



CARRERA DE ESPECIALIZACIÓN EN INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

Sistema de gestión de cultivos aeropónicos

Autor:
Ing. Nahuel Severini

Director:
Mg. Ing. Patricio Bos (FIUBA)

Jurados:
Nombre del jurado 1 (pertenencia)
Nombre del jurado 2 (pertenencia)
Nombre del jurado 3 (pertenencia)

*Este trabajo fue realizado en la ciudad de Castelar,
entre octubre de 2022 y agosto de 2023.*

Resumen

La presente memoria describe el diseño e implementación de un sistema de control y monitoreo para cultivos aeropónicos, cuyo objetivo es aumentar la productividad y reducir la complejidad en el mantenimiento.

Para lograr este proyecto, se emplearon conocimientos especializados en protocolos de comunicación, arquitectura y pruebas de *software*, arquitectura de datos y ciberseguridad.

Índice general

Resumen	I
1. Introducción general	1
1.1. Aeroponía	1
1.2. Objetivos y alcance	2
1.3. Estado del arte	3
1.4. Requerimientos	5
2. Introducción específica	9
2.1. Protocolos de comunicación	9
2.1.1. <i>Hypertext Transfer Protocol</i>	9
2.1.2. <i>Message Queue Telemetry Transport</i>	10
2.1.3. <i>WebSocket</i>	11
2.2. Componentes de <i>hardware</i>	12
2.2.1. Microcontrolador	12
2.2.2. Sensor digital de temperatura y humedad	13
2.2.3. Sensor de intensidad de luz	13
2.2.4. Sensor de calidad del aire	14
2.2.5. Sensor de flujo	15
2.2.6. Sensor digital de temperatura para líquidos	15
2.2.7. Módulo de <i>relay</i>	16
2.3. Herramientas de <i>software</i>	17
2.3.1. <i>Progressive Web Apps</i>	17
2.3.2. <i>Framework del frontend</i>	17
2.3.3. Entorno en tiempo de ejecución del <i>backend</i>	18
2.3.4. Base de datos	19
2.3.5. <i>Framework de pruebas del frontend</i>	20
2.3.6. <i>Framework de pruebas del backend</i>	20
2.3.7. <i>Software de auditorías del frontend</i>	21
3. Diseño e implementación	22
3.1. Arquitectura del sistema	22
3.1.1. Observaciones	23
3.2. Modelo de datos	23
3.2.1. Colección <i>Users</i>	24
3.2.2. Colección <i>Zones</i>	24
3.2.3. Colección <i>Notifications</i>	25
3.2.4. Colección <i>Measurements</i>	26
3.3. Desarrollo del <i>backend</i>	26
3.4. Desarrollo del <i>frontend</i>	30
3.4.1. Principales pantallas de la aplicación	33
3.5. Desarrollo del <i>broker</i>	38

3.6. Desarrollo sobre el microcontrolador	39
4. Ensayos y resultados	42
4.1. Pruebas del <i>frontend</i>	42
4.2. Pruebas del <i>backend</i>	45
4.3. Prueba final de integración	48
4.4. Comparación con el estado del arte	53
5. Conclusiones	55
5.1. Resultados obtenidos	55
5.2. Trabajo futuro	55
Bibliografía	57

Índice de figuras

1.1. Zona de cultivo aeropónica ¹	1
1.2. Diagrama aeroponía ²	2
1.3. Diagrama general de la solución.	8
2.1. Ejemplo de comunicacion en HTTP ³	10
2.2. Ejemplo de comunicación en MQTT ⁴	11
2.3. Ejemplo de comunicación en WebSocket ⁵	12
2.4. Fotografía del sensor DHT22 ⁶	13
2.5. Fotografía del sensor Bh1750 ⁷	14
2.6. Fotografía del sensor MQ135 ⁸	14
2.7. Fotografía del sensor YF-S201B ⁹	15
2.8. Fotografía del sensor DS18B20 sumergible ¹⁰	16
2.9. Fotografía del módulo de <i>relay</i> ¹¹	16
2.10. Ejemplo de modelado de datos en MongoDB ¹²	19
2.11. Ejemplo de auditoría realizada en Lighthouse ¹³	21
3.1. Diagrama en bloques de la solución.	23
3.2. Interface de la colección <i>Users</i>	24
3.3. Interface de la colección <i>Zones</i>	25
3.4. Interface de la colección <i>Notifications</i>	26
3.5. Interface de la colección <i>Measurements</i>	26
3.6. Diagrama de secuencia de la autorización de usuarios.	27
3.7. Diagrama de secuencia de la autenticación de usuarios.	28
3.8. Estructura de directorios.	30
3.9. Uso de certificados TLS.	31
3.10. Estructura de directorios.	33
3.11. Pantalla de registro de usuario.	34
3.12. Pantalla de listado de zonas.	34
3.13. Pantalla de creación de una zona.	35
3.14. Ejemplo de <i>email</i> de notificación de alarma.	35
3.15. Ejemplo de notificacion <i>push</i>	36
3.16. Pantalla de listado de notificaciones.	36
3.17. Listado de mediciones en vista de <i>dashboard</i>	37
3.18. Gráfico de mediciones en vista de <i>dashboard</i>	37
3.19. <i>Cards</i> en vista de <i>dashboard</i>	37
4.1. Pruebas unitarias.	42
4.2. Cobertura de código.	43
4.3. Certificados TLS aplicados.	44
4.4. Auditoría realizada sobre la pantalla de <i>dashboard</i> de una zona	44
4.5. Auditoría realizada sobre la pantalla de listado de zonas	45
4.6. Auditoría realizada sobre la pantalla de listado de notificaciones	45
4.7. Pruebas unitarias parte 1 de 2.	46

4.8. Pruebas unitarias parte 2 de 2	47
4.9. Cobertura de código.	47
4.10. Diagrama del banco de pruebas.	49
4.11. Inicialización del <i>backend</i>	49
4.12. Inicialización del <i>frontend</i>	50
4.13. Pantalla de creación de una zona.	50
4.14. Pantalla de listado de zonas.	51
4.15. Configuración de credenciales en el microcontrolador.	51
4.16. Monitoreo del microcontrolador.	51
4.17. Listado de mediciones en vista de <i>dashboard</i>	52
4.18. <i>Cards</i> en vista de <i>dashboard</i>	52
4.19. Pantalla de listado de notificaciones.	52
4.20. <i>Email</i> enviado por el sistema.	52
4.21. Notificación <i>push</i> enviada por el sistema.	53

Índice de tablas

1.1. Comparativa entre soluciones comerciales similares	4
1.2. Comparativa entre soluciones similares encontradas en publicaciones científicas	5
2.1. Bibliotecas de terceros utilizadas	13
2.2. Comparación entre distintos tipos de aplicaciones	17
2.3. Bibliotecas de terceros utilizadas	18
2.4. Bibliotecas de terceros utilizadas	19
3.1. <i>Endpoints</i> disponibles	29
3.2. Rutas disponibles	32
4.1. Comparativa entre soluciones comerciales similares y el trabajo realizado	53
4.2. Comparativa entre soluciones similares encontradas en publicaciones científicas y el trabajo realizado	53

Capítulo 1

Introducción general

En este capítulo se presenta una introducción al concepto de aeroponía y sus principales ventajas. Además, se describe el estado del arte y se exponen los objetivos, alcance y requerimientos del trabajo.

1.1. Aeroponía

La aeroponía es una técnica de cultivo de plantas en la que las raíces cuelgan suspendidas en el aire mientras se les entrega una solución nutritiva en forma de una fina niebla [1]. Como se usa agua para transmitir nutrientes, a veces se habla de los cultivos aeropónicos como de tipo hidropónico [2].

En la figura 1.1 se presenta un típico ejemplo de una zona de cultivo aeropónica, en la que se puede observar el posicionamiento de las plantas y su sistema de nebulización.



FIGURA 1.1. Zona de cultivo aeropónica¹.

¹Imagen tomada de <https://www.acquagarden.com.ar/aeroponia>

El crecimiento aeropónico es considerado seguro y ecológico por producir cosechas de forma natural manteniendo las plantas saludables. Su principal ventaja ecológica frente a otros sistemas es la conservación de agua, energía y nutrientes [3].

Una de sus características más importante es que requieren poco contacto físico con el sistema radicular del cultivo, para no interferir con el crecimiento y expansión natural de las raíces y lograr un intercambio limpio de agua y aire. Por esta razón, en comparación con las plantas hidropónicas, los cultivos tienden a crecer más rápido y a absorber más nutrientes, ya que están expuestas a más oxígeno. A su vez, hay menos amenazas de enfermedades alrededor de la zona de la raíz ya que no hay lugar para que residan los desechos [4].

En la figura 1.2 puede observarse el diagrama de un cultivo aeropónico junto con su proceso de nebulización.



FIGURA 1.2. Diagrama aeroponía².

Sin embargo, para lograr que un sistema aeropónico tenga éxito es necesario que se controlen múltiples parámetros que son vitales para el correcto desarrollo de los cultivos, como por ejemplo la temperatura y caudal de la solución nutritiva a utilizar en el proceso de nebulización, la temperatura y humedad del ambiente, el nivel de dióxido de carbono y el nivel de iluminación entre otros [5].

1.2. Objetivos y alcance

El propósito de este trabajo es el diseño, desarrollo e implementación de un sistema que permita la gestión de cultivos aeropónicos, con el objetivo de incrementar su productividad y reducir la dificultad de mantenimiento.

El trabajo incluye:

- Diseño y desarrollo de una aplicación SSR (del inglés *server-side rendering*) [6] que permita la interacción del usuario con el sistema.

²Imagen tomada de <https://hidroponia24.com/sistemas-hidroponicos/aeroponía/>

- Diseño y desarrollo de una API (del inglés *application programming interface*) [7] REST (del inglés *representational state transfer*) [8] que permita la comunicación por medio de los protocolos HTTP (del inglés *Hypertext Transfer Protocol*) [9], MQTT (del inglés *Message Queue Telemetry Transport*) [10] y WebSocket [11].
- Diseño de un modelo de datos a utilizar por el DaaS (del inglés *data as a service*) [12] que permita almacenar los datos.
- Configuración y despliegue del DaaS.
- Diseño y desarrollo de un *broker* que permita la comunicación por medio de los protocolos MQTT y WebSockets.
- Desarrollo del *software* del microcontrolador para realizar una prueba de concepto.

El trabajo no incluye:

- Diseño de un gabinete para el microcontrolador y los sensores a utilizar.
- Desarrollo del *software* final del microcontrolador.
- Uso de certificados TLS (del inglés *Transport Layer Security*) [13] emitidos por autoridades de confianza.
- Despliegue del servidor en un entorno *cloud*.
- Accionar la bomba y el nebulizador del cultivo aeropónico por medio del sistema.

1.3. Estado del arte

Se llevó a cabo una búsqueda de sistemas similares en el mercado nacional, pero solo se encontró un producto que ofrece características comparables o que cuente con documentación suficiente para una comparación adecuada. Por este motivo, se decidió incluir en la búsqueda sistemas del mercado internacional y publicaciones científicas.

En la tabla 1.1 se presenta la comparación de las características y funcionalidades de los sistemas comerciales encontrados en el mercado nacional e internacional.

My Autogrow [14] permite la automatización y control de todo tipo de sistemas agrícolas. Los datos del sistema son obtenidos por medio de sensores y se almacenan en una plataforma *cloud*. El sistema utiliza MQTT y HTTP como protocolos de comunicación. Los usuarios acceden al sistema mediante una aplicación web, cuyas principales características se detallan a continuación:

- Histórico de mediciones con filtros.
- Administración de alertas.
- Administración de áreas de cultivo.
- Administración de sistema de fertirrigación [15].
- Administración de sistema de riego.
- Visualización de gráficos.

Smartcultiva [16] permite la obtención de datos y métricas mediante una red de sensores instalados en la zona de cultivo. Los datos son enviados mediante MQTT y se almacenan en una plataforma *cloud*. Los usuarios acceden al sistema mediante una aplicación web o móvil, cuyas principales características se detallan a continuación:

- Histórico de mediciones con filtros.
- Administración de dispositivos externos.
- Exportación de datos a diferentes formatos.
- Administración de alertas.
- Automatización de acciones frente al disparo de alertas.
- Visualización de gráficos.

Es importante notar que My Autogrow y Smartcultiva trabajan con sensores propietarios solamente.

TABLA 1.1. Comparativa entre soluciones comerciales similares.

Funcionalidad	Smartcultiva	My Autogrow
MQTT	Sí	Sí
Sistema de alertas	Sí	Sí
Acciones automatizadas	Sí	No
Notificaciones <i>push</i> [17]	No	No
Notificaciones <i>email</i>	Sí	Sí
Accionamiento de dispositivos externos	Sí	No
Escalabilidad en sensores	Limitado	Limitado
Tipo de aplicación	Móvil y web	Web

En la tabla 1.2 se presenta la comparación de las características y funcionalidades de los sistemas encontrados en publicaciones científicas.

Monitoring Soil and Ambient Parameters in the IoT Precision Agriculture Scenario: An Original Modeling Approach Dedicated to Low-Cost Soil Water Content Sensors [18] presenta el desarrollo de un sistema de monitoreo de parámetros ambientales y del suelo enfocado en la agricultura. Los datos del sistema son obtenidos por nodos a nivel planta y a nivel de invernadero y enviados a un *gateway* mediante el protocolo LoRa (del inglés *long range*) [19]. El *gateway* es el encargado de utilizar MQTT para enviar los datos a una plataforma *cloud*. Los usuarios acceden al sistema mediante una plataforma web que permite monitorear los datos obtenidos y visualizar gráficos.

A Smart Agricultural System Based on PLC and a CloudComputing Web Application Using LoRa and LoRaWan [20] presenta el desarrollo de un sistema compuesto por dos tipos de nodos: a nivel granja y a nivel depósito. Los nodos envían los datos obtenidos a un *gateway* mediante el protocolo LoRa. El *gateway* es el encargado de enviar los datos a una plataforma *cloud*. Los usuarios acceden al sistema mediante una plataforma web que permite monitorear y exportar los datos obtenidos, administrar el sistema de alertas y visualizar gráficos. El sistema envía las notificaciones de las alertas mediante *email* y un *bot* [21] de Telegram [22].

Se considera que la escalabilidad para incorporar nuevos sensores en las dos soluciones anteriores es media, debido a las estructuras planteadas a nivel aplicación.

Es importante destacar que ambas soluciones fueron ejecutadas y probadas, es decir no son publicaciones teóricas.

TABLA 1.2. Comparativa entre soluciones similares encontradas en publicaciones científicas.

Funcionalidad	<i>Monitoring Soil (...)</i>	<i>A Smart (...)</i>
MQTT	Sí	No
LoRa	Sí	Sí
Sistema de alertas	No	Sí
Acciones automatizadas	No	No
Notificaciones <i>push</i>	No	No
Notificaciones <i>email</i>	No	Sí
Accionamiento de dispositivos externos	No	No
Escalabilidad en sensores	Media	Media
Tipo de aplicación	Web	Web

1.4. Requerimientos

Se organizaron los requerimientos en secciones de acuerdo con su relación a las distintas áreas del trabajo.

1. Requerimientos asociados a la aplicación SSR:
 - a) Deberá estar configurada para utilizar certificados TLS.
 - b) Deberá estar configurada para poder enviar y recibir información mediante los protocolos HTTP y WebSockets.
 - c) Deberá incluir el renderizado de la aplicación web en el servidor.
 - d) Deberá contar con una interfaz de usuario para permitir el inicio de sesión en el sistema.
 - e) Deberá contar con una interfaz de usuario para permitir la creación y modificación de usuarios.
 - f) Deberá contar con las interfaces de usuario para realizar el CRUD (del inglés *create, read, update and delete*) [23] de zonas.
 - g) Deberá contar con las interfaces de usuario para realizar el CRUD de alarmas.
 - h) Deberá validar los datos recibidos en cada uno de los formularios de las interfaces de usuario.
 - i) Deberá contar con una interfaz de usuario para monitorear en tiempo real el histórico de las mediciones obtenidas.
 - j) Deberá contar con una interfaz de usuario para permitir la activación de los *relays* disponibles.

- k) Deberá contar con un módulo para controlar el acceso a las interfaces de usuario según su nivel de autenticación.
2. Requerimientos asociados a la API REST:
- a) Deberá estar configurada para utilizar certificados TLS.
 - b) Deberá estar configurada para poder enviar y recibir información mediante los protocolos HTTP, WebSockets y MQTT.
 - c) Deberá incluir los *endpoints* para realizar la alta y la modificación de usuarios.
 - d) Deberá incluir los *endpoints* para realizar el CRUD de zonas.
 - e) Deberá incluir los *endpoints* para realizar el CRUD de alarmas.
 - f) Deberá incluir un *endpoint* para obtener el histórico de mediciones.
 - g) Deberá incluir un *endpoint* para la activación de los *relays* [24] disponibles.
 - h) Deberá incluir un módulo de autenticación y autorización de usuarios con sus respectivos *endpoints*.
 - i) Deberá validar los datos recibidos en cada uno de los *endpoints*.
3. Requerimientos asociados al DaaS:
- a) Deberá ser configurado y desplegado en un entorno *cloud*.
 - b) Deberá persistir la información de los usuarios: id, *email*, contraseña y fecha y hora de creación.
 - c) Deberá persistir la información de las zonas de cultivo: id, nombre, localización, dispositivo, usuario, alarmas asociadas y última medición.
 - d) Deberá persistir la información de las mediciones: id, temperatura del ambiente, humedad del ambiente, temperatura de la solución nutritiva, nivel de dióxido de carbono, nivel de iluminación, fecha con la hora de creación del registro, usuario, zona y dispositivo.
 - e) Deberá persistir la información de los dispositivos: id, nombre, contraseña, estado del módulo de *relay*.
 - f) Deberá persistir la información de las alarmas: id, nombre, condición, zona y usuario.
4. Requerimientos asociados al *broker*:
- a) Deberá estar configurado para utilizar certificados TLS.
 - b) Deberá estar configurado para poder enviar y recibir información mediante los protocolos MQTT y WebSockets.
 - c) Deberá contar con un módulo de autenticación y autorización de dispositivos.
 - d) Deberá permitir la conexión de múltiples microcontroladores de manera simultánea.
 - e) Deberá permitir la activación de los *relays* disponibles.

f) Deberá analizar la información recibida, validarla y en caso de ser necesario almacenarla en el DaaS.

g) Deberá enviar una notificación al usuario en caso de que la condición de una alarma predefinida se cumpla.

5. Requerimientos asociados al microcontrolador y sensores:

a) Deberá estar configurado para utilizar certificados TLS.

b) Deberá utilizar el protocolo MQTT para enviar y recibir información.

c) Deberá tener configurado las credenciales necesarias para unirse a la red local mediante Wi-Fi [25].

d) Deberá permitir accionar el módulo de *relay*.

e) Deberá obtener la temperatura del ambiente con una exactitud de $\pm 0,5$ °C (Celsius) y una resolución de 0,1 °C en un rango de operación entre los 15 °C y 35 °C. La frecuencia de muestreo deberá ser de una muestra cada 5 minutos.

f) Deberá obtener la humedad del ambiente con una exactitud de $\pm 2\%$ RH (humedad relativa) y una resolución de 0,1 % RH en un rango de operación de 40-100 % RH. La frecuencia de muestreo deberá ser de una muestra cada 5 minutos.

g) Deberá obtener la temperatura de la solución nutritiva con una exactitud de $\pm 0,5$ °C en un rango de operación entre los 15 °C y 35 °C. La frecuencia de muestreo deberá ser de una muestra cada 5 minutos.

h) Deberá obtener el nivel de dióxido de carbono del ambiente con una exactitud de $\pm 15\%$ en un rango de operación de 400-1000 PPM (partes por millón). La frecuencia de muestreo deberá ser de una muestra cada 5 minutos.

i) Deberá obtener el nivel de iluminación en el ambiente con un rango de operación de 3000-50000 lx. La frecuencia de muestreo deberá ser de una muestra cada 5 minutos.

j) Deberá obtener el caudal líquido con una exactitud de $\pm 10\%$ en un rango de operación de 1-30 litros por minuto.

6. Requerimientos asociados a las pruebas:

a) La aplicación SSR deberá contar con al menos 70 % de cobertura de código [26].

b) La API REST deberá contar con al menos 70 % de cobertura de código.

En la figura 1.3 se presenta un diagrama general de la solución y los componentes que la forman.

Para cada zona de cultivo aeropónica se emplea un microcontrolador, responsable de recopilar los datos de los sensores y transmitirlos al servidor, el cual se encarga de analizar y validar la información recibida y en caso necesario enviarla al DaaS para ser almacenada.

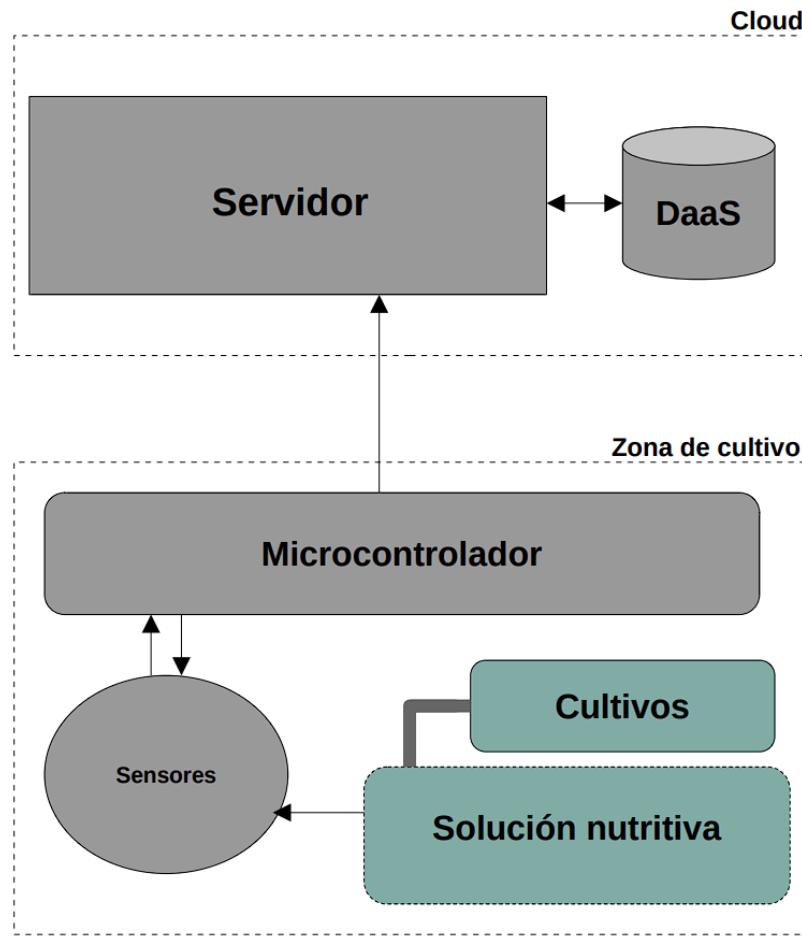


FIGURA 1.3. Diagrama general de la solución.

Capítulo 2

Introducción específica

En este capítulo se introducen los protocolos de comunicación, los diversos componentes de hardware y las herramientas de software utilizadas para el desarrollo del trabajo.

2.1. Protocolos de comunicación

2.1.1. Hypertext Transfer Protocol

HTTP es un protocolo de la capa de aplicación [27] para la transmisión de documentos hipermedia, como HTML (del inglés *HyperText Markup Language*) [28]. Fue diseñado para la comunicación entre los navegadores y servidores web, aunque se puede utilizar para otros propósitos también.

Sigue el clásico modelo cliente-servidor [29], en el que un cliente establece una conexión con el servidor, realiza una petición y espera hasta que recibe una respuesta del mismo.

Se trata de un protocolo sin estados [30], lo cual significa que el servidor no guarda información entre dos peticiones distintas.

Aunque en la mayoría de casos se basa en una conexión del tipo TCP/IP (del inglés *Transmission Control Protocol/Internet Protocol*) [31] y se puede usar sobre cualquier capa de transporte [32] segura o de confianza.

Las características claves de HTTP son:

- Es sencillo: HTTP está pensado y desarrollado para ser leído y fácilmente interpretado por sus usuarios. Esto facilita depurar errores y reduce su curva de aprendizaje.
- Es extensible: las cabeceras de HTTP han hecho que este protocolo sea fácil de ampliar y de experimentar con él.
- Es un protocolo con sesiones pero sin estados.

En la figura 2.1 se presenta un ejemplo de comunicación HTTP en el que puede observarse como interactúa un cliente con el servidor.

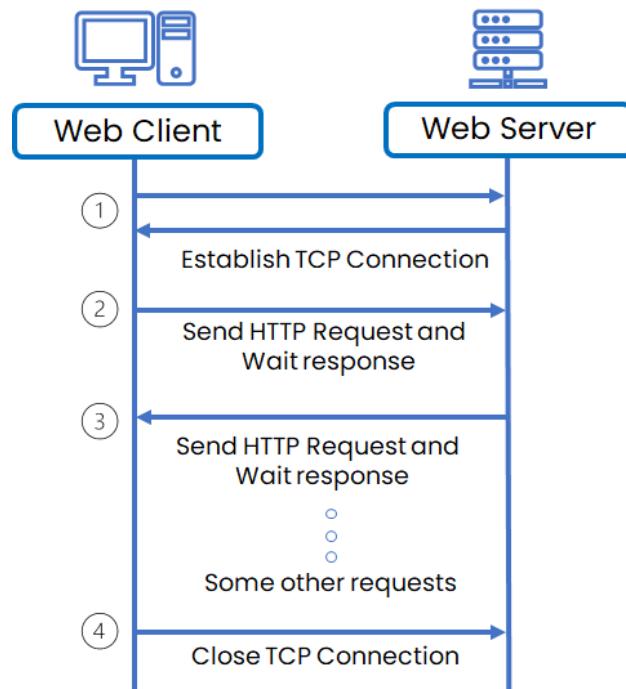


FIGURA 2.1. Ejemplo de comunicación en HTTP¹.

La versión encriptada de HTTP se llama HTTPS (del inglés *Hypertext Transfer Protocol Secure*) [33]. Normalmente utiliza TLS para cifrar toda la comunicación entre un cliente y un servidor. Esta conexión segura permite a los clientes intercambiar datos confidenciales de forma segura con un servidor, por ejemplo, para actividades bancarias o compras en línea.

2.1.2. *Message Queue Telemetry Transport*

MQTT es un protocolo de mensajería basado en estándares o conjuntos de reglas, que se utiliza para la comunicación de un equipo a otro.

Los sensores inteligentes, los dispositivos portátiles y otros dispositivos IoT (del inglés *Internet of Things*) [34] generalmente tienen que transmitir y recibir datos a través de una red con recursos restringidos y un ancho de banda limitado. Estos dispositivos IoT utilizan MQTT para la transmisión de datos, ya que resulta fácil de implementar y puede comunicar datos IoT de manera eficiente. MQTT admite mensajería bidireccional entre dispositivos y una plataforma *cloud*.

El protocolo MQTT funciona según los principios del modelo de publicación o suscripción. Sus componentes son los siguientes:

- **Cliente:** es cualquier dispositivo que ejecuta una biblioteca MQTT. Si el cliente envía mensajes, actúa como editor y si recibe mensajes, actúa como receptor.
- **Agente o broker:** es el sistema que coordina los mensajes entre los diferentes clientes. Además, se encarga de otras tareas como la autorización y autenticación de los clientes, identificar a los clientes suscritos a cada tema y el

¹Imagen tomada de
<https://www.ionos.co.uk/digitalguide/hosting/technical-matters/what-is-http/>

envío de sus mensajes y controlar los mensajes perdidos y las sesiones del cliente.

- Conexión: los clientes inician la conexión al enviar un mensaje CONECTAR al agente MQTT. El agente confirma que se ha establecido una conexión al responder con un mensaje CONNACK. Los clientes nunca se conectan entre sí, solo con el agente.

En la figura 2.2 se presenta un ejemplo de comunicación MQTT en el que puede observarse como interactúan cada uno de los componentes del protocolo.

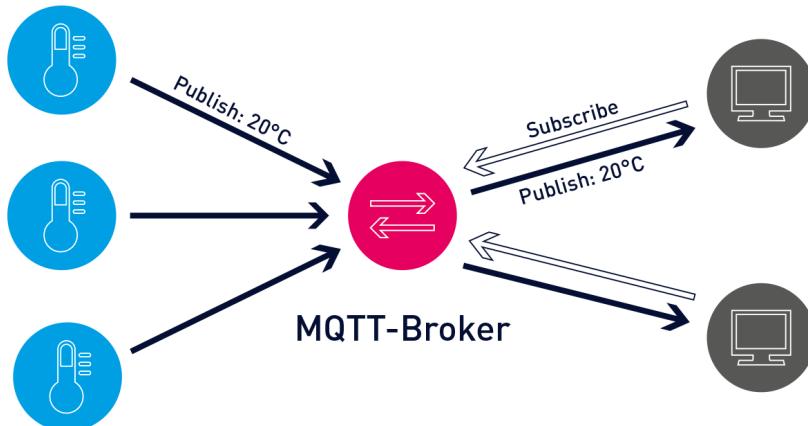


FIGURA 2.2. Ejemplo de comunicación en MQTT².

La comunicación MQTT utiliza el protocolo SSL para proteger los datos confidenciales que transmiten los dispositivos IoT. Puede implementar la identidad, la autenticación y la autorización entre los clientes y el agente mediante certificados SSL y contraseñas. El agente MQTT normalmente autentica a los clientes mediante sus contraseñas e identificadores únicos. En la mayoría de las implementaciones, el cliente autentica el servidor con certificados o búsquedas de DNS [35]. También puede implementar protocolos de cifrado con MQTT.

2.1.3. WebSocket

WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y full-dúplex [36] sobre un único *socket* TCP [37]. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor. Además, puede utilizar SSL para las conexiones seguras y la transmisión de datos.

WebSocket utiliza un secuencia de solicitud-respuesta HTTP estándar para establecer una conexión. Una vez formada, la API WebSocket proporciona una interfaz de lectura y escritura de datos entre las partes de modo dúplex asíncrono. También proporciona una interfaz para el cierre asíncrono de la conexión desde cualquier lado.

En la figura 2.3 se presenta un ejemplo de comunicación WebSocket, en el que puede observarse cómo interactúan el cliente y el servidor.

²Imagen tomada de <https://www.paessler.com/es/it-explained/mqtt>

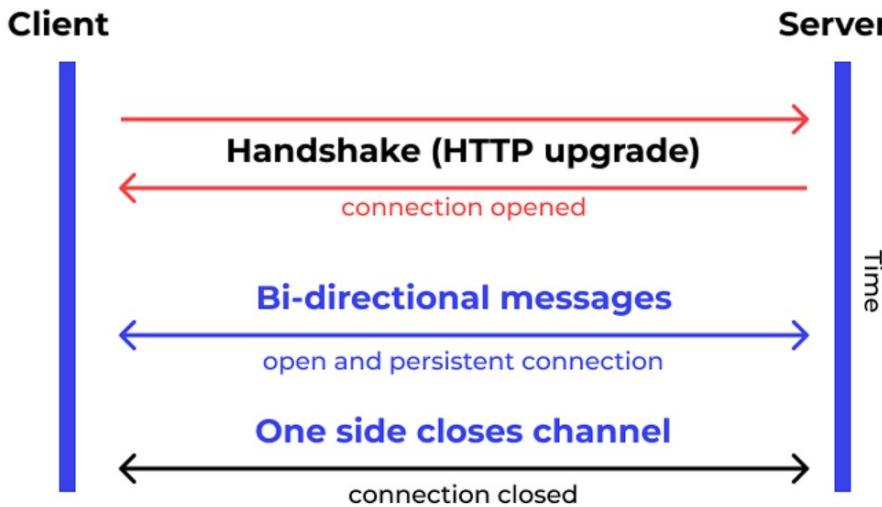


FIGURA 2.3. Ejemplo de comunicación en WebSocket³.

2.2. Componentes de *hardware*

2.2.1. Microcontrolador

ESP32 [38] es la denominación de una familia de chips SoC (del inglés *system on a chip*) [39] de bajo coste y consumo de energía. Los integrados cuentan con tecnología Wi-Fi y Bluetooth [40] de modo dual integrada. El ESP32 emplea un microprocesador 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.

La elección del ESP32 se debe a los siguientes factores:

- Posee una alta disponibilidad en el mercado nacional.
- Tiene una alta cantidad de bibliotecas disponibles.
- Es económico.
- Es fácil de usar.

En la tabla 2.1 se pueden observar las bibliotecas de terceros utilizadas en el trabajo.

³Imagen tomada de
<https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is>

TABLA 2.1. Bibliotecas de terceros utilizadas.

Nombre	Descripción
knolleary/PubSubClient [41]	Permite utilizar el protocolo MQTT
adafruit/Adafruit Unified Sensor [42]	Dependencia para utilizar el sensor DHT22
adafruit/DHT sensor library [43]	Permite leer las mediciones del sensor DHT22
bblanchon/ArduinoJson [44]	Permite utilizar JSON
phoenix1747/MQ135 [45]	Permite leer las mediciones del sensor MQ135
claws/BH1750 [46]	Permite leer las mediciones del sensor Bh1750
milesburton/DallasTemperature [47]	Permite leer las mediciones del sensor DS18B20

2.2.2. Sensor digital de temperatura y humedad

El módulo DHT22 [48] es un sensor digital de temperatura y humedad relativa. Integra un sensor capacitivo de humedad y un termistor [49] para medir el aire circundante y entrega la información mediante una señal digital en el pin de datos.

Es utilizado principalmente en aplicaciones de control automático de temperatura, aire acondicionado y monitoreo ambiental en agricultura. La mayor desventaja del sensor es que sólo se puede obtener nuevos datos cada 2 segundos.

La fotografía del sensor DHT22 se presenta en la figura 2.4.

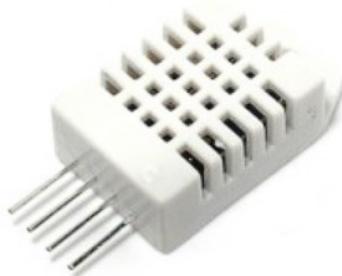


FIGURA 2.4. Fotografía del sensor DHT22⁴.

La elección del sensor DHT22 es debida a su mejor resolución y precisión frente a otros sensores, por ejemplo el DHT11 [50].

2.2.3. Sensor de intensidad de luz

El módulo Bh1750 [51] es un sensor de intensidad de luz que esta compuesto por un conversor A/D (del inglés *Analog to Digital*) [52] de 16 bits. La principal ventaja del sensor es que brinda la intensidad luminosa en unidades de Lux.

⁴Imagen tomada de
<https://www.openhacks.com/page/productos/id/84/title/Sensor-de-Humedad-y-Temperatura-DHT22>

La fotografía del sensor Bh1750 se presenta en la figura 2.5.

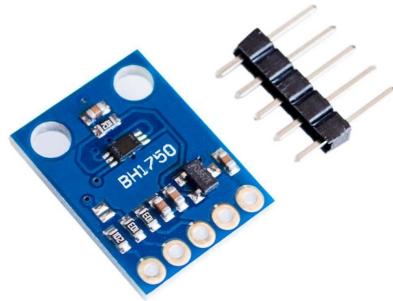


FIGURA 2.5. Fotografía del sensor Bh1750⁵.

La elección del sensor Bh1750 es debida a su alta precisión y bajo costo en el mercado nacional. Otro factor importante en la decisión fue el poder brindar la intensidad de luz en Lux, sin necesidad de hacer alguna conversión.

2.2.4. Sensor de calidad del aire

El módulo MQ135 [53] es un sensor de calidad del aire con alta sensibilidad al Amoníaco (NH₃), óxidos de nitrógeno (NO_x), alcohol, sulfuros, benceno (C₆H₆), CO₂, humo y otros gases nocivos. Posee una salida analógica para indicar la concentración de gas en el ambiente y una digital que baja a 0 cuando la concentración de gas supera un nivel prefijado.

La fotografía del sensor MQ135 se presenta en la figura 2.6.



FIGURA 2.6. Fotografía del sensor MQ135⁶.

La elección del sensor MQ135 es debida a su bajo costo y a la baja oferta de sensores similares en el mercado nacional.

⁵Imagen tomada de
<https://candy-ho.com/producto/modulo-sensor-digital-luz-ambiente-lux-bh1750-arduino/>

⁶Imagen tomada de
<https://www.micro-log.com/sensores-modulos-y-shields/3521-sensor-de-calidad-del-aire.html>

2.2.5. Sensor de flujo

El módulo YF-S201B [54] es un sensor de flujo que se utiliza para la medición de caudal o gasto volumétrico de un fluido en tuberías de 1/2" de diámetro.

Este sensor consiste en un cuerpo de válvula de plástico, un rotor y un sensor de efecto Hall [55]. El rotor gira cuando el agua/líquido fluye a través de la válvula y su velocidad es directamente proporcional a la velocidad de flujo. El sensor de efecto Hall proporciona un impulso eléctrico con cada revolución del rotor. Este módulo sensor de flujo de agua puede ser fácilmente conectado con microcontroladores.

La fotografía del sensor YF-S201B se presenta en la figura 2.7.



FIGURA 2.7. Fotografía del sensor YF-S201B⁷.

La elección del sensor YF-S201B es debida a su bajo costo en el mercado nacional. Otros factores influyentes en la decisión fueron el bajo consumo y su facilidad de uso respecto a otros sensores similares.

2.2.6. Sensor digital de temperatura para líquidos

El módulo DS18B20 [56] sumergible es un sensor digital de temperatura que utiliza el protocolo 1-Wire [57] para comunicarse, el cual necesita solo un pin de datos y permite conectar más de un sensor en el mismo bus. La temperatura de funcionamiento es desde los -55 °C hasta los 125 °C y con una resolución programable desde 9 hasta 12 bits.

La fotografía del sensor DS18B20 sumergible se presenta en la figura 2.8.

⁷Imagen tomada de
<https://naylampmechatronics.com/sensores-líquido/108-sensor-de-flujo-de-agua-12-yf-s201.html>



FIGURA 2.8. Fotografía del sensor DS18B20 sumergible⁸.

La elección del sensor DS18B20 es debida a su facilidad de uso y amplio rango de medición.

2.2.7. Módulo de *relay*

El *relay* es un interruptor eléctrico para controlar el flujo de corriente. Puede estar cerrado o abierto con lo que permite o impide la circulación respectivamente.

La fotografía del módulo de *relay* se presenta en la figura 2.9.



FIGURA 2.9. Fotografía del módulo de *relay*⁹.

La elección de incorporar un módulo de *relay* es debida a que se quiere ofrecer al usuario del sistema un mecanismo para accionar dispositivos externos.

⁸Imagen tomada de <https://www.hobbytronica.com.ar/mla-903024959-ds18b20>

⁹Imagen tomada de <https://candy-ho.com/producto/modulo-rele-5v>

2.3. Herramientas de software

2.3.1. Progressive Web Apps

Las PWA (del inglés *Progressive Web Apps*) [58] son aplicaciones web que utilizan APIs y funciones emergentes del navegador web junto a una estrategia tradicional de mejora progresiva para ofrecer una aplicación nativa.

Para poder llamar PWA a una aplicación web, debe tener las siguientes características: uso de HTTPS, uno o más *service workers* [59] y un archivo de manifiesto [60].

Mediante los *service workers* y otras tecnologías, las PWA pueden seguir ejecutándose en segundo plano sin tener que vivir dentro del navegador. Además, pueden instalarse en celulares y computadoras.

En la tabla 2.2 se puede observar una comparación entre aplicaciones nativas, aplicaciones web y PWA.

TABLA 2.2. Comparación entre distintos tipos de aplicaciones.

Descripción	Web	Nativa	PWA
Instalable	No	Sí	Sí
Multiplataforma	Sí	No	Sí
Notificaciones <i>push</i>	No	Sí	Sí
Uso sin Internet	No	Sí	Sí
Diseño <i>responsive</i> [61]	Sí	Sí	Sí
Acceso al GPS	No	Sí	Sí

2.3.2. Framework del frontend

Angular [62] es un *framework* [63] para aplicaciones web desarrollado en TypeScript [64]. Es de código abierto y mantenido por Google y se utiliza para crear y mantener aplicaciones web de una sola página [65]. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de MVC (del inglés *Model-View-Controller*) [66], en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de JavaScript [67].

Angular se basa en clases tipo componente, cuyas propiedades son las usadas para hacer el *binding* de los datos. Dichas clases tienen propiedades y métodos.

La elección de Angular se debe a los siguientes factores:

- Posee alta escalabilidad.
- Al ser un *framework* contiene muchas funcionalidades incorporadas, por ejemplo el manejo de rutas.
- Es fácil de adaptar a PWA.
- Tiene una documentación detallada.

- Posee una amplia variedad de bibliotecas disponibles.
- Tiene una comunidad activa.

En la tabla 2.3 se pueden observar las bibliotecas de terceros utilizadas en la solución de Angular.

TABLA 2.3. Bibliotecas de terceros utilizadas.

Nombre	Descripción
@angular/material [68]	Permite utilizar Material Design [69]
@swimlane/ngx-charts [70]	Permite utilizar gráficos D3.js [71]
socket.io-client [72]	Permite utilizar Socket.IO [73]
ngx-cookieconsent [74]	Permite generar una alerta del uso de cookies [75]
ngx-countup [76]	Permite generar animaciones numéricas
lite-server [77]	Permite emular un despliegue en un entorno local
compression [78]	Permite habilitar g-zip [79] al usar lite-server

2.3.3. Entorno en tiempo de ejecución del *backend*

Node.js [80] es un entorno de código abierto multiplataforma que ejecuta código JavaScript fuera de un navegador. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web.

La elección de Node.js se debe a los siguientes factores:

- Tiene la capacidad para manejar aplicaciones de alta concurrencia.
- Tiene la capacidad para escalar horizontalmente.
- Posee una amplia variedad de bibliotecas y paquetes disponibles.
- Brinda la posibilidad de usar TypeScript, tanto en el cliente como en el servidor.
- Tiene buen rendimiento.
- Tiene una comunidad activa.

En la tabla 2.4 se pueden observar las bibliotecas de terceros utilizadas en la solución de Node.js.

TABLA 2.4. Bibliotecas de terceros utilizadas.

Nombre	Descripción
bcryptjs [81]	Permite realizar encriptaciones [82] de contraseñas
aedes [83]	Permite generar un <i>broker</i> MQTT
compression [78]	Permite habilitar g-zip
cors [84]	Permite habilitar CORS [85]
dotenv [86]	Permite utilizar variables de entorno [87]
express [88]	Permite utilizar el framework Express [89]
express-rate-limit [90]	Permite limitar la cantidad de <i>requests</i> permitidas
jsonwebtoken [91]	Permite utilizar JSON Web Tokens [92]
mongoose [93]	Permite utilizar el ORM [94] mongoose [95]
node-cron [96]	Permite programar tareas
nodemailer [97]	Permite enviar emails
nodemailer-express-handlebars [98]	Permite agregar código HTML a los emails
web-push [99]	Permite enviar notificaciones push
typescript [100]	Permite utilizar el lenguaje TypeScript
nodemon [101]	Permite resetear automáticamente el servidor si el código cambia
supertest [102]	Permite probar servicios HTTP
jest [103]	Permite el uso del framework Jest
ts-jest [104]	Permite usar TypeScript con el framework Jest

2.3.4. Base de datos

MongoDB [105] es un sistema de base de datos NoSQL (del inglés *Not Only Structured Query Language*) [106], orientado a documentos [107] y de código abierto.

En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales [108], MongoDB guarda estructuras de datos BSON (del inglés *Binary JavaScript Object Notation*) [109] con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

En la figura 2.10 se presenta un ejemplo de modelado de datos en MongoDB, en el que puede observarse el formato de un documento BSON.

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

Diagrama que muestra un ejemplo de modelado de datos en MongoDB en formato BSON. Se muestra un objeto JSON con campos: name, age, status y groups. Cada campo tiene una flecha apuntando a él con la etiqueta 'field: value'.

FIGURA 2.10. Ejemplo de modelado de datos en MongoDB ¹⁰.

La elección de MongoDB se debe a los siguientes factores:

- Posee alta escalabilidad horizontal.
- Es altamente flexible en el modelo de datos.
- Tiene buen rendimiento.
- Es fácil de integrar con otras tecnologías.
- Tiene una comunidad activa.

2.3.5. Framework de pruebas del frontend

Jasmine [110] es un marco de pruebas de JavaScript diseñado para facilitar el desarrollo de pruebas automatizadas para aplicaciones web y aplicaciones escritas en JavaScript. Jasmine se basa en el patrón de pruebas BDD (del inglés *Behavior-Driven Development*) [111] y se utiliza comúnmente para aplicaciones escritas en Angular.

Jasmine proporciona un conjunto de funciones y sintaxis para escribir pruebas que sean fáciles de leer y comprender.

La elección de Jasmine es debida a su integración incluida en Angular y a su velocidad de ejecución.

2.3.6. Framework de pruebas del backend

Jest [112] es un marco de pruebas de JavaScript desarrollado por Facebook. Al igual que Jasmine, Jest se utiliza para escribir y ejecutar pruebas automatizadas para aplicaciones web y aplicaciones escritas en JavaScript. Jest también se basa en el patrón de pruebas BDD y proporciona una sintaxis de prueba intuitiva y fácil de leer.

Jest se destaca por su velocidad de ejecución, que se debe en parte a su capacidad de realizar pruebas en paralelo y al uso de técnicas avanzadas de procesamiento en segundo plano. Jest también viene con una variedad de herramientas y utilidades para ayudar a los desarrolladores a escribir y ejecutar pruebas, incluyendo funciones de aserción para comparar valores, funciones para configurar y limpiar el estado de las pruebas y utilidades para simular el comportamiento de objetos y funciones.

Además, Jest tiene una serie de características únicas, como la capacidad de realizar instantáneas [113] de componentes de la interfaz de usuario y comprobarlos automáticamente para detectar cambios, así como la capacidad de integrarse con herramientas populares de automatización de tareas como Webpack [114] y Babel [115].

La elección de Jest se debe a los siguientes factores:

- Es fácil de usar.
- Es fácil de integrar con Node.js.
- Es rápido y eficiente.

¹⁰Imagen tomada de <https://www.mongodb.com/docs/manual/core/document/>

- Tiene una función integrada para realizar cobertura de código.
- Tiene una gran comunidad activa.

2.3.7. Software de auditorías del frontend

Lighthouse [116] es una herramienta automatizada de código abierto para mejorar la calidad de las páginas web. Se puede ejecutar contra cualquier página web pública o que requiera autenticación. Además, tiene auditorías de rendimiento, accesibilidad, PWA, mejores prácticas y SEO (del inglés *Search Engine Optimization*).

En la figura 2.11 se presenta un ejemplo de una auditoría realizada con Lighthouse, en el que puede observarse las diferentes áreas que se evalúan.

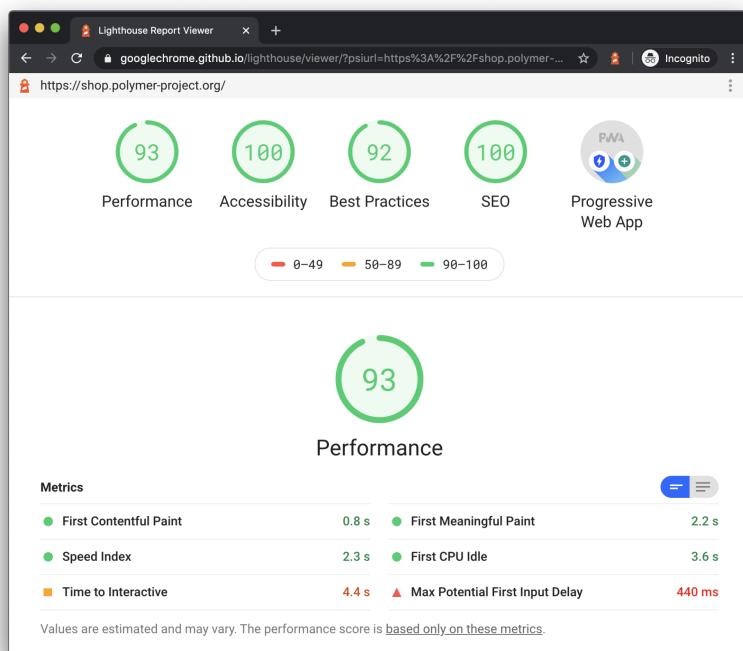


FIGURA 2.11. Ejemplo de auditoría realizada en Lighthouse¹¹.

La elección de Lighthouse como herramienta de auditorías es debida a su integración incorporada en el navegador y sus pruebas extensivas en diferentes áreas. Otro factor influyente en la decisión fue que además de dar un amplio diagnóstico, también brinda posibles soluciones para cada uno de los problemas encontrados.

¹¹Imagen tomada de <https://es.semrush.com/blog/como-utilizar-google-lighthouse/>

Capítulo 3

Diseño e implementación

En este capítulo se detallan los criterios de diseño que se tomaron para el desarrollo del sistema y los pasos seguidos para su implementación.

3.1. Arquitectura del sistema

Una instalación está dividida en diferentes zonas de cultivos y en cada una se instala un microcontrolador ESP32 que se conecta usando MQTT con el *broker* Aedes [83]. En un sentido se envía la información de los sensores y en el otro los comandos para controlar los actuadores. El diagrama de la solución puede verse en la figura 3.1.

El broker se encarga de autenticar y autorizar al microcontrolador. Una vez hecho esto, debe analizar la información recibida, validarla y en caso de ser necesario enviarla al DaaS de MongoDB para que sea almacenada. Además, el *broker* por medio del protocolo WebSocket puede enviar los datos en tiempo real a la PWA. El servidor y el DaaS son compartidos entre todos los usuarios del sistema.

La API REST gestiona el acceso a los datos almacenados en el DaaS por medio de sus *endpoints*. Estos últimos son ejecutados por la PWA por medio de HTTPS.

Por último, la PWA permite al usuario interactuar con el sistema por medio de sus interfaces.

Es importante destacar que todas las conexiones realizadas entre los componentes del sistema utilizan TLS para garantizar la seguridad en la comunicación. Además, todos los certificados TLS utilizados en el trabajo fueron generados mediante la herramienta openssl [117] y se utilizó como DNS la IP de la computadora en la que se ejecutan los proyectos.

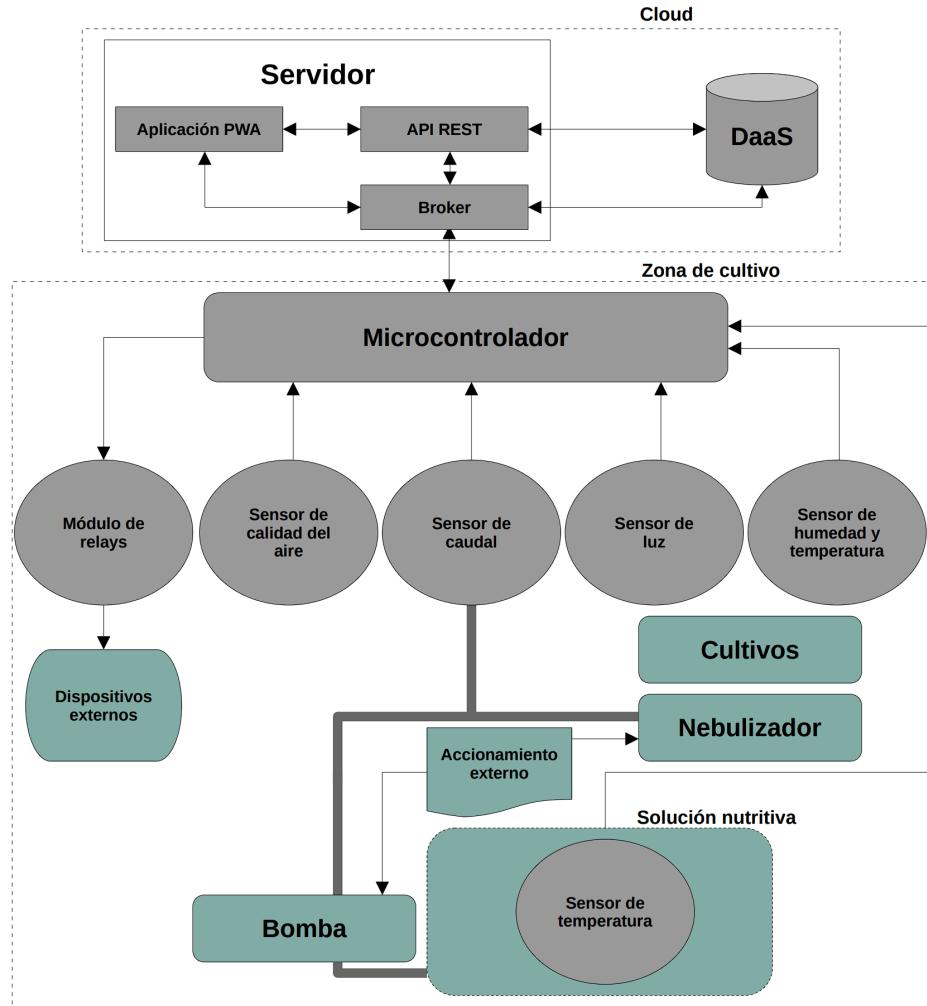


FIGURA 3.1. Diagrama en bloques de la solución.

3.1.1. Observaciones

Inicialmente, en la planificación del trabajo se había planteado utilizar una aplicación SSR. Sin embargo, a la hora de comenzar el desarrollo del *frontend* se decidió que una PWA sería más adecuada, ya que otorga la posibilidad de usar ciertas funcionalidades que son útiles para el sistema, por ejemplo tener un *assets cache*, utilizar notificaciones *push* y permitir instalar la aplicación en los dispositivos del usuario.

También se realizaron modificaciones en los datos a almacenar en el DaaS para cada una de las colecciones con el fin de optimizar las consultas.

Estos cambios mencionados generaron modificaciones en algunos requerimientos y tareas del trabajo.

3.2. Modelo de datos

En esta sección se describen las diferentes colecciones que se almacenan en la base de datos de MongoDB. Con el objetivo de brindar una representación visual de

las mismas, se presenta su estructura por medio de las interfaces utilizadas en la API REST.

3.2.1. Colección *Users*

Es la colección que contiene los datos de los usuarios. Se presenta su estructura en la figura 3.2. El campo *verified* sirve para indicar si el usuario verificó su *email*. La propiedad *passwordHash* almacena el *hash* de la contraseña. El atributo *pushSubscription* es un objeto embebido dentro del documento y permite enviar notificaciones *push* al usuario.

```
interface IKeysSubscription {
  _id: Types.ObjectId;
  auth: string;
  p256dh: string;
}

interface IPushSubscription {
  _id: Types.ObjectId;
  endpoint: string;
  keys: IKeysSubscription;
  expirationTime?: number;
}

export interface IUser {
  _id: Types.ObjectId;
  email: string;
  passwordHash: string;
  verified: boolean;
  firstName?: string;
  lastName?: string;
  dateCreated: Date;
  pushSubscription?: IPushSubscription;
}
```

FIGURA 3.2. Interface de la colección *Users*.

3.2.2. Colección *Zones*

Es la colección que contiene los datos de las zonas de cultivo. Se presenta su estructura en la figura 3.3. Los campos *user* y *lastMeasurement* son identificadores para hacer referencia a un documento de la colección *Users* y *Measurements*. El atributo *device* es un objeto embebido dentro del documento. Este último tendrá las propiedades que indican el nombre, *hash* de la contraseña, identificador de zona, identificador de usuario, alarmas asignadas junto con sus acciones automatizadas, *relays* asignados y configuración de las notificaciones.

```

export interface IZone {
  _id: Types.ObjectId;
  name: string;
  device: IDevice;
  user: Types.ObjectId / IUser;
  lastMeasurement?: Types.ObjectId / IMeasurement;
}

export interface IDevice {
  _id: Types.ObjectId;
  name: string;
  passwordHash: string;
  zone: Types.ObjectId / IZone;
  user: Types.ObjectId / IUser;
  alarms?: IAlarm[];
  relays: IRelay[];
  emailNotificationsEnabled: boolean;
  desktopNotificationsEnabled: boolean;
}

export enum IAlarmTypeEnum {
  "temperatureRoom" = "Temperature room",
  "humidityRoom" = "Humidity room",
  "temperatureFluid" = "Temperature fluid",
  "lightLevel" = "Light level",
  "co2Level" = "CO2 level",
  "flowRate" = "Flow rate",
  "totalLitres" = "Total litres",
}

interface IAlarm {
  _id: Types.ObjectId;
  type: IAlarmTypeEnum;
  maxValue: number;
  minValue: number;
  actions?: IAction[];
}

interface IRelay {
  _id: Types.ObjectId;
  name: string;
  value: boolean;
}

export enum ITriggerEnum {
  "maxValue" = "maxValue",
  "minValue" = "minValue",
  "any" = "any",
}

export enum IActionEnum {
  "turnOn" = "turnOn",
  "turnOff" = "turnOff",
}

export interface IAction {
  _id: Types.ObjectId;
  trigger: ITriggerEnum;
  action: IActionEnum;
  relays: number[];
}

```

FIGURA 3.3. Interface de la colección Zones.

3.2.3. Colección Notifications

Es la colección que contiene los datos de las notificaciones de los usuarios. Se presenta su estructura en la figura 3.4. Los campos *zone*, *device* y *user* son identificadores para hacer referencia a un documento de la colección *Zones*, *Devices*

y *Users*. El atributo *observation* almacena un mensaje que indica la condición que activó la alarma.

```
export interface INotification {
  _id: Types.ObjectId;
  observation: string;
  active: boolean;
  dateRead?: Date;
  dateCreated: Date;
  zone: Types.ObjectId / IZone;
  device: Types.ObjectId / IDevice;
  user: Types.ObjectId / IUser;
}
```

FIGURA 3.4. Interface de la colección *Notifications*.

3.2.4. Colección *Measurements*

Es la colección que contiene los datos de las mediciones de los sensores. Se presenta su estructura en la figura 3.5. Las propiedades *device*, *zone* y *user* son identificadores para hacer referencia a un documento de la colección *Devices*, *Zones* y *Users*. El atributo *timeReceived* contiene la fecha y hora de cuando se creó la medición.

```
export interface IMeasurement {
  _id: Types.ObjectId;
  temperatureRoom: number;
  humidityRoom: number;
  temperatureFluid: number;
  co2Level: number;
  lightLevel: number;
  flowRate: number;
  totalLitres: number;
  timeReceived: Date;
  device: Types.ObjectId / IDevice;
  zone: Types.ObjectId / IZone;
  user: Types.ObjectId / IUser;
}
```

FIGURA 3.5. Interface de la colección *Measurements*.

3.3. Desarrollo del *backend*

El *backend* fue desarrollado en Node.js mediante TypeScript y Express.

En el código 3.1 se presenta la inicialización del *backend* con certificados TLS. En las líneas 1 y 2 se obtienen los certificados. En las líneas 4 a 9 se crea el servidor con HTTPS con los certificados TLS.

```
1 const privateKey = readFileSync("./ssl/localhost.key");
2 const certificate = readFileSync("./ssl/localhost.crt");
3
4 const server = createServer(
5   { key: privateKey, cert: certificate },
```

```

6   app
7 ).listen(port, () => {
8   console.log(`[server]: Server is running at https://localhost:${port}
9   `);

```

CÓDIGO 3.1. Inicialización del *backend* con TLS.

El *backend* utiliza JWT (del inglés *JSON Web Tokens*) para la autorización y autenticación de los usuarios.

La autorización ocurre cuando un usuario inicia sesión en el sistema y el *backend* le brinda un JWT firmado con una clave privada. En la figura 3.6 se visualiza un diagrama de secuencia del proceso.

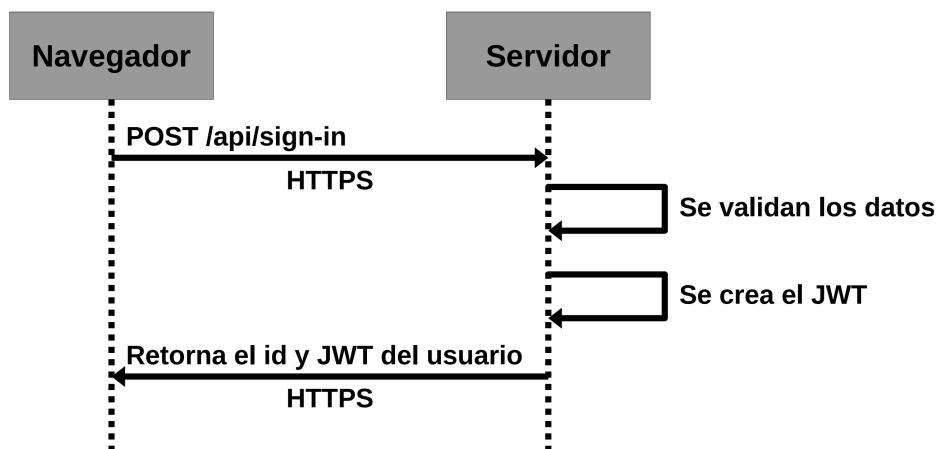


FIGURA 3.6. Diagrama de secuencia de la autorización de usuarios.

Si el usuario no dispone de credenciales válidas, la API responde con un código de error 400 y una breve descripción del error. Si se produce un error interno en el servidor, la API responde con un código de error 500.

En el código 3.2 se presenta el proceso de autorización de los usuarios. En la línea 1 se obtiene la clave privada mediante las variables de entorno. En las líneas 6 a 8 se firma el *payload* con la clave privada, también se asigna la expiración del JWT a siete días. En la línea 10 se le entrega al usuario el JWT junto con su id.

```

1 const accessTokenSecret = process.env["ACCESS_TOKEN_SECRET"] as string;
2 const payload = {
3   _id: user._id,
4   email: user.email,
5 };
6 const accessToken = jwt.sign(payload, accessTokenSecret, {
7   expiresIn: "7d",
8 });
9
10 return response.status(200).json({ _id: user._id, accessToken });

```

CÓDIGO 3.2. Autorización de usuarios.

La autenticación ocurre cuando un usuario quiere acceder a un recurso protegido, por ejemplo un *endpoint* y el *backend* debe validar el JWT del usuario para

determinar si puede acceder o no. En la figura 3.7 se visualiza un diagrama de secuencia del proceso.

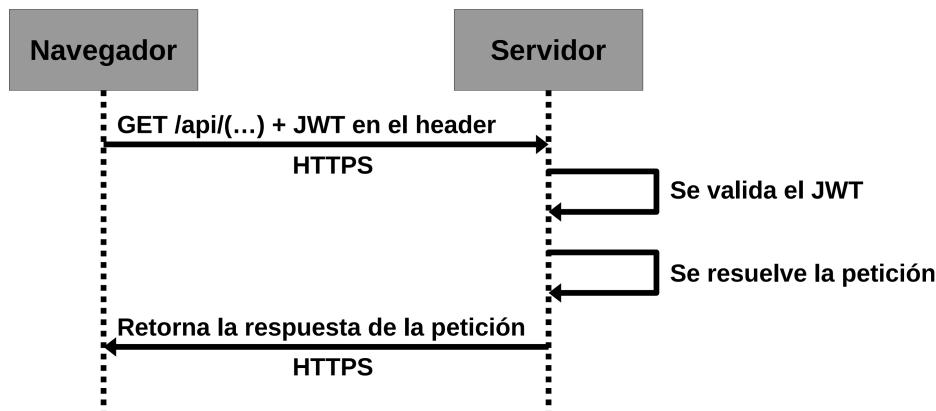


FIGURA 3.7. Diagrama de secuencia de la autenticación de usuarios.

Todos los *endpoints* del sistema, excepto el de registro de usuarios y el de inicio de sesión, requieren el JWT del usuario para poder ser consultados. Si el usuario no envía el JWT, la API responde con un código de error 401.

En el código 3.3 se presenta el proceso de autenticación de los usuarios mediante un *middleware* que protege los *endpoints*. En las líneas 6 a 11 se obtiene el JWT del *authorization header* [118] de la *request* y se valida que no sea nulo. En la línea 14 se verifica el JWT mediante la biblioteca jsonwebtoken. En las líneas 15 y 16 se guarda el id e *email* del usuario en la respuesta HTTP.

```

1 export const authorization = (
2   request: Request,
3   response: Response,
4   next: NextFunction
5 ) => {
6   const accessToken = request.headers.authorization?.split(" ")[1];
7   const accessTokenSecret = process.env["ACCESS_TOKEN_SECRET"] as string
8   ;
9   if (!accessToken) {
10     return response.sendStatus(401);
11   }
12
13   try {
14     const payloadData = verifyToken(accessToken, accessTokenSecret);
15     response.locals._id = payloadData._id;
16     response.locals.email = payloadData.email;
17
18     return next();
19   } catch {
20     return response.sendStatus(401);
21   }
22 };
  
```

CÓDIGO 3.3. Autenticación de usuarios mediante un *middleware*.

En el código 3.4 se presenta un ejemplo de *endpoint* que requiere la autenticación del usuario. En las líneas 1 y 2 se establece el nombre de la ruta junto con el método HTTP. En la línea 3 se aplica un *middleware* a la ruta que verifica el *token* del

usuario. En la línea 6 se obtiene su id. En las líneas 7 a 9 se utiliza el ORM (del inglés *Object-Relational-Mapping*) Mongoose para obtener las zonas que le pertenecen. En la línea 10 se le entrega una lista de sus zonas.

```

1 zoneRouter.get(
2   '/',
3   authorization,
4   async (request: Request, response: Response) => {
5     try {
6       const userId = response.locals._id;
7       const zones = await Zone.find({ user: userId })
8         .populate("lastMeasurement")
9         .exec();
10      return response.status(200).json(zones);
11    } catch (error) {
12      return response.status(500).json(error);
13    }
14  }
15 );

```

CÓDIGO 3.4. *Endpoint* que requiere autenticación.

En la tabla 3.1 se pueden observar los *endpoints* disponibles.

TABLA 3.1. *Endpoints* disponibles.

Endpoint	Método	Descripción
/auth/sign-up	POST	Permite registrar un usuario
/auth/sign-in	POST	Permite iniciar sesión
/auth/verifyToken	GET	Permite verificar al usuario
/auth/profile	GET	Permite obtener los datos del usuario
/auth/profile	PATCH	Permite actualizar los datos del usuario
/notifications	GET	Permite obtener las notificaciones
/notifications/markAsRead/:id	PATCH	Permite marcar una notificación como leída
/notifications/subscribe	PATCH	Permite suscribirse a las notificaciones
/measurements/zones/:id	POST	Permite simular una medición en una zona
/measurements/zones/:id	GET	Permite obtener las mediciones de una zona
/zones	GET	Permite obtener las zonas
/zones/:id	GET	Permite obtener una zona
/zones	POST	Permite crear una zona
/zones	PATCH	Permite actualizar los datos de una zona
/zones/:id	DELETE	Permite eliminar una zona
/zones/relays/:id	PATCH	Permite actualizar los datos de un <i>relay</i>

En la figura 3.8 se presenta la estructura de directorios. La descripción de los contenidos de las carpetas y archivos relevantes es:

- models: contiene las interfaces que permiten al ORM Mongoose tipificar las colecciones de MongoDB.
- routes: contiene los *endpoints*.
- schemas: contiene los esquemas que permiten al ORM Mongoose crear las colecciones en MongoDB.
- ssl: contiene los certificados TLS.
- utils: contiene un *middleware* para autorizar los *endpoints* necesarios. Además, contiene funciones que se reutilizan en todo el proyecto.
- .env: contiene las variables de entorno.
- broker.ts: contiene la inicialización y configuración del *broker* Aedes.
- index.ts: contiene la inicialización y configuración del *backend*.
- jest.config.js: contiene la configuración de Jest.
- ts.config.json: contiene la configuración de TypeScript.

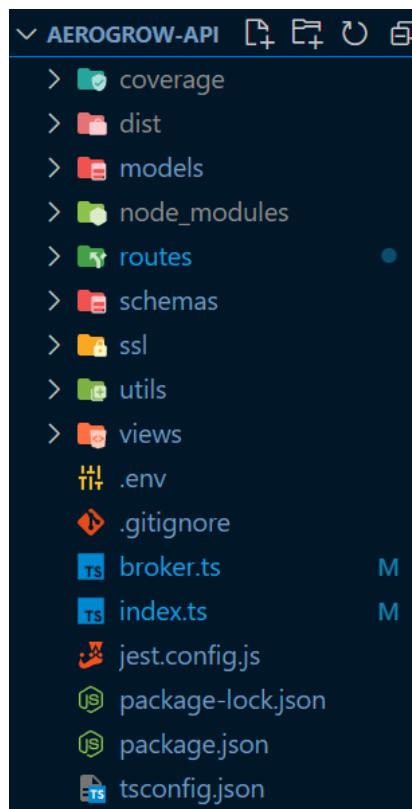


FIGURA 3.8. Estructura de directorios.

3.4. Desarrollo del *frontend*

El *frontend* del trabajo fue desarrollado en TypeScript con Angular y como *framework* de estilos se utilizó Angular Material.

La estrategia de diseño que se utilizó fue la de crear una PWA con un enfoque de diseño *responsive* [61].

Para obtener el mejor rendimiento posible en la aplicación se aplicaron diferentes estrategias de optimización [119] [120], entre ellas destacan las siguientes:

- *Lazy loading* de rutas.
- *Lazy loading* de imágenes.
- Uso de *ChangeDetectionStrategy.onPush* en los componentes.
- *Assets cache*.
- Evitar llamadas de funciones en las vistas HTML.
- Utilizar el *guard* canLoad.
- Evitar el uso de bibliotecas que no sean esenciales para el trabajo.
- Evitar suscripciones manuales a un *Observable* [121].

En la figura 3.9 se presenta la configuración para utilizar certificados TLS.

```
"serve": {
  "builder": "@angular-devkit/build-angular:dev-server",
  "options": {
    "ssl": true,
    "sslCert": "ssl/localhost.crt",
    "sslKey": "ssl/localhost.key"
  },
  "configurations": {
    "production": {
      "browserTarget": "aerogrow-pwa:build:production"
    },
    "development": {
      "browserTarget": "aerogrow-pwa:build:development"
    }
  },
  "defaultConfiguration": "development"
},
```

FIGURA 3.9. Uso de certificados TLS.

La PWA utiliza un *guard* para la autenticación de las rutas. En el código 3.5 se presenta el proceso de autenticación de las rutas. En la línea 7 se obtiene el estado de la sesión del usuario. En las líneas 9 a 13 se verifica su estado: si inició sesión puede entrar a la ruta, caso contrario se le redirecciona a la pantalla del *login*.

```
1 export const authGuard = (
2   route: ActivatedRouteSnapshot,
3   state: RouterStateSnapshot
4 ) => {
5   const router = inject(Router);
6   const authService = inject(AuthService);
7   const isLoggedIn = authService.isLoggedIn();
8
9   return isLoggedIn
10  ? true
11  : router.navigate(['/sign-in'], {
12    queryParams: { returnUrl: state.url },
13  });
14};
```

CÓDIGO 3.5. Autenticación de rutas.

En la tabla 3.2 se pueden observar las rutas disponibles.

TABLA 3.2. Rutas disponibles.

Ruta	Descripción
/	Pantalla de listado de zonas
/profile	Pantalla de edición de un usuario
/zones	Pantalla de listado de zonas
/zones/create	Pantalla de creación de una zona
/zones/edit/:id	Pantalla de edición de una zona
/notifications	Pantalla de listado de notificaciones
/dashboard/:id	Pantalla de <i>dashboard</i> de una zona
/sign-in	Pantalla de inicio de sesión
/sign-up	Pantalla de registro de usuario
/error/:code	Pantalla de manejo de errores

En la figura 3.10 se presenta la estructura de directorios. La descripción de los contenidos de las carpetas y archivos relevantes es:

- src/app/components: contiene los componentes del proyecto.
- src/app/guards: contiene un *guard* para autenticar las rutas.
- src/app/interceptors: contiene un *interceptor* que añade el JWT del usuario al *header* de la *request* y gestiona los errores de las *requests* HTTP.
- src/app/models: contiene las interfaces para tipificar los datos del sistema.
- src/app/services: contiene los servicios del proyecto.
- src/app/utils: contiene funciones que se reutilizan en todo el proyecto.
- src/app/app.component.ts: contiene el componente con el que se inicializa la aplicación.
- src/app/routes.ts: contiene las rutas de la aplicación.
- src/assets: contiene los *assets* del *frontend*.
- src/manifest.webmanifest: contiene el archivo de manifiesto del proyecto.
- src/robots.txt: contiene la configuración para los *crawlers* [122].
- angular.json: contiene la configuración de Angular.
- bs-config.js: contiene la configuración de lite-server.
- karma.config: contiene la configuración de Jasmine y Karma [123] para las pruebas.
- ngsw-config: contiene la configuración del *service worker* de la PWA.
- ts.config.json: contiene la configuración de TypeScript.

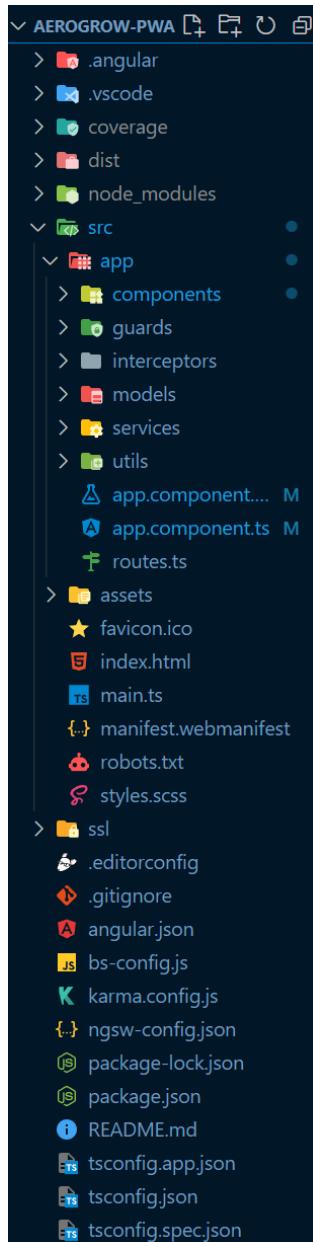


FIGURA 3.10. Estructura de directorios.

3.4.1. Principales pantallas de la aplicación

En la figura 3.11 se visualiza la pantalla de creación de usuarios. Para dar de alta un nuevo usuario se debe ingresar de forma mandatoria su *email* y contraseña, de forma opcional se puede agregar el nombre y apellido. Las cuentas tienen un lapso de 10 días para confirmarse mediante *email* antes de ser eliminadas.



FIGURA 3.11. Pantalla de registro de usuario.

En la figura 3.12 se visualiza la pantalla del listado de zonas de un usuario. Desde esta pantalla, puede realizar las siguientes acciones:

- Acceder a una pantalla para crear una nueva zona.
- Acceder a una pantalla para editar una zona.
- Acceder a una pantalla *dashboard* de una zona.
- Eliminar una zona.
- Monitorear en tiempo real todas las zonas listadas.
- Exportar los datos del listado en formato xlsx.
- Copiar las credenciales de una zona.

El listado de zonas se puede filtrar y ordenar por cualquiera de los campos de la tabla.

Zones										Export	Create new zone	
Filter												
Name	Device	Temperature room	Humidity room	Light level	Temperature fluid	CO2 level	Flow rate	Total litres	C			
Zone	Device	29 C	11 %	263 lm	36 C	46 PPM	7 ml/s	1712 ml	⋮			
										 Edit	 Delete	 Copy credentials
										 Dashboard		

FIGURA 3.12. Pantalla de listado de zonas.

En la figura 3.13 se visualiza la pantalla de creación de zonas. Para crear una nueva zona se debe ingresar un nombre y también el nombre y contraseña del

dispositivo. De manera opcional, se puede ingresar la configuración para las notificaciones, los *relays* y alarmas asociadas al dispositivo y las acciones automatizadas a realizar en caso de que la alarma se dispare. El mismo componente es utilizado para la edición de zonas.

FIGURA 3.13. Pantalla de creación de una zona.

En la figura 3.14 se visualiza un ejemplo de *email* de notificación que envía el sistema. Este tipo de notificación se genera cuando una medición activa una alarma y las notificaciones de *email* están habilitadas en la zona de cultivo.

Hi test@gmail.com

The alarm configured for device ab of zone ab is reporting the following values:

Temperature room: 41 (exceeded limit of 2)

Humidity room: 6 (exceeded limit of 2)

Click on the button below to see your notifications

Notifications

If you need assistance or want to contact us, send us an email to test@gmail.com

FIGURA 3.14. Ejemplo de *email* de notificación de alarma.

En la figura 3.15 se visualiza un ejemplo de notificación *push* que envía el sistema. Este tipo de notificación se genera cuando una medición activa una alarma y las notificaciones de escritorio están habilitadas en la zona de cultivo.

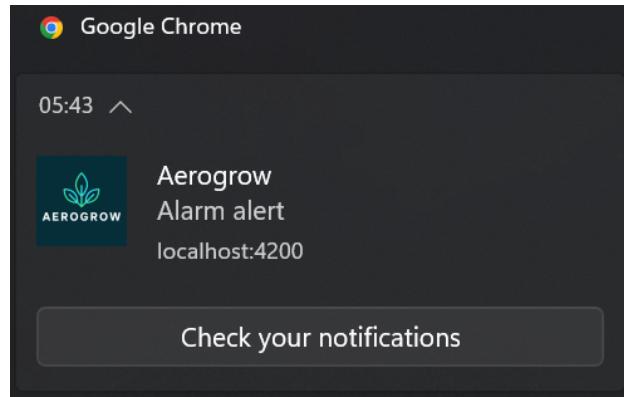


FIGURA 3.15. Ejemplo de notificación *push*.

En la figura 3.16 se visualiza la pantalla del listado de notificaciones de un usuario. El listado se actualiza en tiempo real y se puede filtrar por período de tiempo y por cualquier campo de la tabla. Además, el usuario puede exportar los datos en formato xlsx y marcar las notificaciones como leídas.

Notifications					
Filter Enter a date range 15/5/2023 – 25/5/2023					
Observation	Zone	Device	Active	Date read	Date created
Temperature fluid: 18 (exceeded limit of 15) Light level: 3279 (exceeded limit of 1500)	Zone	Device	true	<button>Mark as read</button>	25/05/2023 22:39
Temperature room: 45 (exceeded limit of 10) Temperature fluid: 41 (exceeded limit of 15) Light level: 2219 (exceeded limit of 1500) Total litres: 3004 (exceeded limit of 2500)	Zone	Device	true	<button>Mark as read</button>	25/05/2023 22:37
Temperature room: 35 (exceeded limit of 10) Temperature fluid: 38 (exceeded limit of 15) Light level: 4227 (exceeded limit of 1500) Total litres: 2868 (exceeded limit of 2500)	Zone	Device	true	<button>Mark as read</button>	25/05/2023 22:36

FIGURA 3.16. Pantalla de listado de notificaciones.

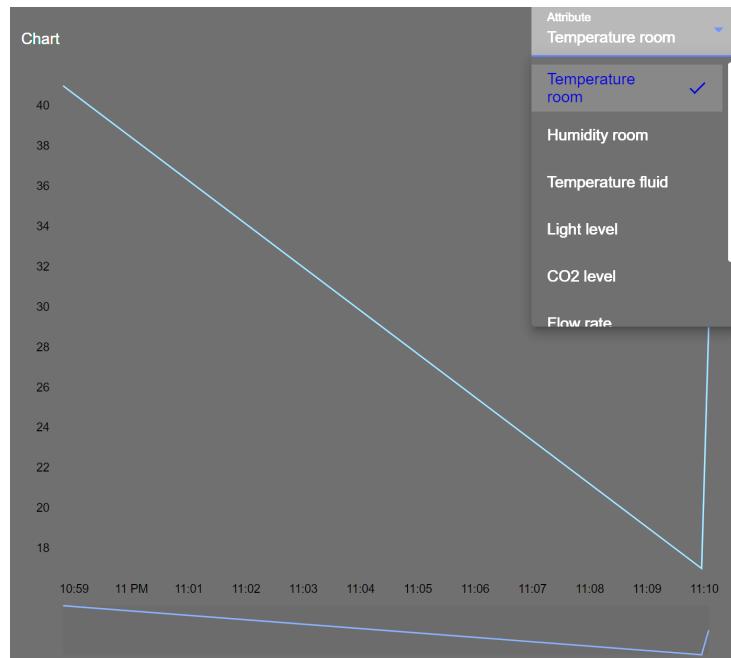
En las figuras 3.17, 3.18 y 3.19 se presentan los componentes que forman la pantalla de *dashboard* de una zona.

En la figura 3.17 se visualiza el listado de mediciones de una zona. El listado se actualiza en tiempo real y se puede filtrar por período de tiempo y por cualquier campo de la tabla. Además, el usuario puede exportar los datos en formato xlsx.

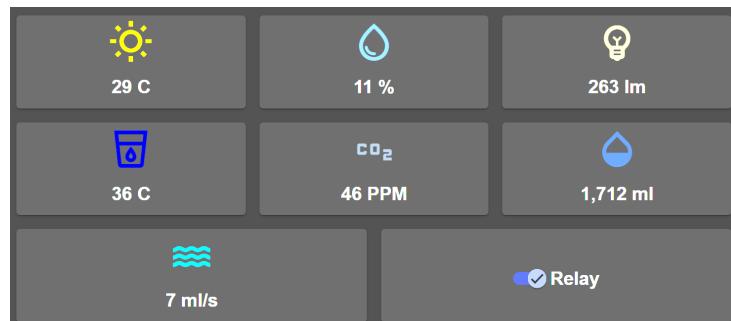
Measurements							
<input type="button" value="Simulate"/> <input type="button" value="Export"/> Enter a date range 14/5/2023 – 24/5/2023 <input type="button" value="Print"/>							
Filter							
Date	Temperature room	Humidity room	Light level	Temperature fluid	CO2 level	Flow rate	Total litres
24/05/2023 23:10	29 C	11 %	263 lm	36 C	46 PPM	7 ml/s	1712 ml
24/05/2023 23:09	17 C	0 %	233 lm	11 C	186 PPM	17 ml/s	1709 ml
24/05/2023 22:58	41 C	15 %	83 lm	11 C	679 PPM	17 ml/s	712 ml

FIGURA 3.17. Listado de mediciones en vista de *dashboard*.

En la figura 3.18 se visualiza el gráfico de las mediciones recibidas en el período de tiempo seleccionado. El componente permite seleccionar el tipo de atributo que se quiere graficar y se actualiza en tiempo real.

FIGURA 3.18. Gráfico de mediciones en vista de *dashboard*.

En la figura 3.19 se presentan las *cards* del *dashboard*. Las *cards* se actualizan en tiempo real y permiten monitorear los datos de la última medición y accionar los *relays* de la zona.

FIGURA 3.19. Cards en vista de *dashboard*.

3.5. Desarrollo del *broker*

El *broker* fue desarrollado en Node.js mediante TypeScript y Aedes.

En el código 3.6 se presenta la inicialización del *broker* con certificados TLS. En las líneas 1 a 3 se obtienen los certificados. En las líneas 5 a 11 se configuran las opciones del *broker*: se asignan los certificados, se solicita a los clientes que tengan certificados válidos y se rechazan las conexiones que no tengan una autoridad de certificación [124] válida. En la línea 12 se establece el número de puerto que utiliza el *broker*. En las líneas 13 a 19 se crea el *broker* con las opciones configuradas.

```

1 const ca = readFileSync("ssl/ca.crt");
2 const privateKey = readFileSync("./ssl/broker.key");
3 const certificate = readFileSync("./ssl/broker.crt");
4
5 const options: TlsOptions = {
6   key: privateKey,
7   cert: certificate,
8   ca: ca,
9   requestCert: true,
10  rejectUnauthorized: true,
11 };
12 const mqttPort = 8883;
13 const mqttServer = createServer(options, aedes.handle);
14
15 mqttServer.listen(mqttPort, () => {
16   console.log(
17     '[server]: MQTT Server is running at mqts://localhost:${mqttPort}'
18   );
19 });

```

CÓDIGO 3.6. Inicialización del *broker* con TLS.

El *broker* utiliza como medida extra de seguridad un proceso de autenticación mediante usuario y contraseña. En el código 3.7 se presenta el proceso de autenticación de los clientes. En las líneas 7 a 12 se verifica que el usuario y contraseña del cliente sea válido. En la línea 13 se autentica al cliente. En las líneas 16 a 19 se entrega al cliente un error que indica que el usuario o la contraseña son inválidos.

```

1 aedes.authenticate = (
2   _client: Client,
3   username: Readonly<string | undefined>,
4   password: Readonly<Buffer | undefined>,
5   done: (error: AuthenticateError | null, success: boolean | null) =>
6     void
7 ) => {
8   if (
9     username &&
10    password &&
11    username === mqttUsername &&
12    Buffer.from(password).toString() === mqttPassword
13  ) {
14    return done(null, true);
15  }
16  const error = new Error() as AuthenticateError;
17  error.returnValue = AuthErrorCode.BAD_USERNAME_OR_PASSWORD;
18
19  return done(error, false);
20 };

```

CÓDIGO 3.7. Autenticación de clientes.

En el código 3.8 se presenta el proceso de validación de mensajes publicados en los tópicos. En las líneas 8 a 11 se verifica que el tópico del mensaje sea válido. En las líneas 12 a 13 se le entrega al cliente un error que indica que no está autorizado para publicar el mensaje. En las líneas 18 a 21 se obtiene el *payload* del mensaje para crear una nueva medición. En las líneas 23 a 28 se le entrega al cliente un error si la medición no pudo ser creada. En la línea 32 se indica que el mensaje publicado es válido.

```

1 aedes.authorizePublish = async (
2   client: Client | null,
3   packet: PublishPacket,
4   done: (error?: AuthenticateError | null | undefined) => void
5 ) => {
6   const error = new Error() as AuthenticateError;
7
8   if (
9     packet.topic != topicMeasurements &&
10    !packet.topic.includes(topicRelays)
11   ) {
12     error.returnValue = AuthErrorCode.NOT_AUTHORIZED;
13     return done(error);
14   }
15
16   if (packet.topic == topicMeasurements) {
17     try {
18       const measurementPayload: MeasurementPayload = JSON.parse(
19         packet.payload.toString()
20       );
21       await createMeasurement(measurementPayload, socket);
22     } catch (errorPayload) {
23       console.error(
24         "[server]: Publish measurement payload has failed",
25         error
26       );
27       error.returnValue = AuthErrorCode.SERVER_UNAVAILABLE;
28       return done(error);
29     }
30   }
31
32   return done(null);
33 };

```

CÓDIGO 3.8. Validación de mensajes.

3.6. Desarrollo sobre el microcontrolador

El *software* del microcontrolador fue desarrollado mediante la plataforma Expressif 32 [125] y el *framework* Arduino [126]. La elección del *framework* es debida a los siguientes factores:

- Facilidad de uso.
- Amplia comunidad y soporte.
- Compatibilidad con una gran cantidad de placas y microcontroladores.
- Desarrollo rápido.

En el código 3.9 se presentan los procesos de conexión al Wi-Fi y configuración de los certificados TLS. En las líneas 1, 2 y 3 se importan las bibliotecas necesarias y

las variables de entorno. En las líneas 5 a 7 se crean las variables para el cliente de la biblioteca y el nombre y contraseña de la red. En la línea 15 se inicia la conexión con las credenciales de la red. En las líneas 22 a 25 se monta el sistema de archivos del microcontrolador. En las líneas 27, 28 y 29 se aplican los certificados TLS al cliente Wi-Fi.

```

1 #include "WiFiClientSecure.h"
2 #include "SPIFFS.h"
3 #include "secrets.h"
4
5 WiFiClientSecure wifiClient;
6 const char* ssid = SSID;
7 const char* password = PASSWORD;
8
9 void setupWifi() {
10     delay(10);
11     Serial.println();
12     Serial.print("Connecting to ");
13     Serial.println(ssid);
14
15     WiFi.begin(ssid, password);
16
17     while (WiFi.status() != WL_CONNECTED) {
18         delay(500);
19         Serial.print(".");
20     }
21
22     if (!SPIFFS.begin(true)){
23         Serial.println("An Error has occurred while mounting SPIFFS");
24         return;
25     }
26
27     wifiClient.setCACert(getFileAsString("/ca.crt"));
28     wifiClient.setCertificate(getFileAsString("/esp.crt"));
29     wifiClient.setPrivateKey(getFileAsString("/esp.key"));
30
31     Serial.println("");
32     Serial.println("WiFi connected");
33     Serial.println("IP address: ");
34     Serial.println(WiFi.localIP());
35 }
```

CÓDIGO 3.9. Conexión Wi-Fi y configuración de los certificados TLS

En el código 3.10 se presenta el proceso de conexión al *broker* MQTT. En las líneas 1, 2 y 3 se importan las bibliotecas necesarias y las variables de entorno. En las líneas 5 y 6 se crean las variables para utilizar las dependencias importadas. En las líneas 8 a 11 se crea un tipo de estructura para agrupar a los *relays*. En las líneas 13 a 17 se crean las variables para la dirección, credenciales y tópicos del *broker*. En las líneas 18 a 20 se crea una variable utilizando la estructura de *relays*. En la línea 23 se establece la dirección y el puerto del *broker*. En la línea 24 se establece el método de *callback*. En las líneas 28 a 44 se intenta realizar la conexión mientras no esté activa. En las líneas 48 a 51 se ejecuta el método para conectarse al *broker* si la conexión no está activa.

```

1 #include "WiFiClientSecure.h"
2 #include <PubSubClient.h>
3 #include "secrets.h"
4
5 WiFiClientSecure wifiClient;
```

```
6 PubSubClient mqttClient(wifiClient);
7
8 typedef struct {
9     int pin;
10    char* id;
11 } Relay;
12
13 const char* mqttServer = BROKER_IP;
14 const char* clientUsername = BROKER_USERNAME;
15 const char* clientPassword = BROKER_PASSWORD;
16 const char* deviceId = DEVICE_ID;
17 const char* topicRelays = "relays/";
18 Relay relays[NUM_RELAYS] = {
19     { 16, RELAY_ID }
20 };
21
22 void setupMQTT() {
23     mqttClient.setServer(mqttServer, 8883);
24     mqttClient.setCallback(callback);
25 }
26
27 void reconnect() {
28     while (!mqttClient.connected()) {
29         Serial.print("Attempting MQTT connection ...");
30         if (mqttClient.connect(deviceId, clientUsername, clientPassword)) {
31             Serial.println("connected");
32             for (int i = 1; i <= NUM_RELAYS; i++) {
33                 char buffer[64];
34                 strcpy(buffer, topicRelays);
35                 strcat(buffer, relays[i-1].id);
36                 mqttClient.subscribe(buffer);
37             }
38         } else {
39             Serial.print("failed, rc=");
40             Serial.print(mqttClient.state());
41             Serial.println(" try again in 5 seconds");
42             delay(5000);
43         }
44     }
45 }
46
47 void loop() {
48     if (!mqttClient.connected()) {
49         reconnect();
50     }
51     mqttClient.loop();
52     ...
53 }
```

CÓDIGO 3.10. Conexión al broker MQTT

Capítulo 4

Ensayos y resultados

En este capítulo se detallan las pruebas unitarias realizadas al *frontend* y *backend* y la prueba de integración del sistema completo.

4.1. Pruebas del *frontend*

Para probar el *frontend* se decidió utilizar una batería de pruebas unitarias [127] creadas con Jasmine y Karma. Estás fueron completamente automatizadas para garantizar y asegurar la calidad de código. Se realizaron 208 pruebas unitarias para lograr un 100 % de cobertura de código.

En la figura 4.1 se visualiza el proceso de ejecución de las pruebas unitarias con Karma.

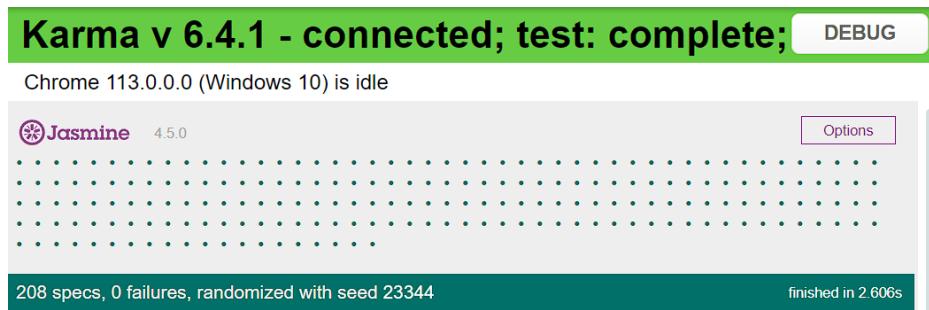


FIGURA 4.1. Pruebas unitarias.

En la figura 4.2 se visualiza la cobertura de código del *frontend*.

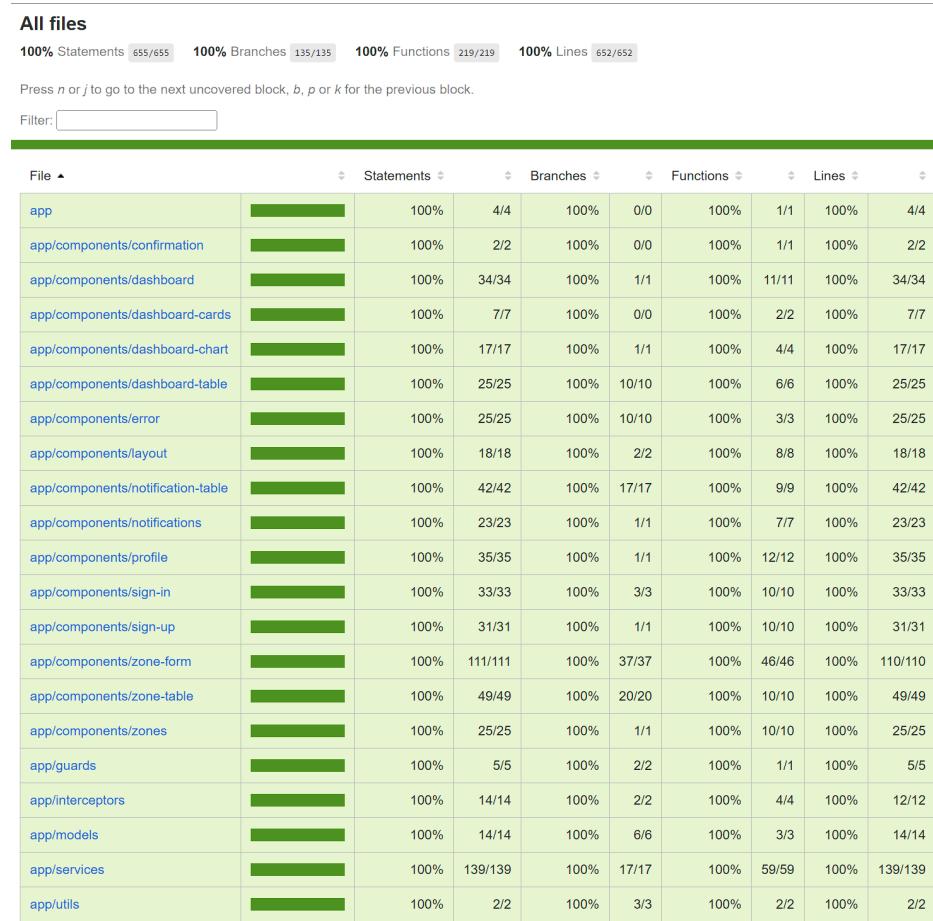


FIGURA 4.2. Cobertura de código.

A modo de ejemplo se presenta en el código 4.1 la prueba unitaria del *guard* utilizado para la autenticación de rutas. En la línea 2 se crea un *mock* del *Router* de Angular. En las líneas 4 a 6 se reinicia el método del *mock* creado. En las líneas 8 a 19 se crea el contexto de prueba. En las líneas 21 y 29 se crean los dos casos de prueba. En las líneas 22 a 25 se crea un *mock* del servicio de autenticación para simular que el usuario tiene una sesión activa. En la línea 26 se verifica que el método del *mock* del *Router* fue ejecutado. En las líneas 30 a 33 se crea un *mock* del servicio de autenticación del *frontend* para simular que el usuario no tiene una sesión activa. En la línea 34 se verifica que el método del *mock* del *Router* no fue ejecutado.

```

1 describe('authGuard', () => {
2   const mockRouter = jasmine.createSpyObj<Router>(['navigate']);
3
4   afterEach(() => {
5     mockRouter.navigate.calls.reset();
6   });
7
8   const setup = (authService: unknown) => {
9     TestBed.configureTestingModule({
10       providers: [
11         { provide: AuthService, useValue: authService },
12         { provide: Router, useValue: mockRouter },
13       ],
14     });
15   };

```

```

16     return TestBed.runInInjectionContext(() =>
17       authGuard({} as ActivatedRouteSnapshot, {} as RouterStateSnapshot)
18     );
19   };
20
21 it('should allow to continue', () => {
22   const mockAuthService: unknown = {
23     isLoggedIn: () => true,
24   };
25   setup(mockAuthService);
26   expect(mockRouter.navigate).not.toHaveBeenCalled();
27 });
28
29 it('should redirect to /sign-in path', () => {
30   const mockAuthService: unknown = {
31     isLoggedIn: () => false,
32   };
33   setup(mockAuthService);
34   expect(mockRouter.navigate).toHaveBeenCalledWith(['/sign-in']);
35 });
36 });

```

CÓDIGO 4.1. Prueba unitaria del *guard* de autenticación de rutas.

En la figura 4.3 se presenta el resultado de configurar los certificados TLS.

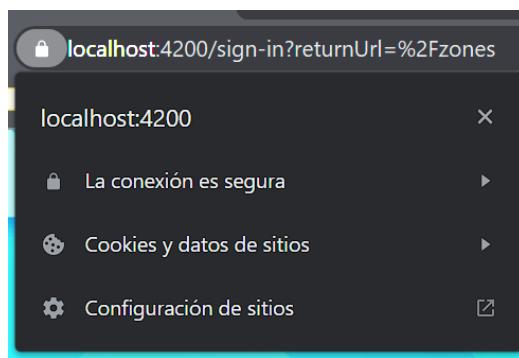


FIGURA 4.3. Certificados TLS aplicados.

Se utilizó Lighthouse para realizar una auditoría del *frontend*, los aspectos evaluados son: rendimiento, accesibilidad, mejores prácticas, SEO y un chequeo del tipo de aplicación: si es PWA se evalúa su implementación, caso contrario se restan puntos al resultado de la auditoría. La herramienta fue aplicada a las tres pantallas más demandantes del sistema: *dashboard* de una zona, listado de zonas y listado de notificaciones.

En la figura 4.4 se presenta el resultado de la auditoría realizada sobre la pantalla de *dashboard* de una zona.

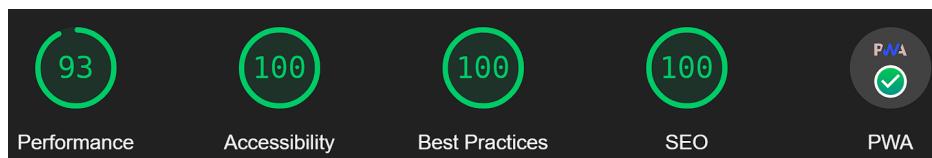


FIGURA 4.4. Auditoría realizada sobre la pantalla de *dashboard* de una zona

En la figura 4.6 se presenta el resultado de la auditoría realizada sobre la pantalla de listado de zonas.

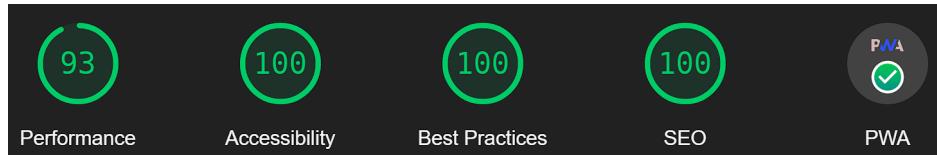


FIGURA 4.5. Auditoría realizada sobre la pantalla de listado de zonas

En la figura 4.6 se presenta el resultado de la auditoría realizada sobre la pantalla de listado de notificaciones.

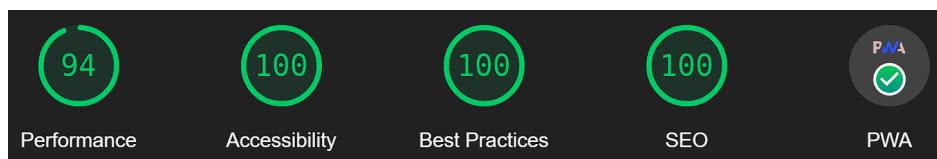


FIGURA 4.6. Auditoría realizada sobre la pantalla de listado de notificaciones

Las tres auditorías realizadas obtuvieron la puntuación máxima en todas las categorías, excepto en rendimiento. Para alcanzar la máxima puntuación en rendimiento, es necesario reducir o eliminar el código JavaScript que no se utiliza en la aplicación.

4.2. Pruebas del *backend*

Para probar el *backend* se decidió utilizar una batería de pruebas unitarias creadas con Jest. Estás fueron completamente automatizadas para garantizar y asegurar la calidad de código. Se realizaron 78 pruebas unitarias para lograr un 100 % de cobertura de código.

En las figuras 4.7 y 4.8 se visualiza el proceso de ejecución de las pruebas unitarias con Jest.

```
PASS routes/zone.route.spec.ts (6.582 s)
zoneRouter
  POST /zones
    ✓ should return a 401 error if there is not authorization token in the header (11 ms)
    ✓ should return 400 if device password is not provided (15 ms)
    ✓ should create a new zone and return it (257 ms)
    ✓ should return 500 if there is an error creating the zone (67 ms)
  GET /zones
    ✓ should return a 401 error if there is not authorization token in the header (7 ms)
    ✓ should return an empty array if the user has no zones (44 ms)
    ✓ should return an array with the zones of the user (274 ms)
    ✓ should return 500 if there is an error retrieving the zones (8 ms)
  GET /zones/:id
    ✓ should return a 401 error if there is not authorization token in the header (6 ms)
    ✓ should return a 404 error if the id is invalid (7 ms)
    ✓ should return a 404 error if the zone does not exists (45 ms)
    ✓ should return a zone if it exists (275 ms)
    ✓ should return 500 if there is an error retrieving the zone (8 ms)
  PATCH /zones/:id
    ✓ should return a 401 error if there is not authorization token in the header (7 ms)
    ✓ should return a 404 error if the id is invalid (7 ms)
    ✓ should return a 404 error if the zone does not exists (41 ms)
    ✓ should patch a zone and return it (314 ms)
    ✓ should patch a zone and return it (369 ms)
    ✓ should return 500 if there is an error patching the zone (268 ms)
  PATCH /zones/relays/:id
    ✓ should return a 401 error if there is not authorization token in the header (6 ms)
    ✓ should return a 404 error if the id is invalid (7 ms)
    ✓ should return a 404 error if the zone does not exists (44 ms)
    ✓ should patch a relay and return it (275 ms)
    ✓ should return 500 if there is an error patching the relay (230 ms)
  DELETE /zones/:id
    ✓ should return a 401 error if there is not authorization token in the header (7 ms)
    ✓ should return a 404 error if the id is invalid (8 ms)
    ✓ should delete a zone and return it (228 ms)
    ✓ should return 500 if there is an error deleting the zone (7 ms)

PASS routes/auth.route.spec.ts
authRouter
  POST /auth/sign-up
    ✓ should create a new user (788 ms)
    ✓ should return 400 if user email is already connected to an account (41 ms)
    ✓ should return 500 if there is an error creating the user (200 ms)
  GET /auth/verifyToken
    ✓ should return a 404 error if the user does not exists (44 ms)
    ✓ should verify the token of the user (318 ms)
    ✓ should return 500 if there is an error retrieving the profile (6 ms)
  GET /auth/profile
    ✓ should return a 401 error if there is not authorization token in the header (8 ms)
    ✓ should return a 404 error if the profile does not exists (399 ms)
    ✓ should return a profile if it exists (42 ms)
    ✓ should return 500 if there is an error retrieving the profile (8 ms)
  PATCH /auth/profile
    ✓ should return a 401 error if there is not authorization token in the header (7 ms)
    ✓ should return a 404 error if the profile does not exists (394 ms)
    ✓ should patch a profile (77 ms)
    ✓ should patch a profile (497 ms)
    ✓ should return 500 if there is an error patching the profile (7 ms)
```

FIGURA 4.7. Pruebas unitarias parte 1 de 2.

```

PASS routes/notification.route.spec.ts
notificationRouter
  PATCH /notifications
    ✓ should return a 401 error if there is not authorization token in the header (9 ms)
    ✓ should return an empty array if there are no notifications (47 ms)
    ✓ should return an array with the notifications (368 ms)
    ✓ should return 400 if the start date is incorrect (278 ms)
    ✓ should return 400 if the start date is incorrect (282 ms)
    ✓ should return 500 if there is an error retrieving the notifications (8 ms)
  PATCH /notifications/markAsRead/:id
    ✓ should return a 401 error if there is not authorization token in the header (6 ms)
    ✓ should return a 404 error if the id is not valid (7 ms)
    ✓ should return a 404 error if the measurement does not exists (44 ms)
    ✓ should patch a notification (378 ms)
    ✓ should return 500 if there is an error patching the notification (7 ms)
  PATCH /notifications/subscribe
    ✓ should return a 401 error if there is not authorization token in the header (6 ms)
    ✓ should return a 404 error if the user does not exists (422 ms)
    ✓ should patch a notification (321 ms)
    ✓ should return 500 if there is an error patching the notification (7 ms)

PASS routes/measurement.route.spec.ts
measurementRouter
  POST measurements/zones/:id
    ✓ should return a 401 error if there is not authorization token in the header (14 ms)
    ✓ should return a 404 error if the id is not valid (10 ms)
    ✓ should return a 404 error if the zone does not exists (46 ms)
    ✓ should create a new measurement for the zone (283 ms)
    ✓ should return 500 if there is an error creating the measurement (8 ms)
  GET measurements/zones/:id
    ✓ should return a 401 error if there is not authorization token in the header (7 ms)
    ✓ should return a 404 error if the id is not valid (8 ms)
    ✓ should return a 404 error if the zone does not exists (43 ms)
    ✓ should return an empty array if the zone has no measurements (307 ms)
    ✓ should return an array with the measurements of the zone (358 ms)
    ✓ should return 400 if the start date is incorrect (328 ms)
    ✓ should return 400 if the end date is incorrect (316 ms)
    ✓ should return 500 if there is an error retrieving the measurements of a zone (7 ms)

Test Suites: 4 passed, 4 total
Tests:      78 passed, 78 total
Snapshots:  0 total
Time:       16.902 s, estimated 18 s

```

FIGURA 4.8. Pruebas unitarias parte 2 de 2.

En la figura 4.9 se visualiza la cobertura de código del *backend*.

File	%Stmts	%Branch	%Funcs	%Lines	Uncovered Line #s
All files	100	100	100	100	
auth.route.ts	100	100	100	100	
measurement.route.ts	100	100	100	100	
notification.route.ts	100	100	100	100	
zone.route.ts	100	100	100	100	

```

Test Suites: 4 passed, 4 total
Tests:      78 passed, 78 total
Snapshots:  0 total

```

FIGURA 4.9. Cobertura de código.

A modo de ejemplo, se presenta en el código 4.2 la prueba unitaria del *endpoint /zones*. En las líneas 2, 9, 19 y 31 se crean los casos de prueba. En las líneas 3 a 6 se simula una *request* al *endpoint* sin haber configurado el *authorization header*, además se valida que el código de la respuesta sea 401. En las líneas 10 a 16 se simula una *request* al *endpoint*, además se valida que el código de la respuesta sea 200 y que el *body* sea una lista vacía. En la línea 20 se crea una zona. En las líneas 21 a 25 se simula una *request* al *endpoint* y se valida que el código de respuesta sea 200. En la línea 27 se elimina la zona creada. En la línea 28 se valida que el *body* de

la respuesta sea una lista con la zona creada. En las líneas 32 a 34 se crea un *mock* sobre un método del ORM Mongoose para simular un error cuando se ejecute. En las líneas 36 a 40 se simula una *request* al *endpoint* y se valida que el código de respuesta sea 500.

```

1 describe("GET /zones", () => {
2   it("should return a 401 error if there is not authorization token in
      the header", async () => {
3     const response = await request(app)
4       .get("/api/zones")
5       .trustLocalhost()
6       .expect(401);
7   });
8
9   it("should return an empty array if the user has no zones", async () => {
10    const response = await request(app)
11      .get("/api/zones")
12      .trustLocalhost()
13      .set("Authorization", 'Bearer ${token}')
14      .expect(200);
15
16    expect(response.body).toEqual([]);
17  });
18
19  it("should return an array with the zones of the user", async () => {
20    await createZone();
21    const response = await request(app)
22      .get("/api/zones")
23      .trustLocalhost()
24      .set("Authorization", 'Bearer ${token}')
25      .expect(200);
26
27    await deleteZone();
28    expect(response.body).toEqual([zone]);
29  });
30
31  it("should return 500 if there is an error retrieving the zones",
32    async () => {
33    jest.spyOn(Zone, "find").mockImplementation(() => {
34      throw new Error();
35    });
36
37    const response = await request(app)
38      .get("/api/zones")
39      .trustLocalhost()
40      .set("Authorization", 'Bearer ${token}')
41      .expect(500);
42
43    jest.spyOn(Zone, "find").mockRestore();
44    expect(response.statusCode).toEqual(500);
45  });
46});
```

CÓDIGO 4.2. Prueba unitaria del *endpoint* /zones

4.3. Prueba final de integración

Se realizó una prueba final de integración para verificar el correcto funcionamiento del sistema.

En la figura 4.10 se visualiza un diagrama del banco de pruebas con los componentes y relaciones que lo forman.

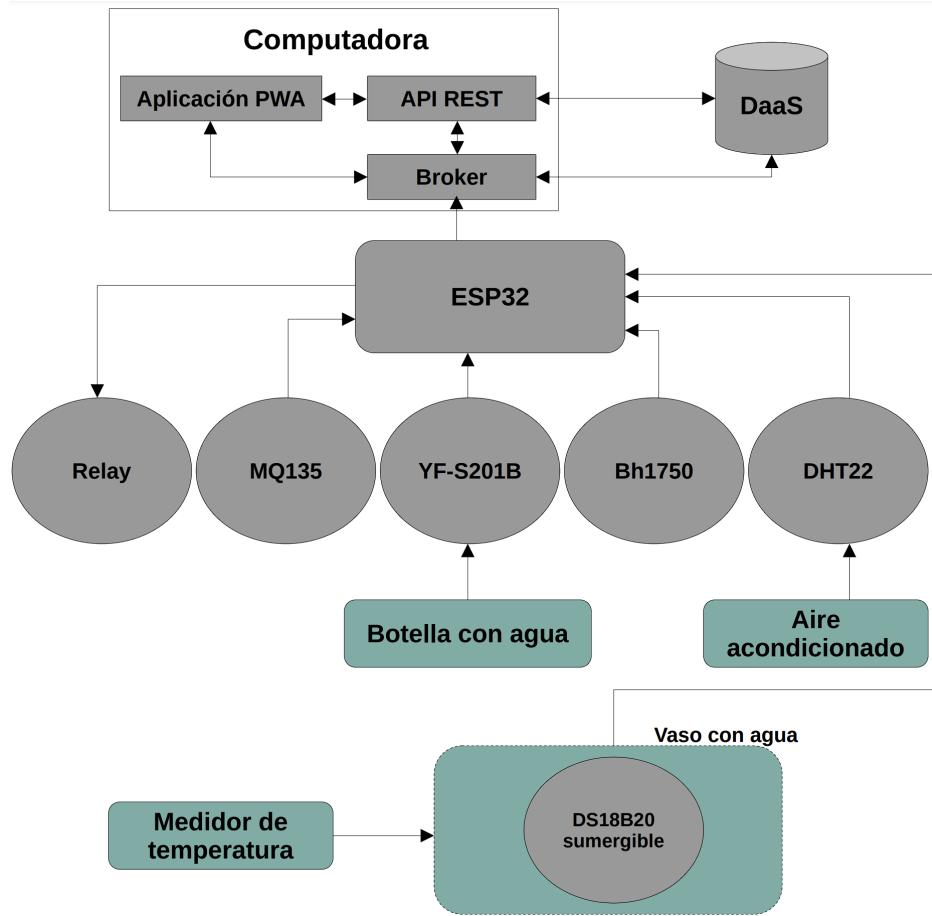


FIGURA 4.10. Diagrama del banco de pruebas.

Los pasos previos de configuración de la prueba consistieron en:

- Inicializar el *backend* y el *broker* por medio del comando `npm run dev`. En la figura 4.11 se visualiza el resultado de la ejecución del comando.

```
05:03:02 - Starting compilation in watch mode...
[0]
[1] ⚡[server]: MongoDB connected successfully
[0]
[0] 05:03:03 - Found 0 errors. Watching for file changes.
[1] ⚡[server]: Server is running at https://localhost:8000
[1] ⚡[server]: MQTT Server is running at mqtt://localhost:8883
[1] ⚡[server]: MongoDB connected successfully
```

FIGURA 4.11. Inicialización del *backend*.

- Inicializar el *frontend* por medio del comando `npm run lite-server`. En la figura 4.12 se visualiza el resultado de la ejecución del comando.

```
[Browsersync] Access URLs:
-----
  Local: https://localhost:4200
  External:
-----
  UI: http://localhost:3001
  UI External: http://localhost:3001
-----
[Browsersync] Serving files from: dist/aerogrow-pwa
[Browsersync] Watching files...
```

FIGURA 4.12. Inicialización del *frontend*.

- Iniciar sesión en el sistema con una cuenta creada previamente.
- Crear una nueva zona de cultivo que tenga un *relay* y dos alarmas. La primera alarma se dispara si la temperatura del ambiente es diferente a 1 °C y tiene una acción asociada que activa el *relay* si la alarma se dispara por una temperatura mayor a 1 °C. La segunda alarma se dispara si la temperatura del líquido es menor a 1 °C o mayor a 2 °C. En las figuras 4.13 y 4.14 se visualiza el proceso de creación de la zona.

Create new zone

Name*
Zone

Device name*
Device

Device password*

Notifications

Enable desktop
 Enable email

Relay 1

Name*
Relay

Remove relay

Alarm 1

Type* Temperature room Minimum value* 1 Maximum value* 1 Add action Remove alarm

Action 1

Trigger* Maximum value Action* Turn on Relays* Relay Remove action

Alarm 2

Type* Temperature fluid Minimum value* 1 Maximum value* 2 Add action Remove alarm

Add alarm Add relay Create Go to zones

FIGURA 4.13. Pantalla de creación de una zona.

Zones									
Filter									
Name	Device	Temperature room	Humidity room	Light level	Temperature fluid	CO2 level	Flow rate	Total litres	G
Zone	Device								⋮

FIGURA 4.14. Pantalla de listado de zonas.

- Conectar y configurar el *software* del microcontrolador con las credenciales de la zona. En la figura 4.15 se presenta la configuración de credenciales en el microcontrolador.

```
#define DEVICE_ID "6487bc489305b6049152bf47"
#define DEVICE_PASSWORD [REDACTED]
#define RELAY_ID "6487bc489305b6049152bf4a"
#define NUM_RELAYS 1
```

FIGURA 4.15. Configuración de credenciales en el microcontrolador.

- Programar el microcontrolador con las nuevas credenciales.
- Monitorear el microcontrolador. En la figura 4.16 se observa una parte del proceso de monitoreo mediante la extensión PlatformIO.

```
* Executing task: \Scripts\platformio.exe device monitor --environment esp32dev
--- Terminal on COM3 | 115200 8-N-1
--- Available filters and text transformations: colorize, debug, default, direct, esp32_exception_decoder, hexlify, log2file, nocontrol, printable, send_on_enter, time
--- More details at https://bit.ly/pio-monitor-filters
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
...
WiFi connected
IP address:
Attempting MQTT connection...connected
```

FIGURA 4.16. Monitoreo del microcontrolador.

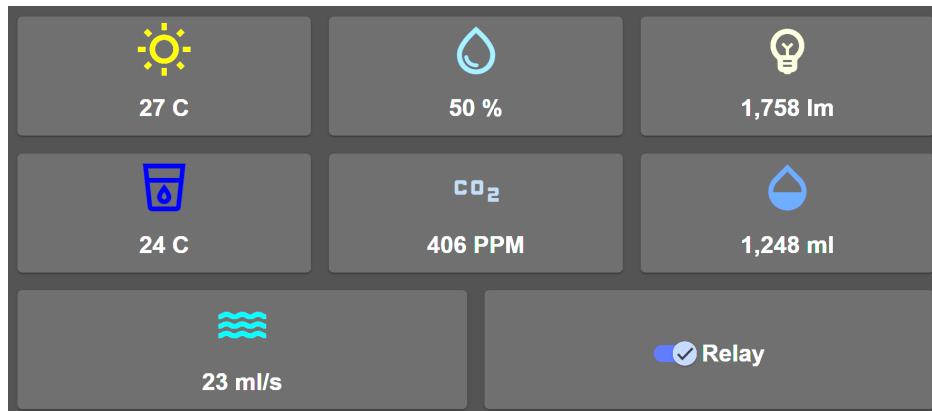
Para asegurar que se active la primera alarma, se colocó el sensor DHT22 en un ambiente con un aire acondicionado [128] encendido en una temperatura de 24 °C por 30 minutos.

Para asegurar que se dispare la segunda alarma, se colocó el sensor DS18B20 en un recipiente con agua previamente validada con un medidor externo [129] para asegurar que su temperatura sea de al menos 20 °C.

Para que todos los sensores reporten valores al microcontrolador, se ingresó agua manualmente al sensor YF-S201B.

En las figuras 4.17 y 4.18 se observa la pantalla de *dashboard* de la zona con la medición registrada y el *relay* activo.

Measurements							
<input type="button" value="Simulate"/> <input type="button" value="Export"/> <input type="text" value="Enter a date range"/> <input type="button" value="16/5/2023 – 26/5/2023"/>							
Filter							
Date	Temperature room	Humidity room	Light level	Temperature fluid	CO2 level	Flow rate	Total litres
26/05/2023 04:27	26.7 C	50.4 %	1758 lm	24.4 C	406 PPM	23 ml/s	1248 ml

FIGURA 4.17. Listado de mediciones en vista de *dashboard*.FIGURA 4.18. *Cards* en vista de *dashboard*.

En las figuras 4.19, 4.20 y 4.21 se presentan las notificaciones enviadas por el sistema al activarse la alarma.

Notifications					
<input type="button" value="Export"/> <input type="text" value="Enter a date range"/> <input type="button" value="16/5/2023 – 26/5/2023"/>					
Filter					
Observation	Zone	Device	Active	Date read	Date created
Temperature room: 26.7 (exceeded limit of 1) Temperature fluid: 24.4 (exceeded limit of 2)	Zone	Device	true	<input type="button" value="Mark as read"/>	26/05/2023 04:27

FIGURA 4.19. Pantalla de listado de notificaciones.

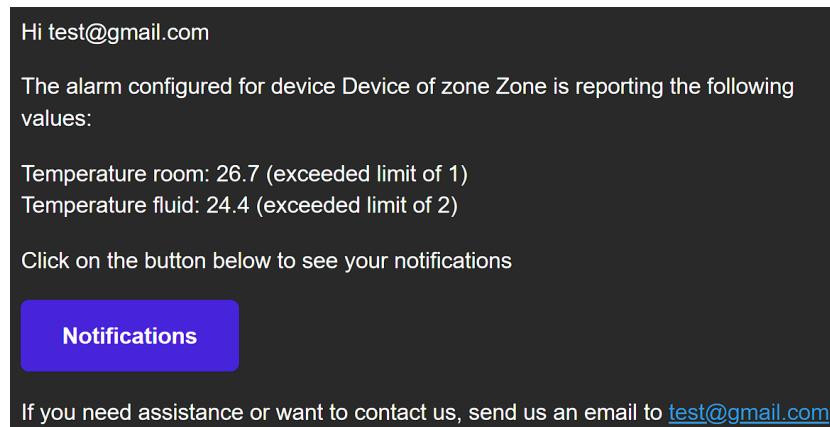
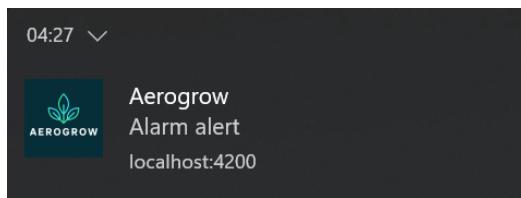


FIGURA 4.20. Email enviado por el sistema.

FIGURA 4.21. Notificación *push* enviada por el sistema.

4.4. Comparación con el estado del arte

En la tabla 4.1 se presenta la comparación de las características y funcionalidades entre los sistemas comerciales encontrados en el mercado nacional e internacional y el trabajo realizado llamado Aerogrow.

TABLA 4.1. Comparativa entre soluciones comerciales similares y el trabajo realizado.

Funcionalidad	Smartcultiva	My Autogrow	Aerogrow
MQTT	Sí	Sí	Sí
Sistema de alertas	Sí	Sí	Sí
Acciones automatizadas	Sí	No	Sí
Notificaciones <i>push</i>	No	No	Sí
Notificaciones <i>email</i>	Sí	Sí	Sí
Accionamiento de dispositivos externos	Sí	No	Sí
Escalabilidad en sensores	Limitado	Limitado	Alta
Tipo de aplicación	Móvil y web	Web	PWA

En la tabla 4.2 se presenta la comparación de las características y funcionalidades entre los sistemas encontrados en publicaciones científicas y el trabajo realizado.

TABLA 4.2. Comparativa entre soluciones similares encontradas en publicaciones científicas y el trabajo realizado.

Funcionalidad	Monitoring Soil (...)	A Smart (...)	Aerogrow
MQTT	Sí	No	Sí
LoRa	Sí	Sí	No
Sistema de alertas	No	Sí	Sí
Acciones automatizadas	No	No	Sí
Notificaciones <i>push</i>	No	No	Sí
Notificaciones <i>email</i>	No	Sí	Sí
Accionamiento de dispositivos externos	No	No	Sí
Escalabilidad en sensores	Media	Media	Alta
Tipo de aplicación	Web	Web	PWA

De acuerdo a la comparación realizada, el trabajo elaborado se destaca por permitir recibir notificaciones *push* y por la escalabilidad para añadir nuevos sensores. Sin embargo, se debe tener en cuenta los siguientes factores:

- Smartcultiva y este trabajo son las únicas soluciones que permiten crear acciones automatizadas cuando se activa una alerta.
- Smartcultiva y este trabajo son las únicas soluciones que permiten el accionamiento de dispositivos externos.
- Las dos publicaciones científicas soportan el uso del protocolo LoRa.

Capítulo 5

Conclusiones

En este capítulo se muestran las conclusiones sobre el trabajo realizado y se presentan algunas mejoras como posible trabajo futuro.

5.1. Resultados obtenidos

En este trabajo se completó el diseño, desarrollo y *testing* de un sistema de gestión de cultivos aeropónicos.

Para evaluar los resultados obtenidos del trabajo es importante destacar los siguientes factores:

- Se cumplió con la planificación en tiempo y forma aunque no se siguió exactamente con el orden de desarrollo planteado para cada tarea.
- No se manifestó ninguno de los riesgos previamente identificados en la planificación.
- Luego de realizar un análisis de los requerimientos, se concluye que todos fueron cumplidos. Sin embargo, como se menciona en la sección 3.1.1 se realizaron dos modificaciones en el trabajo que afectaron algunos requerimientos: se decidió reemplazar la aplicación SSR por una PWA y se realizaron modificaciones en los datos a almacenar en el DaaS.

Fueron de gran utilidad los conocimientos adquiridos a lo largo de la especialización. A continuación, se enumeran las asignaturas que tuvieron mayor relevancia:

- Gestión de proyectos.
- Arquitectura de protocolos.
- Gestión de grandes volúmenes de datos.
- Desarrollo de aplicaciones multiplataforma.
- Ciberseguridad en Internet de las Cosas.

5.2. Trabajo futuro

A continuación se detallan las principales líneas de acción para dar continuidad a este trabajo:

- Mejorar el *responsive* de las tablas del *frontend*.
- Incorporar SSR a la PWA.

- Incorporar sensor de pH de la solución nutritiva.
- Incorporar sensor de TDS de la solución nutritiva.
- Incorporar estadísticas y más gráficos para el *dashboard* de las zonas de cultivo.
- Permitir crear más de un dispositivo por zona de cultivo.
- Permitir seleccionar qué sensores se quiere asociar a cada dispositivo de las zonas de cultivo.
- Desarrollar pruebas unitarias para el *broker*.
- Añadir multidioma al sistema utilizando i18n [130].
- Permitir seleccionar qué unidades se quiere utilizar en las mediciones, por ejemplo: Fahrenheit o Celsius para las temperaturas.
- Calibrar los sensores del sistema.
- Realizar pruebas de campo en un entorno real.
- Integrar el sistema en un gabinete.
- Desplegar el sistema a un entorno *cloud*.

Bibliografía

- [1] Agriculturers. *¿Que es la aeroponía?* Disponible: 2023-05-04. URL: <https://agriculturers.com/que-es-la-aeroponia/>.
- [2] Wikipedia. *Hidroponía*. Disponible: 2023-05-04. URL: <https://es.wikipedia.org/wiki/Hidrop%C3%A3nica>.
- [3] NASA. *Progressive Plant Growing is a Blooming Business*. Disponible: 2023-05-04. URL: https://www.nasa.gov/vision/earth/technologies/aeroponic_plants.html.
- [4] Wikipedia. *Aeroponía*. Disponible: 2023-05-04. URL: <https://es.wikipedia.org/wiki/Aeropon%C3%A3nica>.
- [5] El holandés picante. *Guía de aeroponía para principiantes*. Disponible: 2023-05-04. URL: <https://elholandespiciente.com/guia-de-aeroponía-para-principiantes/>.
- [6] IONOS. *Comparativa del server-side rendering, client-side rendering y los static site generators*. Disponible: 2023-05-04. URL: <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/lenguajes-del-lado-servidor-o-del-cliente-diferencias/>.
- [7] Wikipedia. *REST*. Disponible: 2023-05-04. URL: <https://es.wikipedia.org/wiki/API>.
- [8] Wikipedia. *REST*. Disponible: 2023-05-04. URL: https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional.
- [9] MDN. *HTTP*. Disponible: 2023-05-05. URL: <https://developer.mozilla.org/es/docs/Web/HTTP>.
- [10] AWS. *¿Qué es MQTT?* Disponible: 2023-05-05. URL: <https://aws.amazon.com/es/what-is/mqtt/>.
- [11] Wikipedia. *WebSocket*. Disponible: 2023-05-05. URL: <https://es.wikipedia.org/wiki/WebSocket>.
- [12] MongoDB. *Database as a Service Explained*. Disponible: 2023-05-05. URL: <https://www.mongodb.com/database-as-a-service>.
- [13] MDN. *TLS*. Disponible: 2023-05-04. URL: <https://developer.mozilla.org/en-US/docs/Glossary/TLS>.
- [14] Bluelab. *Autogrow by Bluelab*. Disponible: 2023-05-05. URL: https://bluelab.com/new_zealand/autogrow.
- [15] Marco A. Oltra Cámara. *¿Qué es la Fertirrigación?* Disponible: 2023-05-29. URL: <https://www.fertirrigacion.com/que-es-la-fertirrigacion/>.
- [16] Smartcultiva. *Smartcultiva*. Disponible: 2023-05-06. URL: <https://www.smartcultiva.com/>.
- [17] MDN. *Usando la API de Notificaciones*. Disponible: 2023-05-08. URL: https://developer.mozilla.org/es/docs/Web/API/Notifications_API/Using_the_Notifications_API.
- [18] Pisana Placidi, Renato Morbidelli, Diego Fortunati, Diego Fortunati, Nicola Papini. *Monitoring Soil and Ambient Parameters in the IoT Precision Agriculture Scenario: An Original Modeling Approach Dedicated to Low-Cost Soil Water Content Sensors*. Disponible: 2023-05-08. URL:

- https://www.researchgate.net/publication/353828389_Monitoring_Soil_and_Ambient_Parameters_in_the_IoT_Precision_Agriculture_Scenario_An_Original_Modeling_Approach_Dedicated_to_Low-Cost_Soil_Water_Content_Sensors.
- [19] Tarlogic. ¿Qué es LoRa protocol? Disponible: 2023-05-08. URL: <https://www.tarlogic.com/es/glosario-ciberseguridad/lora-protocol/>.
- [20] Mohamed Saban, Mostapha Bekkour, Ibtisam Amdaouch, Jaouad El Gueri. *A Smart Agricultural System Based on PLC and a Cloud Computing Web Application Using LoRa and LoRaWan*. Disponible: 2023-05-08. URL: https://www.researchgate.net/publication/368978233_A_Smart_Agricultural_System_Based_on_PLC_and_a_Cloud_Computing_Web_Application_Using_LoRa_and_LoRaWan.
- [21] Wikipedia. Bot. Disponible: 2023-05-08. URL: <https://es.wikipedia.org/wiki/Bot>.
- [22] Telegram. ¿Qué es Telegram? ¿Qué puedo hacer aquí? Disponible: 2023-05-08. URL: <https://telegram.org/faq#p-que-es-telegram-que-puedo-hacer-aqui>.
- [23] Wikipedia. CRUD. Disponible: 2023-05-05. URL: <https://es.wikipedia.org/wiki/CRUD>.
- [24] GEYA. *5V Relay Module – How it Works and Application*. Disponible: 2023-05-07. URL: <https://www.geya.net/5v-relay-module-how-it-works-and-application>.
- [25] Wikipedia. Wi-Fi. Disponible: 2023-05-06. URL: <https://es.wikipedia.org/wiki/Wifi>.
- [26] SIMFORM. *Why Test Coverage is an Important part of Software Testing?* Disponible: 2023-05-10. URL: <https://www.simform.com/blog/test-coverage/>.
- [27] Wikipedia. Capa de aplicación. Disponible: 2023-05-04. URL: https://es.wikipedia.org/wiki/Capa_de_aplicaci%C3%B3n.
- [28] MDN. HTML. Disponible: 2023-05-05. URL: <https://developer.mozilla.org/es/docs/Web/HTML>.
- [29] Wikipedia. Cliente-servidor. Disponible: 2023-05-04. URL: <https://es.wikipedia.org/wiki/Cliente-servidor>.
- [30] Wikipedia. Protocolo sin estado. Disponible: 2023-05-04. URL: https://es.wikipedia.org/wiki/Protocolo_sin_estado.
- [31] Wikipedia. Modelo TCP/IP. Disponible: 2023-05-0605. URL: https://es.wikipedia.org/wiki/Modelo_TCP/IP.
- [32] Wikipedia. Capa de transporte. Disponible: 2023-05-04. URL: https://es.wikipedia.org/wiki/Capa_de_transporte.
- [33] Wikipedia. Protocolo seguro de transferencia de hipertexto. Disponible: 2023-05-05. URL: https://es.wikipedia.org/wiki/Protocolo_seguro_de_transferencia_de_hipertexto.
- [34] Wikipedia. Internet de las cosas. Disponible: 2023-05-05. URL: https://es.wikipedia.org/wiki/Internet_de_las_cosas.
- [35] cloudflare. ¿Qué es DNS? | Cómo funciona. Disponible: 2023-05-10. URL: <https://www.cloudflare.com/es-es/learning/dns/what-is-dns/>.
- [36] Wikipedia. Díplex. Disponible: 2023-05-05. URL: [https://es.wikipedia.org/wiki/D%C3%A9plex_\(telecomunicaciones\)#Full-duplex](https://es.wikipedia.org/wiki/D%C3%A9plex_(telecomunicaciones)#Full-duplex).
- [37] Wikipedia. TCP. Disponible: 2023-05-05. URL: https://es.wikipedia.org/wiki/Protocolo_de_control_de_transmisi%C3%B3n.

- [38] Wikipedia. *TSMC*. Disponible: 2023-05-06. URL: <https://es.wikipedia.org/wiki/ESP32>.
- [39] Wikipedia. *Sistema en un chip*. Disponible: 2023-05-05. URL: https://es.wikipedia.org/wiki/Sistema_en_un_chip.
- [40] Wikipedia. *Bluetooth*. Disponible: 2023-05-06. URL: <https://es.wikipedia.org/wiki/Bluetooth>.
- [41] Github. *Arduino Client for MQTT*. Disponible: 2023-05-29. URL: <https://github.com/knolleary/pubsubclient>.
- [42] Github. *Adafruit Unified Sensor Driver*. Disponible: 2023-05-29. URL: https://github.com/adafruit/Adafruit_Sensor.
- [43] Github. *DHT sensor library*. Disponible: 2023-05-29. URL: <https://github.com/adafruit/DHT-sensor-library>.
- [44] Github. *Arduino Json*. Disponible: 2023-05-29. URL: <https://github.com/bblanchon/ArduinoJson>.
- [45] Github. *MQ135 gas sensor*. Disponible: 2023-05-29. URL: <https://registry.platformio.org/libraries/phoenix1747/MQ135>.
- [46] Github. *BH1750*. Disponible: 2023-05-29. URL: <https://github.com/claws/BH1750>.
- [47] Github. *Arduino Library for Maxim Temperature Integrated Circuits*. Disponible: 2023-05-29. URL: <https://github.com/milesburton/Arduino-Temperature-Control-Library>.
- [48] sparkfun. *DHT22*. Disponible: 2023-05-06. URL: <https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf>.
- [49] Electrónica online. *¿Qué es un Termistor y Cómo Funciona?* Disponible: 2023-05-07. URL: <https://electronicaonline.net/componentes-electronicos/resistor/termistor/>.
- [50] sparkfun. *DHT11*. Disponible: 2023-05-06. URL: <https://cdn.sparkfun.com/assets/b/3/f/9/d/OKY3068-1.pdf>.
- [51] mouser. *Bh1750*. Disponible: 2023-05-06. URL: <https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf>.
- [52] Wikipedia. *Conversor de señal analógica a digital*. Disponible: 2023-05-06. URL: https://es.wikipedia.org/wiki/Conversor_de_se%C3%B1al_anal%C3%B3gica_a_digital.
- [53] Electrónicos Caldas. *MQ-135*. Disponible: 2023-05-07. URL: https://www.electronicoscaldas.com/datasheet/MQ-135_Hanwei.pdf.
- [54] Handson Technology. *YF-S201B*. Disponible: 2023-05-07. URL: <https://handsontec.com/dataspecs/sensor/Water%20Flow%20Sensor.pdf>.
- [55] EcuRed. *Efecto Hall*. Disponible: 2023-05-07. URL: https://www.ecured.cu/Efecto_Hall.
- [56] quick-teck. *DS18B20*. Disponible: 2023-05-07. URL: <https://www.quick-teck.co.uk/Management/EEUploadFile/1420644897.pdf>.
- [57] Wikipedia. *1-Wire*. Disponible: 2023-05-07. URL: <https://es.wikipedia.org/wiki/1-Wire>.
- [58] XATAKA. *¿Qué es una Aplicación Web Progresiva o PWA?* Disponible: 2023-05-08. URL: <https://www.xataka.com/basicos/que-es-una-aplicacion-web-progresiva-o-pwa>.
- [59] MDN. *Service Worker API*. Disponible: 2023-05-08. URL: https://developer.mozilla.org/es/docs/Web/API/Service_Worker_API.
- [60] MDN. *Web App Manifest*. Disponible: 2023-05-08. URL: <https://developer.mozilla.org/es/docs/Web/Manifest>.

- [61] MDN. *Diseño receptivo*. Disponible: 2023-05-11. URL: https://developer.mozilla.org/es/docs/Learn/CSS/CSS_layout/Responsive_Design.
- [62] Angular. *Angular*. Disponible: 2023-05-07. URL: <https://angular.io/docs>.
- [63] Wikipedia. *Framework*. Disponible: 2023-05-10. URL: <https://es.wikipedia.org/wiki/Framework>.
- [64] TypeScript. *What is TypeScript?* Disponible: 2023-05-08. URL: <https://www.typescriptlang.org/>.
- [65] IONOS. *Single page application: definición, funcionamiento y utilidad*. Disponible: 2023-05-09. URL: <https://www.ionos.es/digitalguide/paginas-web/creacion-de-paginas-web/single-page-application/>.
- [66] códigofacilito. *MVC (Model, View, Controller) explicado*. Disponible: 2023-05-09. URL: <https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>.
- [67] AWS. *¿Qué es JavaScript?* Disponible: 2023-05-09. URL: <https://aws.amazon.com/es/what-is/javascript/>.
- [68] Github. *Official components for Angular*. Disponible: 2023-05-29. URL: <https://github.com/angular/components>.
- [69] Angular. *Angular Material*. Disponible: 2023-05-07. URL: <https://material.angular.io/>.
- [70] Github. *ngx-charts*. Disponible: 2023-05-29. URL: <https://github.com/swimlane/ngx-charts>.
- [71] D3.js. *D3.js*. Disponible: 2023-05-07. URL: <https://d3js.org/>.
- [72] Github. *socket.io-client*. Disponible: 2023-05-29. URL: <https://github.com/socketio/socket.io-client>.
- [73] Socket.io. *Socket.io*. Disponible: 2023-05-07. URL: <https://socket.io/>.
- [74] Github. *ngx-cookieconsent*. Disponible: 2023-05-29. URL: <https://github.com/tinesoft/ngx-cookieconsent>.
- [75] CLACSO. *¿Qué son las cookies?* Disponible: 2023-05-07. URL: <https://www.clacso.org/que-son-las-cookies/>.
- [76] Github. *ngx-cookieconsent*. Disponible: 2023-05-29. URL: <https://github.com/inorganik/ngx-countUp>.
- [77] Github. *lite-server*. Disponible: 2023-05-29. URL: <https://github.com/johnpapa/lite-server>.
- [78] Github. *compression*. Disponible: 2023-05-29. URL: <https://github.com/expressjs/compression>.
- [79] IONOS. *¿Qué es gzip?* Disponible: 2023-05-07. URL: <https://www.ionos.es/digitalguide/servidores/know-how/que-es-gzip-y-cuales-son-sus-ventajas/>.
- [80] Nodejs. *Acerca de Node.js*. Disponible: 2023-05-07. URL: <https://nodejs.org/es/about>.
- [81] Github. *bcrypt.js*. Disponible: 2023-05-29. URL: <https://github.com/dcodeIO/bcrypt.js/blob/master/README.md>.
- [82] cloudflare. *¿Qué es la encriptación? | Tipos de encriptación*. Disponible: 2023-05-08. URL: <https://www.cloudflare.com/es-es/learning/ssl/what-is-encryption/>.
- [83] GitHub. *Aedes*. Disponible: 2023-05-11. URL: <https://github.com/moscajs/aedes>.
- [84] Github. *cors*. Disponible: 2023-05-29. URL: <https://github.com/expressjs/cors>.

- [85] MDN. *Intercambio de recursos de origen cruzado*. Disponible: 2023-05-08. URL: <https://developer.mozilla.org/es/docs/Web/HTTP/CORS>.
- [86] Github. *dotenv*. Disponible: 2023-05-29. URL: <https://github.com/motdotla/dotenv>.
- [87] Kinsta. *Variables de Entorno: ¿Qué Son y Cómo Usarlas?* Disponible: 2023-05-08. URL: <https://kinsta.com/es/base-de-conocimiento/variables-de-entorno/>.
- [88] Github. *express*. Disponible: 2023-05-29. URL: <https://github.com/expressjs/express>.
- [89] express.js. *Express*. Disponible: 2023-05-08. URL: <https://expressjs.com/es/>.
- [90] Github. *Express Rate Limit*. Disponible: 2023-05-29. URL: <https://github.com/express-rate-limit/express-rate-limit>.
- [91] Github. *jsonwebtoken*. Disponible: 2023-05-29. URL: <https://github.com/auth0/node-jsonwebtoken>.
- [92] jwt. *JSON Web Tokens*. Disponible: 2023-05-08. URL: <https://jwt.io/>.
- [93] Github. *Mongoose*. Disponible: 2023-05-29. URL: <https://github.com/Automattic/mongoose>.
- [94] Wikipedia. *Asignación objeto-relacional*. Disponible: 2023-05-08. URL: https://es.wikipedia.org/wiki/Asignaci%C3%B3n_objeto-relacional.
- [95] mongoose. *mongoose.js*. Disponible: 2023-05-08. URL: <https://mongoosejs.com/>.
- [96] Github. *Node Cron*. Disponible: 2023-05-29. URL: <https://github.com/node-cron/node-cron>.
- [97] Github. *Nodemailer*. Disponible: 2023-05-29. URL: <https://github.com/nodemailer/nodemailer>.
- [98] Github. *Express Handlebars plugin for Nodemailer*. Disponible: 2023-05-29. URL: <https://github.com/yads/nodemailer-express-handlebars>.
- [99] Github. *web-push*. Disponible: 2023-05-29. URL: <https://github.com/web-push-libs/web-push>.
- [100] Github. *TypeScript*. Disponible: 2023-05-29. URL: <https://github.com/microsoft/TypeScript>.
- [101] Github. *nodemon*. Disponible: 2023-05-29. URL: <https://github.com/remy/nodemon>.
- [102] Github. *SuperTest*. Disponible: 2023-05-29. URL: <https://github.com/ladjs/supertest>.
- [103] Github. *Jest*. Disponible: 2023-05-29. URL: <https://github.com/jestjs/jest>.
- [104] Github. *ts-jest*. Disponible: 2023-05-29. URL: <https://github.com/kulshekhar/ts-jest>.
- [105] MongoDB. *¿Qué es MongoDB?* Disponible: 2023-05-07. URL: <https://www.mongodb.com/es/what-is-mongodb>.
- [106] ORACLE. *¿Qué es NoSQL?* Disponible: 2023-05-09. URL: <https://www.oracle.com/ar/database/nosql/what-is-nosql/>.
- [107] Wikipedia. *Base de datos documental*. Disponible: 2023-05-09. URL: https://es.wikipedia.org/wiki/Base_de_datos_documental.
- [108] Oracle. *¿Qué es una base de datos relacional?* Disponible: 2023-05-10. URL: <https://www.oracle.com/ar/database/what-is-a-relational-database/>.
- [109] MongoDB. *JSON and BSON*. Disponible: 2023-05-10. URL: <https://www.mongodb.com/json-and-bson>.
- [110] Jasmine. *Jasmine*. Disponible: 2023-05-07. URL: <https://jasmine.github.io/>.

- [111] IONOS. *El behavior-driven development en el desarrollo ágil de software*. Disponible: 2023-05-09. URL: <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-el-behavior-driven-development/>.
- [112] Jest. *Jest*. Disponible: 2023-05-07. URL: <https://jestjs.io/>.
- [113] Wikipedia. *Copia instantánea de volumen*. Disponible: 2023-05-09. URL: https://es.wikipedia.org/wiki/Copia_instant%C3%A1nea_de_volumen.
- [114] webpack. *webpack concepts*. Disponible: 2023-05-09. URL: <https://webpack.js.org/concepts/>.
- [115] freecodecamp. *¿Qué es Babel?* Disponible: 2023-05-09. URL: <https://www.freecodecamp.org/espanol/news/que-es-babel/>.
- [116] Google. *Lighthouse*. Disponible: 2023-05-07. URL: <https://developer.chrome.com/docs/lighthouse/overview/>.
- [117] openssl. *Welcome to OpenSSL!* Disponible: 2023-05-10. URL: <https://www.openssl.org/>.
- [118] MDN. *Authorization*. Disponible: 2023-05-11. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization>.
- [119] Javier Ruiz Vázquez. *Observable*. Disponible: 2023-05-08. URL: <https://medium.com/javascript-comunidad/optimizando-aplicaciones-en-angular-pt-1-526367e4d26f>.
- [120] Christian Ludemann. *18 Performance Optimization Techniques For Angular Applications (Podcast With Michael Hladky)*. Disponible: 2023-05-08. URL: <https://christianlydemann.com/18-performance-optimization-techniques-for-angular-applications-podcast-with-michael-hladky/>.
- [121] rxjs. *Observable*. Disponible: 2023-05-08. URL: <https://rxjs.dev/guide/observable>.
- [122] MDN. *Crawler*. Disponible: 2023-05-11. URL: <https://developer.mozilla.org/en-US/docs/Glossary/Crawler>.
- [123] Karma. *Karma*. Disponible: 2023-05-11. URL: <https://karma-runner.github.io/latest/index.html>.
- [124] Wikipedia. *Autoridad de certificación*. Disponible: 2023-05-13. URL: https://es.wikipedia.org/wiki/Autoridad_de_certificaci%C3%B3n.
- [125] platformIO. *Espressif 32*. Disponible: 2023-05-15. URL: <https://docs.platformio.org/en/latest/platforms/espressif32.html>.
- [126] Arduino. *About Arduino*. Disponible: 2023-05-15. URL: <https://www.arduino.cc/en/about>.
- [127] Geekflare. *Explicación de las pruebas unitarias: qué es, por qué es importante y cómo empezar*. Disponible: 2023-05-13. URL: <https://geekflare.com/es/unit-testing-guide/>.
- [128] Manual. *BGH R-410A manual*. Disponible: 2023-06-04. URL: <https://www.manual.ar/bgh/bs30cns/manual>.
- [129] Scribd. *TP101 Thermometer*. Disponible: 2023-06-04. URL: <https://es.scribd.com/document/466736403/TP101-Thermometer-pdf>.
- [130] Angular. *Angular Internationalization*. Disponible: 2023-06-05. URL: <https://angular.io/guide/i18n-overview>.