ngOnDestroy*. Ambos son métodos de Angular en el ciclo de vida de un componente. Sus funciones son la inicialización de variables y configuración de un componente y la liberación de recursos antes que el componente se destruya [32].

3.3.1. Rutas y páginas destacadas

Las rutas disponibles en la aplicación se describen en la tabla 3.1. Estas rutas permiten la navegación dentro de la aplicación y el acceso a las diferentes pantallas. Se encuentran definidas en el archivo `app-routing.module.ts`, donde también se hace referencia al módulo de la aplicación que debe abrirse al acceder a cada una de las rutas listadas.

TABLA 3.1. Rutas de la aplicación.

Ruta	Descripción
/login	Pantalla de inicio de sesión.
/home	Pantalla principal con funciones y listado de dispositivos.
/usuario/:userId	Pantalla de edición de datos de usuario.
/ayuda	Pantalla de manual de usuario.
/dispositivos/:id	Pantalla de visualización de estado del dispositivo.
/medicion/:id	Pantalla de visualización de las mediciones del dispositivo.
/grafico/:id	Pantalla de visualización del gráfico de las mediciones.
/config/:id	Pantalla de configuración del dispositivo.
/modificar/:id	Pantalla de edición de datos de un dispositivo existente.
/agregar	Pantalla para agregar un dispositivo nuevo.

En los casos en los que se encuentra el campo `:id`, este valor se completa con el valor de identificador del elemento en cuestión. En el caso de los usuarios, cada uno de ellos tiene un identificador, lo mismo que para los dispositivos. Para las rutas `medicion`, `grafico`, `config` y `modificar`, el `:id` al que se hace referencia es el que corresponde al identificador del dispositivo que se quiere leer o modificar.

Pantalla de *login*

En la figura 3.8 se muestra la página de inicio de sesión en la aplicación.

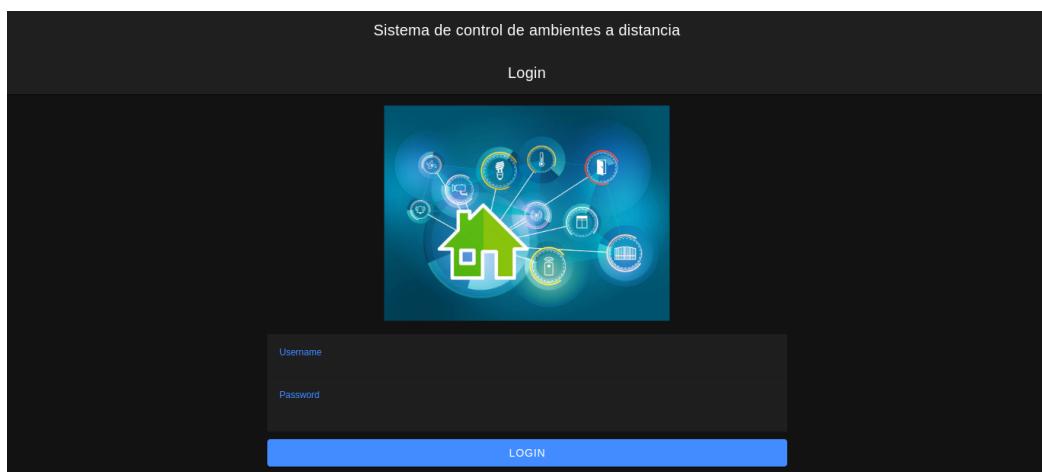


FIGURA 3.8. Pantalla de *login*.

El sistema permite de configurar 3 usuarios distintos para que utilicen, visualicen y configuren la aplicación. Todos poseen el mismo rol y pueden configurar cualquier campo que corresponda a dicho usuario ingresando en la opción Usuario dentro de la pantalla principal.

Pantalla principal *home*

En la figura 3.9 puede verse la pantalla principal de la aplicación. En la parte superior, se encuentran los botones principales de la aplicación tales como el de configuración del usuario actual, cerrar sesión y la página de ayuda o manual de usuario. En la parte inferior, se encuentra el listado de dispositivos implementado con *ion-cards* los cuales permiten ver el estado de cada dispositivo, modificar los datos y configurar alarmas. Además, es posible eliminar un dispositivo de la lista. Es importante tener en cuenta que al borrar un dispositivo del sistema, se borran también todas las mediciones y documentos asociados a él dentro de la base de datos.



FIGURA 3.9. Pantalla de *home*.

Por último, en la parte inferior de la pantalla principal se encuentra el botón para agregar un dispositivo nuevo. Este botón dirige a la página de creación de un dispositivo nuevo.

Pantalla de creación de dispositivo nuevo

En la pantalla que se muestra en la figura 3.10, se deben cargar los datos de un dispositivo nuevo. Esto es importante, ya que aquí se dan de alta los dispositivos nuevos que se incorporan al sistema.

Agregar dispositivo

Ubicación	Sala	X
Nombre	Sensor temperatura	X
ID fabricante	0028192334001	X
Temperatura	25	
Activada	Activada	

[✓ CONFIRMAR](#) [INICIO](#)

FIGURA 3.10. Pantalla de creación de dispositivo nuevo.

Aquí deben colocarse los datos de ubicación del nodo, el nombre con el que se desea identificarlo, el ID de fabricante, el tipo, y los valores de la alarma en caso de que se trate de un nodo de temperatura. El estado de la alarma puede configurarse por primera vez en esta sección, pero puede modificarse cuando se desee. Se debe tener en cuenta que actualmente existen 2 tipos de dispositivos: temperatura y luz dimmer. La funcionalidad de la alarma solo está habilitada para los nodos del tipo de temperatura, ya que no tiene sentido configurar una alarma para un nodo que no realiza mediciones ni controla parámetros críticos.

El ID del fabricante es un número de 13 dígitos con el siguiente formato: los 2 primeros identifican el tipo de dispositivo, los siguientes 4 son obligatoriamente los números "2819" y se utilizan como número de identificación del sistema y como medida de seguridad. Los 2 dígitos subsiguientes representan el año de fabricación del dispositivo; los 2 siguientes son la semana de fabricación; y los últimos 3 dígitos que restan son el número de fabricación en esa semana, es decir que comienza en el "000" y finaliza en "999". Este número resultante es proporcionado por el fabricante y será único para cada dispositivo.

Desde el frontend, se verifica que se ingrese el código de seguridad dentro del ID del fabricante en el fragmento de código 3.2:

```
1 verificarID(dispositivoId: string): boolean {
2     const posicion = 2;
3     return dispositivoId.charAt(posicion) === '2' &&
4         dispositivoId.charAt(posicion + 1) === '8' &&
5         dispositivoId.charAt(posicion + 2) === '1' &&
6         dispositivoId.charAt(posicion + 3) === '9';
7 }
```

CÓDIGO 3.2. Verificación de ID.

Una vez que el dispositivo nuevo esté registrado, al encenderlo se conectará automáticamente a la red y registrará su dirección MAC, que quedará guardada en la tabla de dispositivos.

Pantalla de configuración de modo

En esta sección se va a configurar el modo de trabajo y todos los parámetros asociados. En la figura 3.11 puede observarse la pantalla típica de configuración de un dispositivo de temperatura.

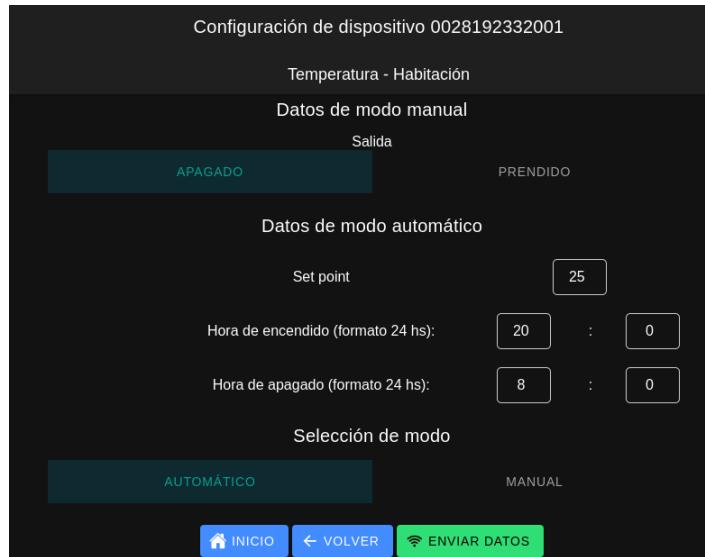


FIGURA 3.11. Pantalla de configuración de modo.

En la parte superior, se puede elegir el estado de la salida en modo manual si fuese un dispositivo de temperatura y el valor de la salida de 0 a 100 si fuese un dimmer. Luego sigue la configuración del modo automático con el set point y las horas de encendido y apagado. Por último, en la parte inferior se puede elegir el modo, automático o manual.

Al cargarse la página de configuración se leen los datos de la última medición del dispositivo y se autocompletan las variables de configuración con estos valores por defecto. Una vez que se hayan seleccionado todos los valores, se debe hacer clic en enviar para que se envíen los datos al dispositivo.

Pantallas de mediciones y gráfico

En la figura 3.12 puede verse el gráfico de medición actual de temperatura del correspondiente nodo.

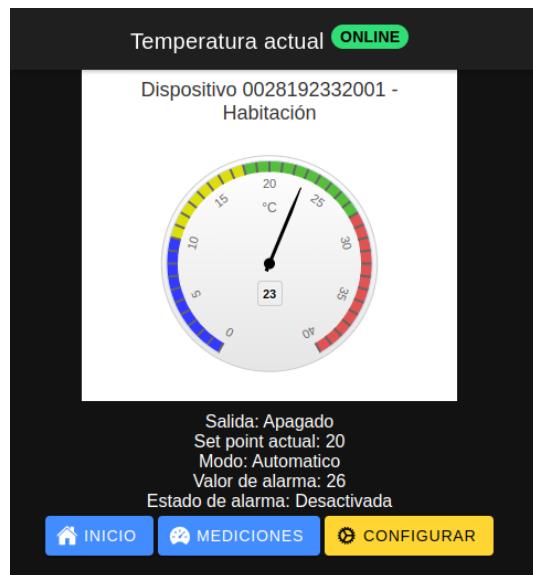


FIGURA 3.12. Pantalla de medición gráfica y estado.

Se utiliza un gráfico de la biblioteca online *highcharts* y se importa desde el archivo *dispositivo.page.ts*, como se muestra en el fragmento de código 3.3. Esta biblioteca posee varios modelos para desarrollo web y móvil para frameworks tales como Javascript, Angular, React, VueJS, iOS, R, .NET, Python y más [33].

```
1 import * as Highcharts from 'highcharts';
```

CÓDIGO 3.3. Importación de gráfico de *highcharts*.

En la figura 3.13 pueden verse las mediciones del dispositivo de temperatura ordenadas desde la más nueva a la más antigua.

Fecha	Hora	Temperatura	Set point	Modo	Salida
Oct 19, 2023	00:58:35	23	20	Automatico	Apagado
Oct 19, 2023	00:53:28	23	20	Automatico	Apagado
Oct 19, 2023	00:48:20	23	20	Automatico	Apagado
Oct 19, 2023	00:43:13	23	20	Automatico	Apagado
Oct 19, 2023	00:38:06	23	20	Automatico	Apagado
Oct 19, 2023	00:32:58	23	20	Automatico	Apagado
Oct 19, 2023	00:27:51	23	20	Automatico	Apagado
Oct 19, 2023	00:22:43	23	20	Automatico	Apagado
Oct 19, 2023	00:17:36	23	20	Automatico	Apagado
Oct 19, 2023	00:12:29	23	20	Automatico	Apagado
Oct 19, 2023	00:07:21	23	20	Automatico	Apagado
Oct 18, 2023	23:57:07	23	20	Automatico	Apagado
Oct 18, 2023	23:52:00	23	20	Automatico	Apagado
Oct 18, 2023	23:46:53	23	20	Automatico	Apagado

FIGURA 3.13. Pantalla de mediciones del dispositivo de temperatura.

En caso que se quieran borrar, se puede hacer con el botón correspondiente como se muestra en la parte superior de la misma figura. Esta opción se agregó en caso de que el usuario no desee ver las mediciones antiguas.

Además, se puede acceder a una página que muestre una representación gráfica de las mediciones del día en curso. Esto hace que sea más fácil leer las mediciones y poder sacar conclusiones o datos que se necesiten. En la figura 3.14 puede verse un ejemplo de gráfico de mediciones reales tomadas por el sensor de temperatura y almacenadas en la base de datos. Se utilizó un gráfico de *highcharts*, cuya función es hacer gráficos temporales basados en mediciones.

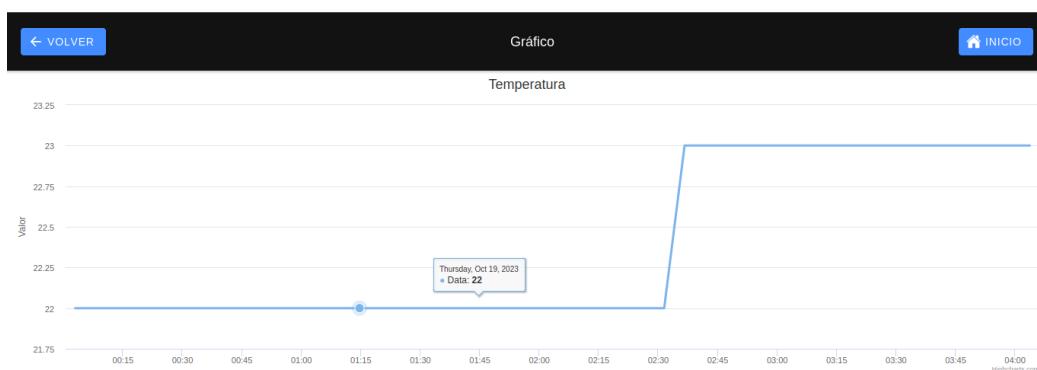


FIGURA 3.14. Pantalla de presentación gráfica de mediciones.

3.4. Desarrollo del backend

El backend fue desarrollado en Node.js mediante JavaScript y Express. Cuenta con un archivo principal *index.js* desde el cual se invocan los distintos archivos que tengan funciones, *endpoints* e inicializaciones de otros módulos y que tengan relación con el backend. Con lo anterior se realizó una primera versión de modularización del código del backend.

En el código 3.4 puede verse el fragmento de importación de bibliotecas y módulos utilizados por el backend y sus respectivas configuraciones:

```

1 const express = require('express');
2 const app = express();
3 const pool = require('./mysql-connector');
4 const cors = require('cors');
5 const jwt = require('jsonwebtoken');
6 const mqtt = require('mqtt');
7 const fs = require('fs');
8 const mqttClient = require('./mqtt-handler');
9 const transporter = require('./nodemailer');
10 const { authRouter } = require('./auth');
11 const { JWT_Secret } = require('./auth');
12 const { dispositivosRouter, ultMedicionRouter, graficoRouter,
    medicionesRouter, deleteDispositivoRouter,
    estadoConexionRouter, borrarTablaRouter, usuariosRouter,
    agregaRouter, modificarDispositivoRouter } =
        require('./dispositivos');
```

CÓDIGO 3.4. Configuración de *index.js*.

En este fragmento de código, se puede ver que se inicializa Express, la base de datos, *cors* para poder conectarse a la API, *jsonwebtoken* para la generación del token, *mqtt* y su configuración, el gestor de mail *nodemailer* para la generación de mails para las alarmas y el archivo *dispositivos.js* con las funciones que se reciben en cada endpoint.

3.4.1. Conexión a la base de datos

La conexión con la base de datos se realiza en *mysql-connector.js* y se muestra en el código 3.5.

```

1 const configmariadb = {
2     connectionLimit: 20,
3     host: '192.168.0.70',
4     port: '3306',
5     user: 'root',
6     password: 'userpass',
7     database: 'Domotica'
8 };
9 const pool = mariadb.createPool(configmariadb);
10 console.log("Iniciando DB");
11 (async () => {
12     try {
13         const connection = await pool.getConnection();
14         console.log ("Conexion exitosa a", configmariadb.database,
15                     "a", configmariadb.host, ":", configmariadb.port);
```

```

15     connection.release();
16 } catch (err) {
17     console.log ("Error al establecer la conexión:", err);
18     process.exit(1);
19 }
20 })();
21
22 module.exports = pool;

```

CÓDIGO 3.5. Conexión con la base de datos.

En la primera sección del código, se configura la conexión estableciendo la URL y puerto de conexión, así como también los datos para poder acceder y el nombre de la base de datos. Luego se crea un pool de conexiones usando `mariadb.createPool` con la configuración definida en `configmariadb`. Un *pool* es una colección de conexiones reutilizables a la base de datos que se gestionan automáticamente para optimizar el rendimiento. Por último, se utiliza una función asíncrona anónima para intentar establecer una conexión a la base de datos y emite mensajes por consola dependiendo si se conecta con éxito o tiene un error. Finalmente, el objeto *pool* se exporta, lo que permite que otros módulos de la aplicación accedan al *pool* de conexiones para interactuar con la base de datos.

3.4.2. Configuración de la conexión *mqtt*

En el documento `mqtt-handler.js` se encuentra la configuración del broker *mosquitto* y de los *topics* utilizados de *mqtt*. En el código 3.6 se muestra el fragmento de inicialización del broker.

```

1 const caCert = fs.readFileSync('/home/node/app/certs/ca.pem');
2 const privateKey =
    fs.readFileSync('/home/node/app/certs/client.key');
3 const clientCert =
    fs.readFileSync('/home/node/app/certs/client.pem');
4 console.log('Certificados SSL para MQTT leidos correctamente');
5
6 const mqttBrokerUrl = '192.168.0.70';
7 const mqttOptions = {
8     host: mqttBrokerUrl,
9     port: 8883,
10    protocol: 'mqtts',
11    ca: caCert,
12    key: privateKey,
13    cert: clientCert,
14 };

```

CÓDIGO 3.6. Inicialización del broker MQTT.

Como se puede observar en el fragmento de código, se leen los certificados TLS generados con anterioridad para añadir seguridad a la comunicación dentro del sistema utilizando OpenSSL. Este es un software desarrollado por el Proyecto OpenSSL, y está compuesto por un conjunto de herramientas robusto, de calidad comercial y con todas las funciones para criptografía de uso general y comunicación segura [34].

3.4.3. Configuración del envío de mails

Para el envío de mails al dispararse una alarma, se utilizó Nodemailer, que es un módulo de *Node.js* para permitir el envío de mails. Las principales características son: es un único módulo sin dependencias, gran enfoque en la seguridad, soporta *Unicode* para usar cualquier carácter incluidos emojis y soporte de contenido HTML y texto sin formato, entre otras [35].

La configuración de el módulo de envío de mensajes se configura como se describe en el código 3.7. Además de este archivo, se debió configurar la cuenta de mail para que pueda aceptar conexiones desde otras aplicaciones en la configuración de seguridad de Google.

```

1 const nodemailer = require('nodemailer');
2
3 const transporter = nodemailer.createTransport({
4   service: 'gmail',
5   user: "smtp.gmail.com",
6   port: 465,
7   secure: true,
8   auth: {
9     type: 'login',
10    user: 'automaticoh@gmail.com',
11    pass: '***** ***** ***** *****'
12  }
13 });
14
15 module.exports = transporter;

```

CÓDIGO 3.7. Configuración del módulo de envío de mails.

3.4.4. API y rutas

Una interfaz de programación de aplicación o API (en inglés, *Application Programming Interface*) es un conjunto de reglas y protocolos que permite que diferentes aplicaciones se comuniquen entre sí. En el contexto de la programación web, las APIs son comunes para permitir la interacción entre aplicaciones cliente y servicios web. En este caso está definida en el archivo *dispositivos.js* y contiene las rutas, funciones y métodos HTTP recibidos en cada consulta desde el frontend, permitiendo interactuar con los otros componentes del sistema.

En el desarrollo del trabajo se utilizan los métodos get, put, post y delete en las distintas rutas. Por ejemplo, en el código 3.8 se puede ver el fragmento de backend que recibe la consulta de listado de dispositivos y hace la propia consulta a la base de datos.

```

1 dispositivosRouter.get('/:id', async function (req, res, next) {
2   try {
3     const connection = await pool.getConnection();
4     const result = await connection.query('SELECT * FROM
5       Dispositivos WHERE dispositivoId = ?', req.params.id);
6     connection.release();
7     res.send(JSON.stringify(result)).status(200);
8   } catch (err) {
9     res.send(err).status(400);
}

```

10 });

CÓDIGO 3.8. Consulta de listado de dispositivos al backend.

Por último, en este archivo se exportan todas las rutas definidas para poder utilizarlas en cualquier módulo que se lo requiera.

3.5. Nodos, sensores y actuadores

En esta sección se describen las características más destacadas de los componentes de hardware del sistema. El software embebido de los microcontroladores se desarrolló con el *framework* ESP-IDF [36] en Visual Studio Code.

Para este trabajo se utilizaron fragmentos de código desarrollados en diversas materias del posgrado así como también bibliotecas para este *framework* disponibles en GitHub. Una de estas bibliotecas corresponde al manejo del display con interfaz SSD1306 [37]. Otro repositorio utilizado fue el que compartió un usuario con el código para el manejo del sensor de temperatura DHT22 y el encoder rotativo con pulsador [38].

Dentro de la carpeta *main* del software de los nodos, se encuentran además los certificados SSL necesarios para que el dispositivo pueda comunicarse con el broker, como se muestra en la figura 3.15.

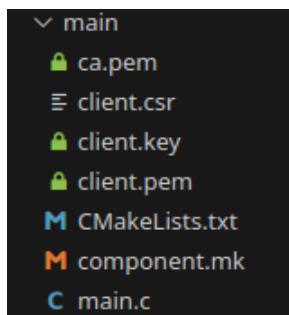


FIGURA 3.15. Ubicación de los certificados SSL.

3.5.1. Características de los nodos

Los nodos están compuestos por el módulo central y los periféricos de entrada y salida, que son los sensores y los actuadores. En el desarrollo del trabajo se utilizó una versión de módulo central con un microcontrolador ESP32 incluido en una placa de desarrollo. En la sección 2.2.2 se describieron las especificaciones técnicas de los microcontroladores utilizados en el trabajo.

En la figura 3.16 puede observarse el diagrama en bloques del nodo de temperatura y control de calefacción. Los periféricos de entrada incluyen el encoder y el sensor de temperatura, mientras que los periféricos de salida comprenden el display que muestra la información y la salida de potencia.

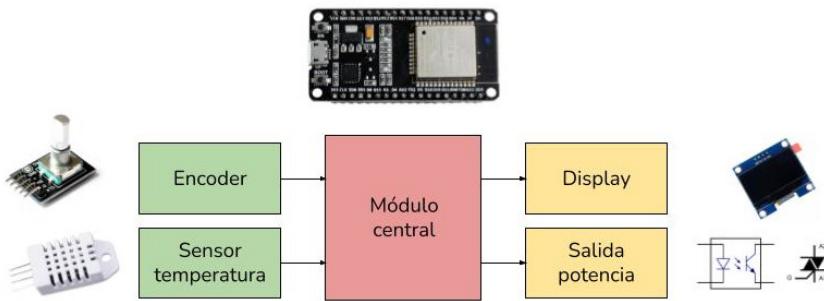


FIGURA 3.16. Diagrama en bloques del nodo de temperatura.

En la figura 3.17 puede verse el diagrama en bloques del nodo de dimerización de la luminaria de corriente continua. En este caso, a diferencia del modelo descrito anteriormente, posee un solo periférico de entrada y el periférico de salida controla una tensión de corriente continua que puede variar entre 5 y 24 V. En el caso del trabajo se utilizó una luminaria de 5 V con un consumo de 200 mA.

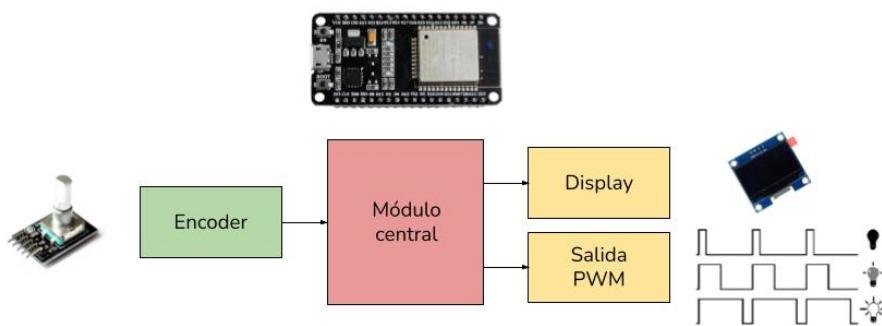


FIGURA 3.17. Diagrama en bloques del nodo de dimerización.

En la figura 3.18 se muestra la placa de desarrollo con el ESP32 montada sobre una placa adicional para facilitar las conexiones y la alimentación de los periféricos.

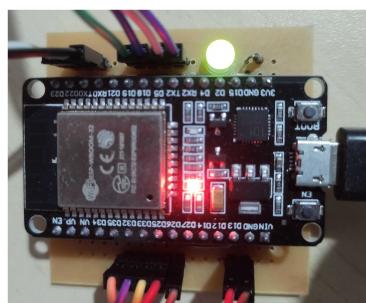


FIGURA 3.18. Placa del módulo principal.

Se puede observar que se encuentra conectada con cables a los demás módulos y posee un LED rojo de encendido. Además cuenta con un LED de 2 colores agregado para mostrar el estado de conexión a la red. Si el nodo está conectado se enciende de color verde, y si está desconectado se enciende de color rojo. También se incorpora un tercer LED de color azul que parpadea cuando el dispositivo envía o recibe datos por MQTT.

En el nodo de temperatura, cuando está seleccionado el modo manual y se gira el encoder, la salida se enciende o se apaga en función del sentido de giro, ya sea horario o antihorario. Por otro lado en el nodo dimmer, al girar en sentido horario se aumenta un 10% la intensidad , mientras que al girar en sentido antihorario se disminuye un 10 %. En ambos dispositivos, si el modo elegido es automático, al girar el encoder no se modifica la salida.

3.5.2. Menús y pantallas

Cada nodo tiene la posibilidad de ser comandado desde el lugar y poder mostrar el estado y configuraciones por el display asociado. Esto es posible gracias a menús implementados en el software que posibilitan la muestra de estos datos. En la figura 3.19 pueden verse las imágenes de las pantallas más importantes del dispositivo.

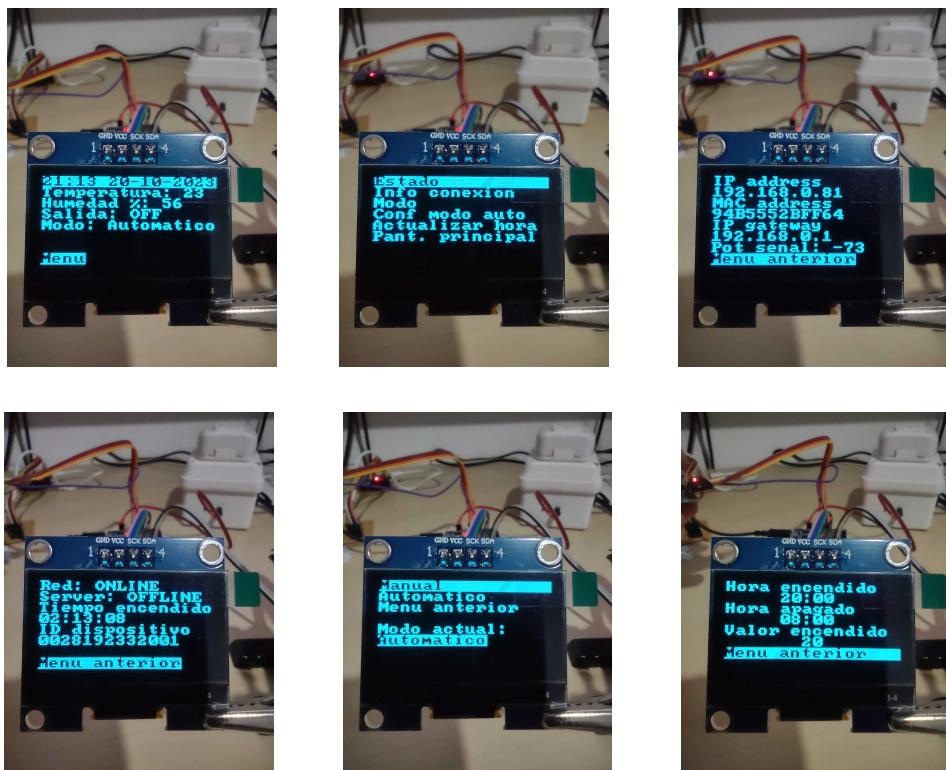


FIGURA 3.19. Pantallas principales del nodo.

A continuación, se listan en orden de presentación:

- Pantalla principal: presenta las mediciones y estado actual.
- Menú principal: muestra el menú principal de opciones.
- Datos de conexión: muestra datos de conexión a la red.
- Datos de red: muestra el estado del servidor y dispositivo.
- Selección de modo: permite elegir el modo de funcionamiento.
- Datos de configuración del modo automático: muestra las horas de encendido y apagado junto con el set point, pero no permite modificar los datos.

Para acceder al menú principal o seleccionar una opción dentro de un menú, es necesario presionar el pulsador incorporado en el encoder. Para desplazarse por las diferentes opciones, debe girar el encoder en sentido horario para desplazarse hacia abajo y en sentido antihorario para desplazarse hacia arriba.

Los menús se implementaron mostrando en pantalla distintos textos y leyendo los valores de las variables auxiliares de posición. En el apéndice A puede verse el fragmento de código que muestra en pantalla las opciones y lee el valor de la variable de posición dentro del mismo.

3.5.3. Modelo de programación

Los nodos fueron programados de forma modularizada, teniendo un código principal en el archivo *main.c*, reducido con funciones definidas. En el código 3.9 puede verse el fragmento de código del módulo principal del software embebido del dispositivo de temperatura.

```

1 void app_main(void)
2 {
3
4     ESP_ERROR_CHECK(nvs_flash_init());
5     ESP_ERROR_CHECK(esp_netif_init());
6
7     _queue = xQueueCreate(QUEUE_LENGTH,
8                           sizeof(rotary_encoder_event_t));
9
10    config_dis();
11    pant_bien();
12    config_led();
13    pant_inicio();
14    wifi_init_sta();
15    if(net_con)
16        mqtt_app_start();
17
18    ESP_ERROR_CHECK(rotary_encoder_init(_queue));
19    ESP_ERROR_CHECK(rotary_encoder_add(&control));
20
21    btn_enc=false;
22    ssd1306_clear_screen(&devd, false);
23    xTaskCreate(get_temp, "get_temp", 4096*8, NULL, 3, NULL);
24    xTaskCreate(read_enc, "read_enc", 4096*2, NULL, 4, NULL);
25    power_on_device();
}

```

CÓDIGO 3.9. Código de *main.c*.

En esta sección, se inicializa el microcontrolador, el encoder y el display. Luego se conecta a la red y si lo consigue, inicializa el módulo de comunicación MQTT. Por último, se crean las tareas más importantes del sistema, la de leer la temperatura y comparar los valores del encoder para leer el sentido de giro y pulsación del botón.

En el nodo de temperatura, la lectura se realiza cada 5 segundos y se incrementa un contador en una unidad. Cada vez que es leída se refresca el valor en pantalla. Si dicho contador llega a un valor de 60 (lo que equivale a 5 minutos) se envían los

valores al servidor. Caso contrario, sigue el desarrollo del programa. En el código 3.10 se puede ver el fragmento de ejecución de tareas al leerse dicho valor.

```

1 void get_temp(void *pvParameter)
2 {
3     while(1) {
4         set_times();
5         if (dht_read_data(sensor_type, dht_gpio, &humidity,
6             &temperature) == ESP_OK) {
7             ESP_LOGI(TAG, "Humidity: %d% Temperature: %dC\n",
8                 humidity/10, temperature/10);
9             if (!time_sinc_ok)
10                obtain_time();
11             time_t now = time(NULL);
12             now-=3*3600;
13             timeinfo = localtime(&now);
14             strftime(pant_time, sizeof(pant_time), "%H:%M
15                 %d-%m-%Y", timeinfo);
16             now = time(NULL);
17             timeinfo = localtime(&now);
18             strftime(formatted_time, sizeof(formatted_time),
19                 "%Y-%m-%d %H:%M:%S", &timeinfo);
20             sprintf(hum_char, "%d", humidity/10);
21             sprintf(temp_char, "%d", temperature/10);
22             if(level==0)
23                 pant_main();
24             esp_wifi_sta_get_ap_info(&ap_info);
25             net_con = (ap_info.authmode != WIFI_AUTH_OPEN);
26             if(cont_mqtt==60)
27                 {
28                     cont_mqtt=0;
29                     if (net_con==false)
30                         esp_wifi_connect();
31                     if(mqtt_state)
32                         mqtt_send_info();
33                 }
34             cont_mqtt++;
35             if(modo==1){
36                 if(time_func &&
37                     ((temperature/10)<=(set_point-hist))){
38                     out_temp=true;
39                     gpio_set_level(CONTROL, 1);
40                 }
41                 if(time_func &&
42                     ((temperature/10)>=(set_point+hist))){
43                     out_temp=false;
44                     gpio_set_level(CONTROL, 0);
45                 }
46             }
47             vTaskDelay(pdMS_TO_TICKS(1000*refresh));
48         } else {
49             if (cont_temp > 5){
50                 ESP_LOGE(TAG, "Could not read data from sensor\n");
51             }
52         }
53     }
54 }
```

```

49         pant_no_sensor();
50     }
51     cont_temp++;
52     xTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(100));
53 }
54 }
55 vTaskDelete(NULL);
56 }
```

CÓDIGO 3.10. Código de lectura de temperatura.

En el nodo de dimmer, el funcionamiento es el mismo y el código muy similar, con la diferencia que se envía el valor de la salida ya que no posee sensores de entrada que estén leyendo parámetros.

3.6. Comunicación del sistema

El sistema utiliza el protocolo MQTT con certificados SSL [39]. Estos certificados son de tipo autofirmado y se generaron en el servidor, uno para el broker y otro para los nodos, como se detalla en la página de EMQX [40]. El sistema admite la comunicación bidireccional, es decir, permite el envío de mensajes tanto desde los nodos hacia el servidor como desde el servidor hacia los nodos, según el tipo y función del mensaje.

3.6.1. Topics MQTT utilizados

El sistema posee distintos *topics* para el envío de información. A continuación, se detalla cada uno de ellos:

- */home/temperatura/data*: es el que utilizan los nodos de temperatura para enviar las mediciones al servidor.
- */home/temperatura/settings*: es el que utiliza el servidor para enviar los datos a los nodos de temperatura, incluyendo los parámetros de programación seleccionados por el usuario a través de la aplicación web.
- */home/dimmer/data*: es el que utilizan los nodos de dimerización para enviar las mediciones al servidor.
- */home/dimmer/settings*: es el que utiliza el servidor para enviar los datos a los nodos de dimerización, incluyendo los parámetros de programación seleccionados por el usuario a través de la aplicación web.
- */home/config*: por aquí se envían los mensajes de los nodos al servidor cuando se encienden, dando un mensaje de que el nodo está activo.
- */home/setup*: aquí el servidor les envía los datos de configuración a los dispositivos cuando se encienden, de modo que tengan acceso a los datos de configuración correspondientes a la última medición realizada y se preserve el estado más reciente.

En todos los casos, se utilizó un QoS de cero, dado que la pérdida de un mensaje, en caso de que ocurriera, no representaría un problema significativo.

Capítulo 4

Ensayos y resultados

En este capítulo se detallan los ensayos realizados al sistema completo y los resultados obtenidos durante las pruebas de funcionamiento.

4.1. Banco de pruebas

El sistema consta físicamente de dos tipos de componentes: nodos y servidor. El servidor incluye tanto la aplicación en el frontend y el backend con sus respectivos módulos. Además, el sistema posee dos tipos de nodos: uno para el control de calefacción y otro para el control de iluminación.

Para ensayar el nodo de temperatura, se utilizaron los materiales que lo conforman que se muestran en la figura 4.1.

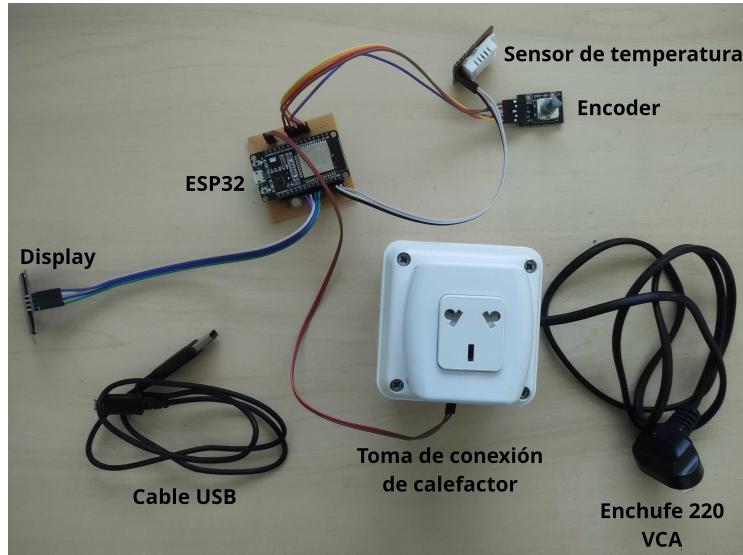


FIGURA 4.1. Componentes del nodo de temperatura.

En la figura 4.1, se puede observar la placa ESP32 montada sobre la placa de conexiones, el display, el encoder, el sensor de temperatura y una caja estanca con una conexión de toma corriente para conectar la estufa a encender. Además, esta caja incluye un cable para conectarse a una fuente de alimentación de 220 VCA, que actúa como un interruptor. Cabe aclarar que en modo automático, el control de temperatura opera con una histéresis de 1 grado. Esto significa que al configurar una temperatura específica, el control apagará la salida cuando la

temperatura supere en 1 grado al set-point y la encenderá cuando esté 1 grado por debajo de este valor.

Para ensayar el nodo de dimerización, se utilizaron los materiales que lo conforman que se muestran en la figura 4.2.

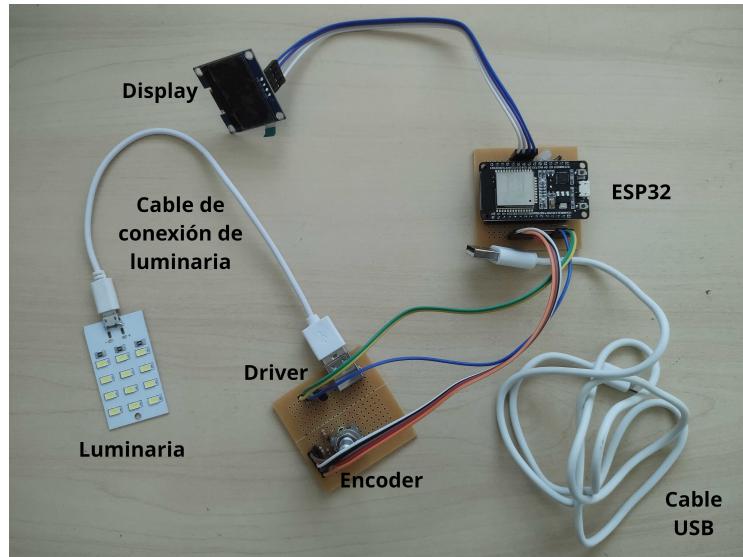


FIGURA 4.2. Componentes del nodo de dimerización.

En la figura 4.2 se puede observar la placa ESP32 montada sobre la placa de conexiones, el display, el encoder, el *driver* de iluminación y la placa de LEDs de 5 VCC. El control posee un conector USB por lo que puede conectarse cualquier lámpara que funcione en este caso con 5 VCC y posea este conector.

En la figura 4.3 pueden verse los componentes que forman parte del servidor.



FIGURA 4.3. Componentes del servidor.

Puede observarse la Raspberry Pi400, la fuente de 5 VCC 3A con su cable, un disco rígido externo SSD de 120 GB y un cable Ethernet para la conexión a la red.

4.2. Metodología empleada

El hardware se testeó de forma funcional. En cuanto al software se utilizaron distintas metodologías para la depuración de forma manual de los embebidos, el frontend y el backend. A continuación, se describen cada una de ellas.

4.2.1. Pruebas del frontend

Para el frontend se utilizaron pruebas funcionales manuales con el sistema funcionando. El desarrollo fue progresivo y se llevó a cabo en paralelo. Se creó una aplicación básica y, posteriormente, se fueron agregando nuevas funcionalidades. A medida que el sistema fue creciendo, se fueron agregando más componentes hasta tener la versión actual.

En lo que respecta a las pruebas, se empleó la salida de la consola del navegador a medida que se implementaban nuevas funciones y páginas. Esto permitió exhibir en tiempo real el funcionamiento del sistema en cada página y con cada función agregada. En el código 4.1 se muestra a modo de ejemplo parte del desarrollo de la página para la visualización de un mensaje por consola de la configuración de un dispositivo.

```
1  ngOnInit() {  
2      const deviceId =  
3          this.activatedRoute.snapshot.paramMap.get('id') as string;  
4      this.dispositivoId = deviceId, 10;  
5      this.subscription =  
6          this.dispositivoService.getDeviceById(this.dispositivoId) .  
7              subscribe((data) => {  
8                  console.log(data);  
9                  this.device = data[0];  
10                 this.tipo = data[0].tipo;  
11                 this.ubicacion = data[0].ubicacion;  
12             });  
13     this.leerdatos();  
14 }
```

CÓDIGO 4.1. Muestra por consola de los datos recibidos.

Allí se puede ver la función *console.log* mostrando los datos por consola. En la figura 4.4 se puede ver la pantalla de configuración del dispositivo y el mensaje por consola con los datos recibidos.

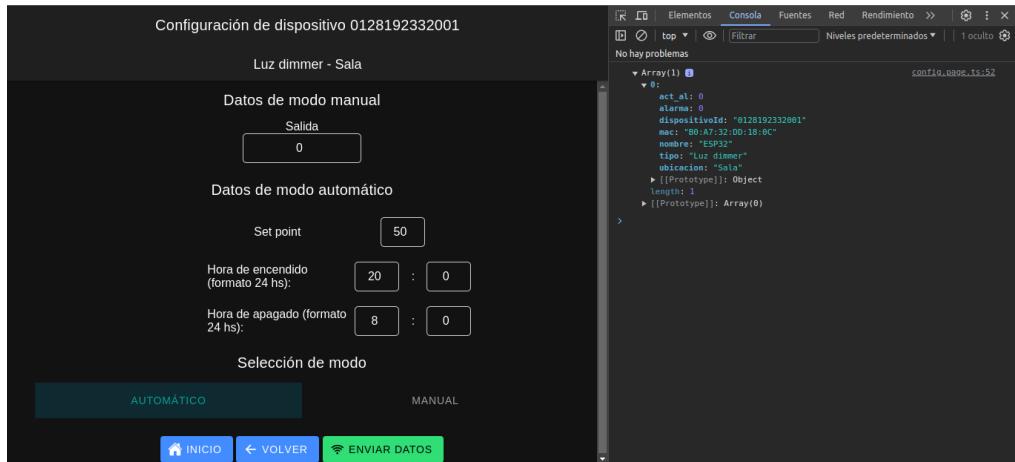


FIGURA 4.4. Pantalla de aplicación y consola.

Este mecanismo se utilizó en todas las páginas de la aplicación web y gracias a esto se consiguieron los resultados esperados de funcionalidad y depuración de errores.

4.2.2. Pruebas del backend

Para el backend se utilizó un mecanismo similar de pruebas funcionales que para el frontend.

El desarrollo siguió un enfoque progresivo lo que significa que a medida que se fueron incorporando funciones o endpoints nuevos, se fueron colocando muestras por consola para poder observar si los datos y las consultas estaban siendo resueltos de forma correcta. En el código 4.2 se muestra como ejemplo el *router* para borrar la tabla de mediciones de un dispositivo y el mensaje por consola de éxito y el ID del dispositivo.

```

1  borrarTablaRouter.delete('/:id', async function (req, res,
2      next) {
3      const id = req.params.id;
4      let connection;
5      try {
6          connection = await pool.getConnection();
7          await connection.beginTransaction();
8          const deleteMedicionesQuery = 'DELETE FROM Mediciones WHERE
9              dispositivoId = ?';
10         await connection.query(deleteMedicionesQuery, id);
11         await connection.commit();
12         connection.release();
13         res.send({ message: 'Mediciones eliminadas exitosamente'
14             }).status(200);
15         console.log('Solicitud de eliminacion recibida para
16             dispositivoId:', id);
17     } catch (err) {
18         if (connection)
19             await connection.rollback();
20             connection.release();
21         }
22         res.send(err).status(400);
23         console.log('Error al eliminar mediciones:', err);
24     }

```

21 });

CÓDIGO 4.2. Muestra por consola de los datos consultados.

En la figura 4.5 se puede ver la consola con el mensaje de éxito y el ID del dispositivo cuya tabla de mediciones fue eliminada.

```
ionic-ui | 
ionic-ui | 
node-backend | Solicitud de eliminación recibida para dispositivoId: 0128192332001
```

FIGURA 4.5. Mensaje por consola del servidor.

Además de estas pruebas, se incorporó el uso de la aplicación MQTTX para testear la comunicación con el *broker* y los dispositivos, especialmente durante la implementación de la seguridad con los certificados SSL. De esta forma, se llevaron a cabo pruebas que abarcaron desde la primera conexión segura hasta la inserción de datos desde un dispositivo simulado, el envío de configuración desde la aplicación a los nodos y el mensaje de solicitud de configuración inicial de un nodo, entre otros.

En la figura 4.6 puede verse la configuración de MQTTX para el envío de datos de una medición simulada en el *topic* \home\temperatura\data para luego verificar la correcta inserción en la base de datos.

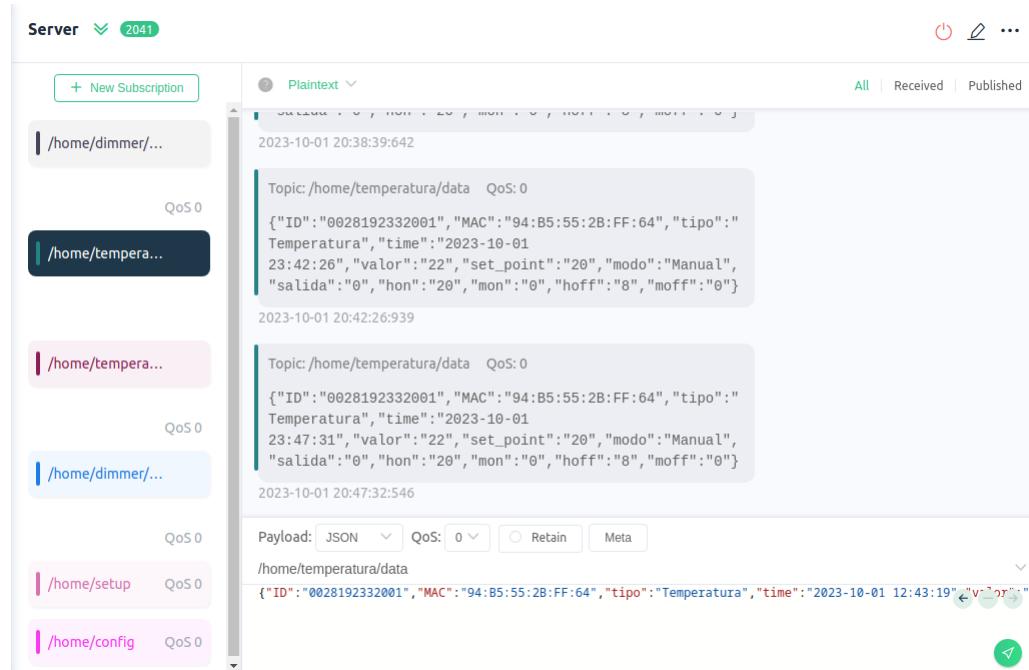


FIGURA 4.6. Pantalla de MQTTX.

4.2.3. Pruebas de los nodos

El hardware se testeó haciendo pruebas de funcionamiento con los materiales mencionados en la sección anterior. Las pruebas consistieron en poner en funcionamiento los nodos durante dos días consecutivos, verificando que el control de temperatura funcionara dentro de los valores configurados y que la luminaria se

encendiera y mantuviera los valores de los saltos. También se evaluó el funcionamiento de los horarios de encendido y apagado.

En cuanto al software, las placas ESP32 poseen comunicación serial incorporada por lo que se colocaron funciones de muestra por consola al momento de incorporar funcionalidades nuevas. En el código 4.3, se puede ver un ejemplo de cómo se utilizó la función *ESP_LOGI* para mostrar por consola el valor de la salida recibido por MQTT.

```

1 const cJSON *salida = cJSON_GetObjectItemCaseSensitive(root,
2     "salida");
3 if (cJSON_IsNumber(salida) && select) {
4     if(salida->valueint==100)
5         out_temp=true;
6     if(salida->valueint==0)
7         out_temp=false;
8     ESP_LOGI(TAG, "Received MQTT salida: %d",
9             salida->valueint);
10    pant_main();
11 }
```

CÓDIGO 4.3. Muestra por consola de la terminal ESP-IDF.

En este caso se muestra por consola el mensaje de recepción del valor de salida y su valor correspondiente.

4.3. Resultados finales

Para la prueba final, se ensamblaron los 2 nodos, y se conectó un calefactor eléctrico de 750 W al nodo de temperatura. En el lado del servidor, se ejecutó el *docker-compose* con el sistema completo. Se realizaron ajustes tanto desde la aplicación web como de cada uno de los nodos.

En la figura 4.7, se puede observar el nodo de iluminación funcionando encendido, y en la pantalla se muestra el valor de la salida.

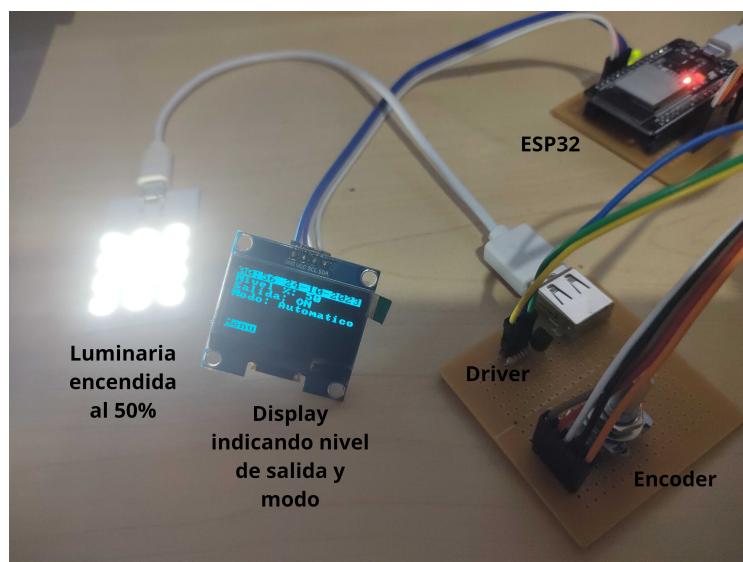


FIGURA 4.7. Dimmer en modo automático funcionando.

En la figura 4.8, se puede observar la pantalla de configuración de modo automático.



FIGURA 4.8. Pantalla de configuración del dimmer en modo automático.

En la figura 4.9 puede observarse la página de configuración del nodo dimmer. Puede observarse que los datos ingresados corresponden con los datos guardados en el nodo, y que este último está funcionando según lo esperado.

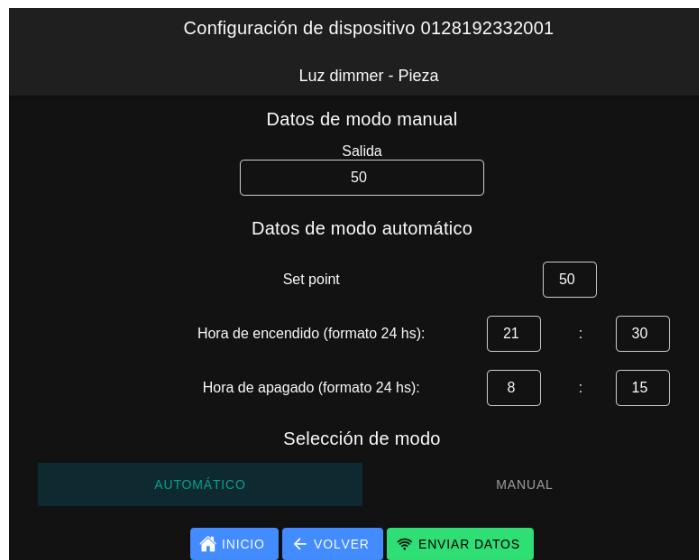
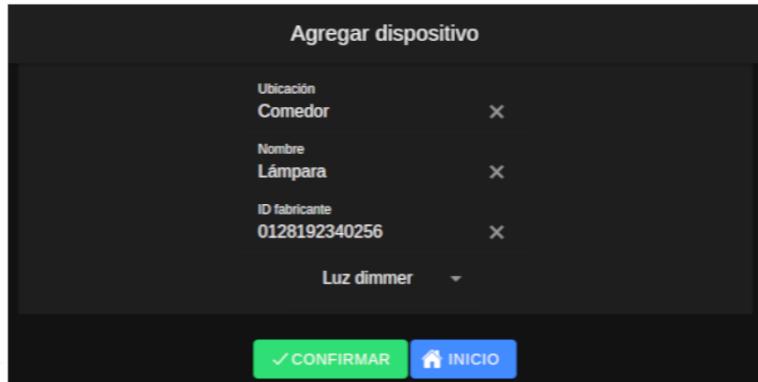


FIGURA 4.9. Pantalla de configuración de dimmer.

Se ha verificado la capacidad de agregar y modificar los datos de los dispositivos desde la página correspondiente, contrastando los datos visualizados en la pantalla de *phpMyAdmin*. Durante una prueba que se realizó, se cargó un dispositivo nuevo y se corroboró tanto desde la aplicación del sistema como desde la página de administración de la base de datos. En la figura 4.10 puede verse cómo se agregó un dispositivo nuevo desde la página en la parte superior y cómo se ve reflejado en la base de datos en la inferior.



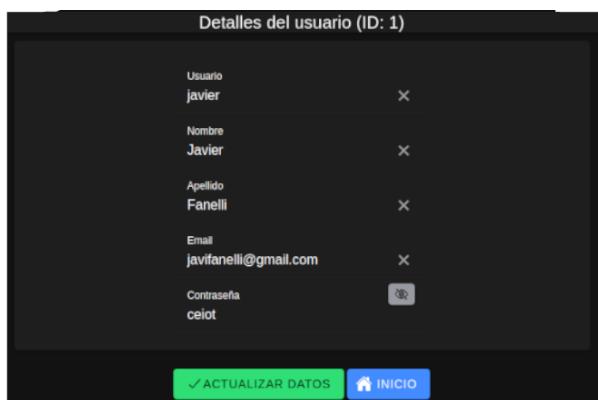
	dispositivoId	nombre	ubicación	mac	tipo	alarma	act_al
<input type="checkbox"/>	0028192332001	ESP32+DHT22	Habitación	94:B5:55:2B:FF:64	Temperatura	26	0
<input type="checkbox"/>	0128192332001	Lámpara lechuza	Pieza	B0:A7:32:DD:18:0C	Luz dimmer	0	0
<input type="checkbox"/>	0128192340256	Lámpara	Comedor		Luz dimmer	25	0

↑ Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar Exportar

FIGURA 4.10. Valores de dispositivo nuevo.

Aquí se observa que los datos cargados coinciden con los almacenados. La diferencia radica en que no se registró la MAC del dispositivo ya que al momento de obtener la imagen el dispositivo no había emitido datos al servidor.

Algo similar a la prueba de los dispositivos se hizo con los usuarios. El sistema por default tiene los valores *user1*, *user2* y *user3* y contraseña *user*. Esto se logró creando estos usuarios con sus datos en el archivo de creación de la base de datos *domotica.sql*. En la figura 4.11 se muestran los valores ingresados en la página de configuración en la parte superior y los que están almacenados en la base de datos en la inferior. Como se puede observar en la imagen, los valores cargados desde la página coinciden con los almacenados en la base de datos.



	userId	user	password	nombre	apellido	email	updated
<input type="checkbox"/>	1	javier	ceiot	Javier	Fanelli	javifanelli@gmail.com	1
<input type="checkbox"/>	2	romina	user	Romina	Carrizo	rominamarcarrizo@gmail.com	1
<input type="checkbox"/>	3	user3	user	Nombre	Apellido	user3@example.com	0

↑ Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar Exportar

FIGURA 4.11. Valores de usuario modificados.

Por último, se hizo la corroboración de la activación de la alarma y el envío del mail. En la figura 4.12 puede observarse la configuración del dispositivo en la parte superior y el mail recibido en la parte inferior.

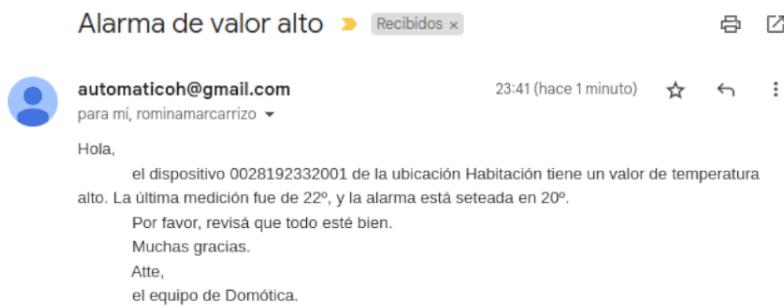
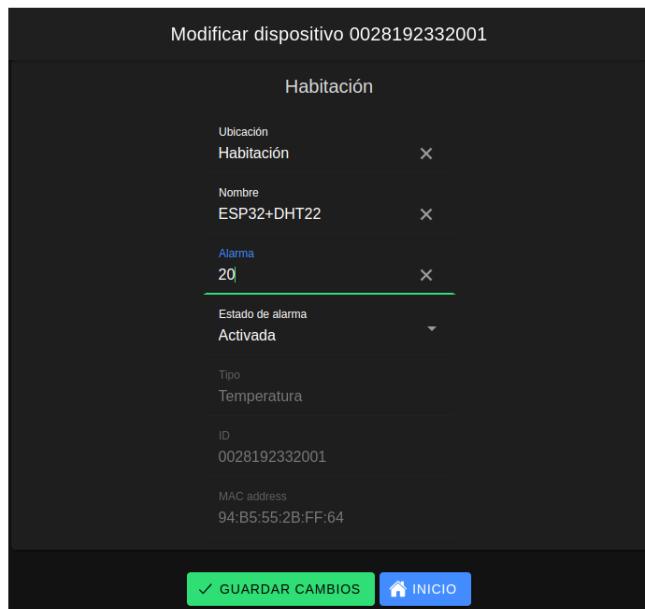


FIGURA 4.12. Envío de alarma de medición.

De este modo, se realizaron las pruebas más importantes de funcionalidades del sistema. Al dar resultados de funcionamiento satisfactorios, se concluye que la aplicación funciona de manera correcta.

4.4. Comparación con el estado del arte

En la tabla 4.1 se encuentra la comparación entre las soluciones de hogares inteligentes existentes en el mercado nacional, Domotic y Reactor, y el trabajo realizado.

TABLA 4.1. Comparativa entre las distintas opciones.

Funcionalidad	Domotic	Reactor	Trabajo final
Posee unidad central	Sí	No	Sí
Capacidad de diseño de dispositivos nuevos	Sí	Sí	Sí
Almacenamiento de mediciones	No	Sí	Sí
Programación de reacciones	Sí	Sí	Sí
Aplicación en ambientes profesionales y oficinas	No	Sí	Sí
Avisos por mail	No	Sí	Sí
Aplicación móvil	Sí	Sí	No
Conexión desde el exterior	Sí	Sí	No

Como se puede observar de la tabla comparativa, el trabajo está a la par de otras soluciones similares. Entre sus puntos positivos, el sistema desarrollado posee una unidad central, en este caso, el servidor, que puede aceptar diseños nuevos de dispositivos, proporciona notificaciones en caso de eventos y almacena valores de mediciones que sean útiles.

En cuanto a los aspectos a mejorar se describirán en el capítulo 5 en la sección de trabajo futuro.

Capítulo 5

Conclusiones

En el presente capítulo se detallan los resultados obtenidos del trabajo realizado y se describen las mejoras en un posible trabajo futuro.

5.1. Resultados obtenidos

En este trabajo se concluyó el desarrollo y pruebas funcionales de un prototipo de sistema de automatización de hogares, contando con 2 tipos de dispositivos.

Para evaluar los resultados finales, es necesario analizar los siguientes temas:

- Se cumplió con las fechas finales de finalización del trabajo, aunque la realización de las tareas en la práctica difirió un poco de lo planificado. Se planificaron las tareas de forma muy secuencial pero se hicieron algunas de ellas en paralelo junto con otras.
- No se manifestó ninguno de los riesgos advertidos en la planificación, aunque se siguieron todas las medidas de mitigación de aquellos riesgos más graves.
- Se logró cumplir con todos los requerimientos establecidos y acordados con el cliente, aunque fue necesario realizar una modificación que implicó dividir el nodo previsto, de modo que ahora dos nodos tengan funciones específicas.

Fueron de gran aporte y utilidad todos los conocimientos adquiridos en el transcurso del posgrado, y en especial aquellos conocimientos adquiridos en las materias que se listan a continuación:

- Protocolos de Internet.
- Arquitecturas de datos.
- Arquitecturas de protocolos.
- Desarrollo de aplicaciones multiplataforma.
- Desarrollo de aplicaciones para Internet de las cosas.

5.2. Trabajo futuro

A continuación, se describen las tareas necesarias para implementar mejoras en el sistema, y su ejecución se contemplará en un posible trabajo futuro:

- Hacer aplicaciones móviles para Android y iOS, ya que el desarrollo fue hecho en Ionic con este propósito.
- Implementar un *access point* en el servidor para que los dispositivos se conecten automáticamente a él una vez que sean encendidos por primera vez.
- Implementar actualizaciones de software OTA en los dispositivos.
- Integración con sistemas de seguridad y cámaras.
- Mejoras en la interfaz gráfica de la web tales como el uso de menús en las páginas en lugar de usar solo botones.
- Integración con Google Assistant.
- Implementación de un servicio en la nube para el acceso fuera del hogar o edificación, manteniendo todas las otras funcionalidades en el servidor local.
- Implementación de niveles de usuarios para que haya un usuario *root* que pueda crear cuantos usuarios desee y los administre.
- Implementación de alarmas y avisos a través de la aplicación móvil o web y a través de plataformas como Telegram.
- Diseño de circuitos impresos y gabinetes para los nodos.

Apéndice A

Ejemplo de creación de menú en ESP32

En este anexo se describe un ejemplo de manejo del menú principal en los nodos.

```

1 void menu1 (void)
2 {
3     ssd1306_clear_screen (&devd, false);
4     pos_menu=1;
5     while (level==1) {
6         if (inc_enc) {
7             pos_menu++;
8             inc_enc=false;
9             if (pos_menu>6)
10                pos_menu=1;
11        }
12        if (dec_enc) {
13            pos_menu--;
14            dec_enc=false;
15            if (pos_menu<1)
16                pos_menu=6;
17        }
18        if (pos_menu==1) {
19            ssd1306_display_text (&devd, 0, "Estado ", 16, true);
20            ssd1306_display_text (&devd, 1, "Info conexion ", 16,
21                                false);
22            ssd1306_display_text (&devd, 2, "Modo ", 16, false);
23            ssd1306_display_text (&devd, 3, "Conf modo auto ", 16,
24                                false);
25            ssd1306_display_text (&devd, 4, "Actualizar hora ", 16,
26                                false);
27            ssd1306_display_text (&devd, 5, "Pant. principal ", 16,
28                                false);
29            if (btn_enc) {
30                btn_enc=false;
31                level=10;
32            }
33        }
34        if (pos_menu==2) {
35            ssd1306_display_text (&devd, 0, "Estado ", 16, false);
36            ssd1306_display_text (&devd, 1, "Info conexion ", 16,
37                                true);
38            ssd1306_display_text (&devd, 2, "Modo ", 16, false);
39            ssd1306_display_text (&devd, 3, "Conf modo auto ", 16,
40                                false);

```

```
35     ssd1306_display_text(&devd, 4, "Actualizar hora ", 16,
36         false);
37     ssd1306_display_text(&devd, 5, "Pant. principal ", 16,
38         false);
39     if(btn_enc) {
40         btn_enc=false;
41         level=11;
42     }
43     if(pos_menu==3) {
44         ssd1306_display_text(&devd, 0, "Estado ", 16, false);
45         ssd1306_display_text(&devd, 1, "Info conexion ", 16,
46             false);
47         ssd1306_display_text(&devd, 2, "Modo ", 16, true);
48         ssd1306_display_text(&devd, 3, "Conf modo auto ", 16,
49             false);
50         ssd1306_display_text(&devd, 4, "Actualizar hora ", 16,
51             false);
52         if(btn_enc) {
53             btn_enc=false;
54             level=2;
55         }
56     }
57     if(pos_menu==4) {
58         ssd1306_display_text(&devd, 0, "Estado ", 16, false);
59         ssd1306_display_text(&devd, 1, "Info conexion ", 16,
60             false);
61         ssd1306_display_text(&devd, 2, "Modo ", 16, false);
62         ssd1306_display_text(&devd, 3, "Conf modo auto ", 16,
63             true);
64         ssd1306_display_text(&devd, 4, "Actualizar hora ", 16,
65             false);
66         ssd1306_display_text(&devd, 5, "Pant. principal ", 16,
67             false);
68         if(btn_enc) {
69             btn_enc=false;
70             level=3;
71         }
72     }
73     if(pos_menu==5) {
74         ssd1306_display_text(&devd, 0, "Estado ", 16, false);
75         ssd1306_display_text(&devd, 1, "Info conexion ", 16,
76             false);
77         ssd1306_display_text(&devd, 2, "Modo ", 16, false);
78         ssd1306_display_text(&devd, 3, "Conf modo auto ", 16,
79             false);
80         ssd1306_display_text(&devd, 4, "Actualizar hora ", 16,
81             true);
82         ssd1306_display_text(&devd, 5, "Pant. principal ", 16,
83             false);
84         if(btn_enc) {
85             btn_enc=false;
86             ssd1306_display_text(&devd, 6, "Obteniendo la", 13,
87                 false);
88             ssd1306_display_text(&devd, 7, "hora...", 7, false);
```

```
77     obtain_time();
78     ssd1306_display_text(&devd, 7, "hora... OK", 10,
79         false);
80     vTaskDelay(pdMS_TO_TICKS(1000));
81     ssd1306_display_text(&devd, 6, "    ", 15, false);
82     ssd1306_display_text(&devd, 7, "    ", 15, false);
83 }
84 if(pos_menu==6){
85     ssd1306_display_text(&devd, 0, "Estado ", 16, false);
86     ssd1306_display_text(&devd, 1, "Info conexion ", 16,
87         false);
88     ssd1306_display_text(&devd, 2, "Modo ", 16, false);
89     ssd1306_display_text(&devd, 3, "Conf modo auto ", 16,
90         false);
91     ssd1306_display_text(&devd, 4, "Actualizar hora ", 16,
92         false);
93     ssd1306_display_text(&devd, 5, "Pant. principal ", 16,
94         true);
95     if(btn_enc){
96         btn_enc=false;
97         level=0;
98     }
99 }
100 if(level==10){
101     if(net_con)
102         pant_conok();
103     if(!net_con)
104         pant_nocon();
105 }
106 if(level==11){
107     pant_est();
108 }
109 if(level==2){
110     menu2();
111 }
112 if(level==3){
113     menu3();
114 }
115 }
```

CÓDIGO A.1. Código del menú principal.

Bibliografía

- [1] *Domótica ¿Qué es la domótica? ¿Cómo funciona?* 2023. URL: <https://e-ficiencia.com/domotica-que-es-y-como-funciona/>.
- [2] *Domótica - Wikipedia.* 2023. URL: <https://es.wikipedia.org/wiki/Domotica>.
- [3] *Matter.* 2023. URL: <https://csa-iot.org/all-solutions/matter/>.
- [4] *Home assistant.* 2023. URL: <https://www.home-assistant.io/>.
- [5] *Domotic.* 2018. URL: <https://www.sistemasdomotic.com.ar/>.
- [6] *Commax.* 2023. URL: <http://domotica.com.ar/>.
- [7] *Reactor.* 2023. URL: <https://www.reactor.com.ar/>.
- [8] *Modelo OSI, Wikipedia.* 2023. URL: https://es.wikipedia.org/wiki/Modelo_OSI.
- [9] *Protocolo HTTP, Wikipedia.* 2023. URL: https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto.
- [10] *Protocolo MQTT, Goto IoT.* 2021. URL: https://www.gotiot.com/pages/articles/mqtt_intro/index.html.
- [11] *Seguridad de la capa de transporte, Wikipedia.* 2023. URL: https://es.wikipedia.org/wiki/Seguridad_de_la_capa_de_transporte.
- [12] *Sistema en un chip, Wikipedia.* 2023. URL: https://es.wikipedia.org/wiki/Sistema_en_un_chip.
- [13] *Familia de módulos y placas ESP.* 2023. URL: <https://docs.espressif.com/projects/esp-idf/en/v4.2.3/esp32/hw-reference/modules-and-boards.html>.
- [14] *Especificaciones del ESP32.* 2023. URL: <https://www.espressif.com/en/support/documents/technical-documents>.
- [15] *Hoja de datos del DHT22.* 2023. URL: <https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf>.
- [16] *Hoja de datos del KY-040.* 2023. URL: <https://www.rcscomponents.kiev.ua/datasheets/ky-040-datasheet.pdf>.
- [17] *Hoja de datos del SSD1306.* 2023. URL: <https://datasheethub.com/ssd1306-128x64-mono-0-96-inch-i2c-oled-display/>.
- [18] *Hoja de datos del BT137.* 2023. URL: <https://pdf1.alldatasheet.com/datasheet-pdf/view/16764/PHILIPS/BT137.html>.
- [19] *Hoja de datos del MOC3041.* 2023. URL: <https://pdf1.alldatasheet.com/datasheet-pdf/view/5039/MOTOROLA/MOC3041.html>.
- [20] *Hoja de datos del BC337.* 2023. URL: <https://pdf1.alldatasheet.com/datasheet-pdf/view/2884/MOTOROLA/BC337.html>.
- [21] *Visual Studio Code.* 2023. URL: <https://code.visualstudio.com/>.
- [22] *Lenguaje de programación C, Wikipedia.* 2023. URL: [https://es.wikipedia.org/wiki/C_\(lenguaje_de_programaciÃ§Ã\)n](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaciÃ§Ã)n).
- [23] *Lenguaje de programación JavaScript, Wikipedia.* 2023. URL: <https://es.wikipedia.org/wiki/JavaScript>.

- [24] *Lenguaje de programación TypeScript*, Wikipedia. 2023. URL: <https://es.wikipedia.org/wiki>TypeScript>.
- [25] *¿Qué es Angular?*, Hubspot. 2022. URL: <https://blog.hubspot.es/website/que-es-angular>.
- [26] *Qué es Ionic: ventajas y desventajas de usarlo para desarrollar apps móviles híbridas*, profile. 2021. URL: <https://profile.es/blog/que-es-ionic/>.
- [27] *¿Cuál es la diferencia entre MariaDB y MySQL?*, AWS. 2023. URL: <https://aws.amazon.com/es/compare/the-difference-between-mariadb-vs-mysql/#:~:text=MySQL%20is%20more%20scalable%20and,multiple%20engines%20in%20one%20table..>
- [28] *¿Qué son los contenedores?*, NetApp. 2021. URL: <https://www.netapp.com/es/devops-solutions/what-are-containers/>.
- [29] *Docker (software)*, Wikipedia. 2023. URL: [https://es.wikipedia.org/wiki/Docker_\(software\)](https://es.wikipedia.org/wiki/Docker_(software)).
- [30] *Descripción general de Docker Compose*, Docker. 2023. URL: <https://docs.docker.com/compose/>.
- [31] *Apunte teórico de la materia Desarrollo de Aplicaciones Multiplataforma*. 2023. URL: <https://github.com/brianducca/dam/blob/master/Material/Material%20DAM.pdf>.
- [32] *Ciclo de vida de un componente de Angular*. 2023. URL: <https://angular.io/guide/lifecycle-hooks>.
- [33] *Documentación de gráficos highcharts*. 2023. URL: <https://www.highcharts.com/>.
- [34] *Proyecto OpenSSL*. 2023. URL: <https://www.openssl.org/>.
- [35] *Documentación de Nodemailer*. 2023. URL: <https://nodemailer.com/>.
- [36] *Framework de desarrollo ESP-IDF*. 2023. URL: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/#>.
- [37] *Librería para manejo de display SSD1306*. 2023. URL: <https://github.com/nopnop2002/esp-idf-ssd1306>.
- [38] *Librerías para DHT22 y encoder rotativo*. 2023. URL: <https://github.com/nopnop2002/esp-idf-ssd1306>.
- [39] *Generación de certificados SSL*. 2023. URL: <https://www.emqx.com/en/blog/enable-two-way-ssl-for-emqx>.
- [40] *Guía de generación de certificados SSL*. 2023. URL: <https://www.emqx.com/en/blog/emqx-server-ssl-tls-secure-connection-configuration-guide>.