



Sistema de monitoreo y gestión remota del clima en invernaderos

Lic. Martín Anibal Lacheski

Carrera de Especialización en Internet de las Cosas

Director: Mg. Lic. Leopoldo Alfredo Zimperz (FIUBA)

Jurados:

Jurado 1 (pertenencia)
Jurado 2 (pertenencia)
Jurado 3 (pertenencia)

Ciudad de Montecarlo, Misiones, junio de 2025



Sistema de monitoreo y gestión remota del clima en invernaderos

Lic. Martín Anibal Lacheski

Carrera de Especialización en Internet de las Cosas

Director: Mg. Lic. Leopoldo Alfredo Zimperz (FIUBA)

Jurados:

Esp. Ing. Juan Carlos Bampini Basualdo (FIUBA)
Esp. Lic. Claudio Omar Biale (FIUBA)
Esp. Bioing. Denis Jorge Genero (FIUBA)

Ciudad de Montecarlo, Misiones, junio de 2025

Agradecimientos

Esta sección es para agradecimientos personales y es totalmente **OPCIONAL**.

Agradecimientos

A Dios, por darme la fuerza y la sabiduría necesarias para alcanzar este objetivo.

A Valeria, por su amor, su paciencia y el apoyo incondicional que me acompañaron a lo largo de todo este proceso.

A mis padres, por estar siempre presentes con su cariño y aliento en cada etapa de mi vida.

A mi director, por su guía experta y constante acompañamiento a lo largo del proceso.

A los miembros del jurado, por su tiempo y dedicación al evaluar este trabajo.

A los docentes y revisores del taller de trabajo final, por su ayuda para dar forma a esta memoria.

A los docentes y al personal de apoyo de la carrera, por su compromiso y entrega en la formación académica.

A Matías, por su amistad, su orientación y aportes durante el desarrollo del trabajo.

A mis compañeros de cursada, por la colaboración y el compañerismo compartido.

A la Universidad Nacional de Misiones, por ser parte fundamental en mi crecimiento profesional.

Y a mí mismo, por la constancia, el esfuerzo y la dedicación que me permitieron alcanzar este logro.

Índice general

Resumen	1
1. Introducción general	
1.1. Problemática actual	1
1.2. Motivación	2
1.3. Estado del arte	2
1.4. Objetivos y alcance	3
1.4.1. Objetivo principal	3
1.4.2. Objetivos específicos	3
1.4.3. Alcance del trabajo	3
1.5. Requerimientos	4
2. Introducción específica	7
2.1. Protocolos de comunicación	7
2.1.1. Wi-Fi	7
2.1.2. MQTT	7
2.2. Componentes de hardware	8
2.2.1. Microcontrolador	8
2.2.2. Sensor de temperatura ambiente, humedad relativa y presión atmosférica	8
2.2.3. Sensor de luz digital	9
2.2.4. Sensor de dióxido de carbono	9
2.2.5. Sensor de detección de pH	9
2.2.6. Sensor de conductividad eléctrica	10
2.2.7. Sensor de sólidos disueltos totales	10
2.2.8. Sensor de temperatura digital sumergible	10
2.2.9. Sensor ultrasónico	11
2.2.10. Sensor de medición de consumo eléctrico	11
2.2.11. Módulo Relay	11
2.3. Desarrollo de firmware	12
2.3.1. MicroPython	12
2.4. Desarrollo backend y API	12
2.4.1. FastAPI	12
2.4.2. MongoDB	12
2.5. Desarrollo frontend	13
2.5.1. React	13
2.6. Infraestructura y despliegue	13
2.6.1. Docker	13
2.6.2. AWS IoT Core	13
2.6.3. AWS EC2	13
2.7. Herramientas de desarrollo	14
2.7.1. Visual Studio Code	14

Índice general

Resumen	1
1. Introducción general	1
1.1. Problemática actual	1
1.2. Motivación	2
1.3. Estado del arte	2
1.4. Objetivos y alcance	3
1.4.1. Objetivo principal	3
1.4.2. Objetivos específicos	3
1.4.3. Alcance del trabajo	3
1.5. Requerimientos	4
2. Introducción específica	7
2.1. Protocolos de comunicación	7
2.1.1. Wi-Fi	7
2.1.2. MQTT	7
2.1.3. TLS	8
2.2. Componentes de hardware	8
2.2.1. Microcontrolador	8
2.2.2. Sensor de temperatura ambiente, humedad relativa y presión atmosférica	8
2.2.3. Sensor de luz digital	9
2.2.4. Sensor de dióxido de carbono	9
2.2.5. Sensor de detección de pH	9
2.2.6. Sensor de conductividad eléctrica	10
2.2.7. Sensor de sólidos disueltos totales	10
2.2.8. Sensor de temperatura digital sumergible	10
2.2.9. Sensor ultrasónico	11
2.2.10. Sensor de medición de consumo eléctrico	11
2.2.11. Módulo de relés	11
2.3. Desarrollo de firmware	12
2.3.1. MicroPython	12
2.4. Desarrollo backend y API	12
2.4.1. FastAPI	12
2.4.2. MongoDB	12
2.5. Desarrollo frontend	13
2.5.1. React	13
2.6. Infraestructura y despliegue	13
2.6.1. Docker	13
2.6.2. AWS IoT Core	13
2.6.3. AWS EC2	13
2.7. Herramientas de desarrollo	14

2.7.2. Postman	14
2.7.3. GitHub	14
3. Diseño e implementación	15
3.1. Arquitectura del sistema	15
3.1.1. Capa de percepción	15
3.1.2. Capa de red	16
3.1.3. Capa de aplicación	16
3.2. Modelo de datos	16
3.2.1. Parametrización del sistema	17
3.2.2. Gestión de usuarios y roles	17
3.2.3. Sensores y actuadores	18
3.2.4. Auditoría y seguimiento	18
3.3. Servidor IoT	18
3.3.1. Arquitectura del servidor	18
3.4. Desarrollo del backend	19
3.4.1. Diseño de la API	19
3.4.2. Autenticación y autorización	20
3.4.3. Persistencia de datos	20
3.4.4. Comunicación con el broker MQTT	22
Pasos en AWS IoT Core	22
Implementación de MQTT en FastAPI	22
Comunicación con MQTT en FastAPI	23
3.4.5. Implementación de WebSockets	24
3.5. Desarrollo del frontend	24
3.6. Desarrollo de nodos sensores y actuadores	24
3.7. Despliegue del sistema	24
A. Modelo de datos implementado en el trabajo	25
B. Resumen de endpoints de la API	27
C. Autenticación con JWT	31
D. Conexión de la aplicación FastAPI con la base de datos MongoDB	33
E. Conexión al broker MQTT con FastAPI y AWS IoT SDK para Python	35
Bibliografía	39

2.7.1. Visual Studio Code	14
2.7.2. Postman	14
2.7.3. GitHub	14
3. Diseño e implementación	15
3.1. Arquitectura del sistema	15
3.1.1. Capa de percepción	15
3.1.2. Capa de red	16
3.1.3. Capa de aplicación	16
3.2. Modelo de datos	16
3.2.1. Parametrización del sistema	17
3.2.2. Gestión de usuarios y roles	17
3.2.3. Sensores y actuadores	18
3.2.4. Auditoría y seguimiento	18
3.3. Servidor IoT	18
3.3.1. Arquitectura del servidor	18
3.4. Desarrollo del backend	19
3.4.1. Diseño de la API	19
3.4.2. Autenticación y autorización	20
3.4.3. Persistencia de datos	20
3.4.4. Comunicación con el Broker MQTT	22
Configuración del Thing y Certificados de Seguridad	22
Gestión de Políticas de Acceso	22
Implementación de MQTT en FastAPI	22
Comunicación con MQTT en FastAPI	24
3.4.5. Implementación de WebSocket	24
3.5. Desarrollo del frontend	25
3.5.1. Tecnologías utilizadas	25
3.5.2. Arquitectura de la interfaz	25
3.5.3. Componentes principales de la interfaz de usuario	26
Gestión de autenticación	26
Layout	26
Arquitectura de navegación y control de acceso	27
3.5.4. Visualización de datos en tiempo real	28
3.5.5. Reportes de datos históricos	29
3.6. Desarrollo del firmware de los dispositivos IoT	31
3.6.1. Arquitectura general del firmware	31
3.6.2. Estructura del código	31
3.6.3. Gestión de la configuración	33
Archivo de configuración	33
Archivo de configuración del intervalo	33
Archivo de configuración Wi-Fi	33
Archivo de configuración de la zona horaria	33
3.6.4. Comunicación inalámbrica	33
Wi-Fi	34
MQTT	34
Comunicación y formato de mensajes	34
3.6.5. Gestión de tareas y concurrencia	35
3.6.6. Manejo de errores	35
Manejo de excepciones	35
Reintentos	35

Índice de figuras

1.1. Diagrama en bloques del sistema:	4
2.1. Microcontrolador ESP-WROOM-32 ¹ :	8
2.2. Sensor BME280 ² :	9
2.3. Sensor BH1750 ³ :	9
2.4. Sensor MHZ19C ⁴ :	9
2.5. Sensor PH-4502C ⁵ :	10
2.6. Sensor CE ⁶ :	10
2.7. Sensor TDS ⁷ :	10
2.8. Sensor de temperatura DS18B20 ⁸ :	11
2.9. Sensor HC-SR04 ⁹ :	11
2.10. Sensor de medición de consumo eléctrico ¹⁰ :	11
2.11. Relay de 2 Canales 5 V 10 A ¹¹ :	12
3.1. Arquitectura de la solución propuesta:	15
3.2. Modelo de datos implementado:	17
3.3. Arquitectura del servidor del sistema IoT:	18
3.4. Esquema de autenticación y autorización:	20
3.5. Diagrama de clases de los modelos implementados:	21
3.6. Pasos para la conexión de FastAPI y MongoDB:	21
3.7. Pasos para verificar la conexión con el broker MQTT:	23
3.8. Pasos para publicar un mensaje en un tópico específico:	23
3.9. Pasos para la conexión del cliente MQTT:	24
A.1. Modelo de datos implementado en el trabajo:	26

Logging	35
Optimización de recursos	35
3.6.7. Resumen de nodos sensores y actuadores	35
3.7. Despliegue del sistema	37
3.7.1. Entorno de prueba	37
Creación de la instancia EC2	37
Acceso remoto y configuración inicial	37
3.7.2. Instalación de componentes del sistema	37
Docker y Docker Compose	37
Cliente de actualización dinámica de DuckDNS	38
Clonación del repositorio y preparación	38
Configuración de certificados	38
Construcción y ejecución de los servicios	39
3.7.3. Configuración del sistema como servicio	39
3.7.4. Resumen del proceso de despliegue	39
3.8. Código fuente del sistema	40
4. Ensayos y resultados	41
4.1. Banco de pruebas	41
4.2. Criterios de evaluación	42
4.3. Pruebas de backend	43
4.3.1. Pruebas en Postman	44
Evaluación general	44
Resultados generales de las pruebas	45
4.3.2. Pruebas de validación de WebSocket	46
4.3.3. Pruebas de validación de almacenamiento en MongoDB	46
4.4. Pruebas de frontend	47
4.4.1. Pruebas de autenticación y autorización	48
Pruebas de autenticación	48
Pruebas de autenticación y autorización	48
4.4.2. Pruebas de funcionalidad	49
Validación de formularios	49
Validación de tablas y gráficos	50
Validación de gráficos con WebSocket	51
Pruebas de usabilidad	52
4.5. Pruebas de componentes	53
4.5.1. Pruebas de sensores	53
Prueba de sensor ambiental	53
Prueba de sensor de solución nutritiva	54
Prueba de sensor de consumos	54
4.5.2. Pruebas de actuadores	55
4.6. Pruebas de comunicación	55
4.6.1. Pruebas en sensor ambiental	55
4.6.2. Pruebas en sensor de consumos	56
4.6.3. Pruebas en sensor de solución nutritiva	57
4.6.4. Pruebas de comunicación en el actuador	57
4.7. Prueba integral del sistema	58
5. Conclusiones	61
5.1. Conclusiones generales	61
5.2. Próximos pasos	62

VIII

A. Modelo de datos implementado en el trabajo	63
B. Resumen de endpoints de la API	65
C. Despliegue del sistema en instancia EC2	69
C.1. Introducción	69
C.2. Creación de la instancia EC2	69
C.3. Instalación y configuración del servidor EC2	71
C.3.1. Conectarse a la instancia EC2	71
C.3.2. Instalación de Docker	72
C.3.3. Instalación de Docker Compose	72
C.3.4. Instalación del cliente DuckDNS	73
C.3.5. Descarga y configuración del repositorio EnviroSense	74
C.3.6. Configuración del sistema como servicio	75
Bibliografía	77

Índice de tablas

1.1. Características de la competencia.	2
3.1. Resumen de principales endpoints de la API	19
B.1. Resumen de principales endpoints de la API	27
B.2. Resumen de principales endpoints de la API	28
B.3. Resumen de principales endpoints de la API	29

Índice de figuras

1.1. Diagrama en bloques del sistema.	4
2.1. Microcontrolador ESP-WROOM-32 ¹ .	8
2.2. Sensor BME280 ² .	9
2.3. Sensor BH1750 ³ .	9
2.4. Sensor MH-Z19C ⁴ .	9
2.5. Sensor PH-4502C ⁵ .	10
2.6. Sensor CE ⁶ .	10
2.7. Sensor TDS ⁷ .	10
2.8. Sensor de temperatura DS18B20 ⁸ .	11
2.9. Sensor HC-SR04 ⁹ .	11
2.10. Sensor de medición de consumo eléctrico ¹⁰ .	11
2.11. Relé de 2 Canales 5 V 10 A ¹¹ .	12
3.1. Arquitectura de la solución propuesta.	15
3.2. Modelo de datos implementado.	17
3.3. Arquitectura del servidor del sistema IoT.	18
3.4. Esquema de autenticación y autorización.	20
3.5. Diagrama de clases de los modelos implementados.	21
3.6. Pasos para la conexión de FastAPI y MongoDB.	21
3.7. Pasos para verificar la conexión con el broker MQTT.	23
3.8. Pasos para publicar un mensaje en un tópico específico.	23
3.9. Pasos para la conexión del cliente MQTT.	24
3.10. Pasos para establecer la conexión WebSocket.	25
3.11. Pantalla de Login.	26
3.12. Estructura del componente layout.	27
3.13. Interfaz de navegación según permisos de usuario estándar.	28
3.14. Visualización de datos en tiempo real.	28
3.15. Pantalla de reportes.	29
3.16. Pantalla de reportes.	30
3.17. Pantalla de reportes.	30
3.18. Visualización de datos en tiempo real desde un dispositivo móvil.	31
3.19. Flujo de ejecución del código en los nodos.	32
3.20. Flujo de comunicación entre nodos y broker MQTT.	34
4.1. Banco de pruebas del sistema EnviroSenseloT.	42
4.2. Interfaz de Swagger.	43
4.3. Pruebas realizadas en Postman.	44
4.4. Pruebas de WebSocket.	46
4.5. Validación de almacenamiento en MongoDB.	47
4.6. Pruebas de autenticación.	48
4.7. Autenticación y autorización rol administrador.	49
4.8. Autenticación y autorización rol usuario.	49

4.9. Validación de formularios	50
4.10. Verificación de creación de usuario	50
4.11. Pruebas de funcionalidad en tablas	51
4.12. Pruebas de funcionalidad en gráficos	51
4.13. Pruebas de funcionalidad en el dashboard	52
4.14. Interfaz de ambientes en dispositivo móvil	52
4.15. Listado en formato apaisado desde dispositivo móvil	53
4.16. Pruebas de sensor ambiental	54
4.17. Pruebas de sensor de solución nutritiva	54
4.18. Pruebas de sensor de consumos	54
4.19. Pruebas de actuadores	55
4.20. Pruebas de encendido de relés	55
4.21. Pruebas de comunicación con sensor ambiental	56
4.22. Pruebas de comunicación con sensor ambiental	56
4.23. Pruebas de comunicación con sensor de consumos	56
4.24. Pruebas de comunicación con sensor de consumos	56
4.25. Pruebas de comunicación con sensor de solución nutritiva	57
4.26. Pruebas de comunicación con sensor de solución nutritiva	57
4.27. Pruebas de comunicación con actuador	57
4.28. Pruebas de comunicación con actuador	58
4.29. Pruebas de comunicación con actuador	58
4.30. Envío de comando de activación al microcontrolador	58
4.31. Salida por consola del frontend	59
4.32. Salida por consola del backend	59
4.33. Salida por consola del microcontrolador	59
4.34. Salida por consola del backend	60
4.35. Verificación de persistencia en MongoDB Compass	60
4.36. Actualización del estado de los relés en el frontend	60
A.1. Modelo de datos implementado en el trabajo.	64
C.1. Resumen de la instancia EC2 creada.	70
C.2. Conexión SSH a la instancia EC2	71
C.3. Acceso a la aplicación web EnviroSenseloT.	76
C.4. Visualización de datos en tiempo real.	76

Dedicado a... [OPCIONAL]

Índice de tablas

1.1. Características de la competencia.	2
3.1. Resumen de principales endpoints de la API	19
3.2. Resumen de nodos	36
3.3. Resumen del despliegue del sistema	39
4.1. Criterios de evaluación del banco de pruebas	43
4.2. Resultados de tiempos de respuesta	45
4.3. Promedios de tiempos de respuesta	45
B.1. Resumen de principales endpoints de la API	65
B.2. Resumen de principales endpoints de la API	66
B.3. Resumen de principales endpoints de la API.	67

Capítulo 1

Introducción general

Este capítulo presenta una visión general de los sistemas de gestión y monitoreo en invernaderos, se abordan los desafíos actuales y las oportunidades de mejora en el ámbito de la agricultura. Se describe la problemática relacionada con la falta de optimización en los sistemas de cultivo tradicionales. Además, se describen la motivación, los objetivos, el alcance y los requerimientos asociados a los diferentes componentes del sistema.

1.1. Problemática actual

La agricultura enfrenta desafíos crecientes en la optimización de la productividad y la eficiencia, especialmente en regiones con condiciones climáticas adversas y variables. Según la FAO (del inglés, *Food and Agriculture Organization of the United Nations*) [1], para el año 2050, se estima que la población superará los 9 mil millones de personas, lo que demandará un aumento del 60 % en la producción de alimentos. Para abordar este desafío, es fundamental optimizar el uso del agua, mejorar la productividad agrícola y fomentar prácticas que contribuyan a la sostenibilidad ambiental.

Ante estos retos, los cultivos hidropónicos han surgido como una solución prometedora debido a su capacidad para utilizar los recursos de manera más eficiente. Entre sus principales ventajas se destacan la reducción en el consumo de agua [2], la posibilidad de cultivar durante todo el año en entornos controlados y un aumento significativo en la productividad, gracias a la mayor velocidad de crecimiento y rendimiento de los cultivos.

En la provincia de Misiones, la producción hidropónica ha experimentado un crecimiento notable en los últimos años [3], [4]. No obstante, persisten desafíos en la gestión eficiente de los recursos esenciales. Actualmente, la mayoría de los productores emplean sistemas de control basados en temporizadores programables, los cuales no consideran las variaciones ambientales. Esto implica la necesidad de intervenciones manuales frecuentes y mediciones directas, limitando la eficiencia del proceso.

La ausencia de un monitoreo en tiempo real impacta negativamente en la calidad y el rendimiento de los cultivos, aumentando los costos operativos y afectando la sostenibilidad ambiental debido a la implementación de prácticas poco optimizadas.

Capítulo 1

Introducción general

Este capítulo presenta una visión general de los sistemas de gestión y monitoreo en invernaderos, se abordan los desafíos actuales y las oportunidades de mejora en el ámbito de la agricultura. Se describe la problemática relacionada con la falta de optimización en los sistemas de cultivo tradicionales. Además, se describen la motivación, los objetivos, el alcance y los requerimientos asociados a los diferentes componentes del sistema.

1.1. Problemática actual

La agricultura enfrenta desafíos crecientes en la optimización de la productividad y la eficiencia, especialmente en regiones con condiciones climáticas adversas y variables. Según la FAO (del inglés, *Food and Agriculture Organization of the United Nations*) [1], para el año 2050, se estima que la población superará los 9 mil millones de personas, lo que demandará un aumento del 60 % en la producción de alimentos. Para abordar este desafío, es fundamental optimizar el uso del agua, mejorar la productividad agrícola y fomentar prácticas que contribuyan a la sostenibilidad ambiental.

Ante estos retos, los cultivos hidropónicos han surgido como una solución prometedora debido a su capacidad para utilizar los recursos de manera más eficiente. Entre sus principales ventajas se destacan la reducción en el consumo de agua [2], la posibilidad de cultivar durante todo el año en entornos controlados y un aumento significativo en la productividad, gracias a la mayor velocidad de crecimiento y rendimiento de los cultivos.

En la provincia de Misiones, la producción hidropónica ha experimentado un crecimiento notable en los últimos años [3], [4]. No obstante, persisten desafíos en la gestión eficiente de los recursos esenciales. Actualmente, la mayoría de los productores emplean sistemas de control basados en temporizadores programables, los cuales no consideran las variaciones ambientales. Esto implica la necesidad de intervenciones manuales frecuentes y mediciones directas, lo que limita la eficiencia del proceso.

La falta de monitoreo en tiempo real impacta negativamente en la calidad y el rendimiento de los cultivos, incrementa los costos operativos y afecta la sostenibilidad ambiental debido a la implementación de prácticas ineficientes.

1.4. Objetivos y alcance

1.4.1. Objetivo principal

Diseñar y desarrollar un prototipo de sistema para el monitoreo y control remoto de las condiciones climáticas en invernaderos, mediante sensores y actuadores conectados a través de Wi-Fi, un servidor IoT en la nube y una aplicación web, con el fin de optimizar el uso de los recursos, reducir costos operativos y mejorar la sostenibilidad ambiental, además de servir como plataforma de datos para la investigación académica y científica.

1.4.2. Objetivos específicos

- Implementar una arquitectura IoT basada en Wi-Fi para monitorear sensores y actuadores en tiempo real.
- Desarrollar un servidor IoT en la nube para la recolección, almacenamiento y procesamiento de los datos obtenidos.
- Diseñar una aplicación web que permita la visualización en tiempo real y el control remoto de las condiciones del invernadero.
- Facilitar el acceso a los datos generados para su uso en investigaciones académicas, trabajos finales y estudios específicos.

1.4.3. Alcance del trabajo

El alcance del trabajo incluyó las siguientes tareas:

- Diseño e implementación de nodos IoT.
 - Selección de sensores, actuadores y microcontroladores.
 - Configuración de conexión Wi-Fi en nodos sensores y actuadores.
 - Desarrollo de firmware para la adquisición de datos de los sensores y el control de los actuadores.
- Comunicación y protocolos.
 - Configuración de un servidor IoT para gestión de mensajes entre nodos y aplicaciones.
 - Transmisión de datos al servidor IoT mediante MQTT (del inglés, *Message Queue Telemetry Transport*).
 - Cifrado de comunicaciones mediante TLS (del inglés, *Transport Layer Security*).
- Desarrollo de software.
 - Diseño e implementación de una base de datos para almacenar los datos recolectados por los sensores y permitir su consulta y análisis.
 - Diseño y desarrollo de una API (del inglés, *Application Programming Interface*) REST (del inglés, *Representational State Transfer*) que permita la comunicación con el sistema utilizando HTTP (del inglés, *Hypertext Transfer Protocol*), MQTT y *WebSockets*.

1.4. Objetivos y alcance

1.4.1. Objetivo principal

Diseñar y desarrollar un prototipo de sistema para el monitoreo y control remoto de las condiciones climáticas en invernaderos, mediante sensores y actuadores conectados a través de Wi-Fi, un servidor IoT en la nube y una aplicación web, con el fin de optimizar el uso de los recursos, reducir costos operativos y mejorar la sostenibilidad ambiental, además de servir como plataforma de datos para la investigación académica y científica.

1.4.2. Objetivos específicos

- Implementar una arquitectura IoT basada en Wi-Fi para monitorear sensores y actuadores en tiempo real.
- Desarrollar un servidor IoT en la nube para la recolección, almacenamiento y procesamiento de los datos obtenidos.
- Diseñar una aplicación web que permita la visualización en tiempo real y el control remoto de las condiciones del invernadero.
- Facilitar el acceso a los datos generados para su uso en investigaciones académicas, trabajos finales y estudios específicos.

1.4.3. Alcance del trabajo

El alcance del trabajo incluyó las siguientes tareas:

- Diseño e implementación de nodos IoT.
 - Selección de sensores, actuadores y microcontroladores.
 - Configuración de conexión Wi-Fi en nodos sensores y actuadores.
 - Desarrollo de firmware para la adquisición de datos de los sensores y el control de los actuadores.
- Comunicación y protocolos.
 - Configuración de un servidor IoT para gestión de mensajes entre nodos y aplicaciones.
 - Transmisión de datos al servidor IoT mediante MQTT (del inglés, *Message Queue Telemetry Transport*).
 - Cifrado de comunicaciones mediante TLS (del inglés, *Transport Layer Security*).
- Desarrollo de software.
 - Diseño e implementación de una base de datos para almacenar los datos recolectados por los sensores y permitir su consulta y análisis.
 - Diseño y desarrollo de una API (del inglés, *Application Programming Interface*) REST (del inglés, *Representational State Transfer*) que permita la comunicación con el sistema a través de HTTP (del inglés, *Hypertext Transfer Protocol*), MQTT y *WebSockets*.

1.5. Requerimientos

5

- 1) Nodos ambientales: temperatura ambiente, humedad relativa, presión atmosférica, nivel de luminosidad y nivel de CO_2 .
- 2) Nodos de solución nutritiva: valores de pH (potencial de Hidrógeno), conductividad eléctrica (CE) y TDS (del inglés, *Total Dissolved Solids*); nivel y temperatura de la solución.
- 3) Nodos de consumos: agua, nutrientes y energía eléctrica.
- g) Los nodos actuadores deben transmitir al servidor IoT:
 - 1) Configuración remota de parámetros por cada canal.
 - 2) Reporte del estado de cada canal.
- h) Los nodos actuadores deben recibir desde el servidor IoT:
 - 1) Comandos de activación/desactivación remota de canales.
2. Broker MQTT:
 - a) Soportar conexiones cifradas mediante TLS.
 - b) Poseer comunicación bidireccional (publicación/suscripción).
 - c) Implementar QoS (del inglés, *Quality of Service*) para garantizar entrega de mensajes.
3. Frontend (aplicación web)
 - a) Interfaz intuitiva y responsive (accesible desde móviles y escritorio).
 - b) Autenticación de usuarios mediante credenciales.
 - c) Realización de las operaciones CRUD (**Crear, Leer, Actualizar, Eliminar**).
 - d) Visualización en tiempo real de datos de sensores y actuadores.
 - e) Envío remoto de comandos y configuraciones.
 - f) Acceso a datos históricos mediante gráficos y tablas.
 - g) Tablero interactivo para monitoreo y control centralizado.
4. Backend:
 - a) Tener conexiones seguras mediante TLS.
 - b) Implementar JWT (del inglés, *JSON Web Token*).
 - c) Realizar la persistencia de los datos.
 - d) Soportar métodos HTTP (CRUD y reportes), WebSockets (datos en tiempo real) y MQTT (interacción con dispositivos).
5. Requerimientos de documentación:
 - a) Se entregará el código del sistema, que incluye todos los componentes desarrollados (sensores, actuadores, broker MQTT, frontend, backend y API).
 - b) Se entregarán las guías y diagramas de instalación, configuración y operación.

1.5. Requerimientos

5

- 1) Nodos ambientales: temperatura ambiente, humedad relativa, presión atmosférica, nivel de luminosidad y nivel de CO_2 .
- 2) Nodos de solución nutritiva: valores de pH (potencial de Hidrógeno), conductividad eléctrica (CE) y TDS (del inglés, *Total Dissolved Solids*); nivel y temperatura de la solución.
- 3) Nodos de consumos: nutrientes y energía eléctrica.
- g) Los nodos actuadores deben transmitir al servidor IoT:
 - 1) Configuración remota de parámetros por cada canal.
 - 2) Reporte del estado de cada canal.
- h) Los nodos actuadores deben recibir desde el servidor IoT:
 - 1) Comandos de activación/desactivación de canales.
2. Broker MQTT:
 - a) Soportar conexiones cifradas mediante TLS.
 - b) Poseer comunicación bidireccional (publicación/suscripción).
 - c) Implementar QoS (del inglés, *Quality of Service*) para garantizar entrega de mensajes.
3. Frontend:
 - a) Interfaz intuitiva y responsive (accesible desde móviles y escritorio).
 - b) Autenticación de usuarios mediante credenciales.
 - c) Realización de las operaciones CRUD (**Crear, Leer, Actualizar, Eliminar**).
 - d) Visualización en tiempo real de datos de sensores y actuadores.
 - e) Envío remoto de comandos y configuraciones.
 - f) Acceso a datos históricos mediante gráficos y tablas.
 - g) Tablero interactivo para el monitoreo y control centralizado.
4. Backend:
 - a) Tener conexiones seguras mediante TLS.
 - b) Implementar JWT (del inglés, *JSON Web Token*).
 - c) Realizar la persistencia de los datos.
 - d) Soportar métodos HTTP (CRUD y reportes), WebSocket (datos en tiempo real) y MQTT (interacción con dispositivos).
5. Requerimientos de documentación:
 - a) Se entregará el código del sistema, que incluye todos los componentes desarrollados (sensores, actuadores, broker MQTT, frontend, backend y API).

Capítulo 2

Introducción específica

En este capítulo se presentan los protocolos de comunicación, componentes de hardware y herramientas de software utilizados en el desarrollo del trabajo. Se detallan las características y sus especificaciones técnicas.

2.1. Protocolos de comunicación

En esta sección se describen los diferentes protocolos de comunicación utilizados en el desarrollo del trabajo.

2.1.1. Wi-Fi

Wi-Fi es el nombre **comercial** propiedad de la Wi-Fi Alliance para designar a su familia de protocolos de comunicación inalámbrica basados en el estándar IEEE 802.11 para redes de área local sin cables [10].

El estandar identifica dos modos principales de topología de red: infraestructura y ad-hoc.

- Modo infraestructura: los dispositivos se conectan a una red inalámbrica a través de un router o AP (del inglés, *Access Point*) inalámbrico, como en las WLAN. Los AP se conectan a la infraestructura de la red mediante el sistema de distribución conectado por cable o de manera inalámbrica.
- Modo ad-hoc: los dispositivos se conectan directamente entre sí sin necesidad de un punto de acceso.

2.1.2. MQTT

MQTT es un protocolo de mensajería estándar internacional OASIS [11] para Internet de las Cosas (IoT). Está diseñado como un transporte de mensajería de publicación/suscripción **extremadamente** ligero, ideal para conectar dispositivos remotos con un consumo de código reducido y un ancho de banda de red **mínimo**.

MQTT es un protocolo ligero basado en TCP/IP [12] que sigue un modelo de publicación/suscripción, donde:

- Broker: funciona como un servidor central que recibe los mensajes de los clientes y los distribuye a los suscriptores correspondientes, actúa como intermediario en la comunicación.

Capítulo 2

Introducción específica

Este capítulo presenta los protocolos de comunicación, componentes de hardware y herramientas de software utilizados en el desarrollo del trabajo. Se detallan las características y sus especificaciones técnicas.

2.1. Protocolos de comunicación

En esta sección se describen los diferentes protocolos de comunicación utilizados en el desarrollo del trabajo.

2.1.1. Wi-Fi

Wi-Fi es el nombre **comercial**, propiedad de la Wi-Fi Alliance, que designa a una familia de protocolos de comunicación inalámbrica basados en el estándar IEEE 802.11 para redes de área local sin cables [10].

El estandar identifica dos modos principales de topología de red: infraestructura y ad-hoc.

- Modo infraestructura: los dispositivos se conectan a una red inalámbrica a través de un router o AP (del inglés, *Access Point*) inalámbrico, como en las WLAN. Los AP se conectan a la infraestructura de la red mediante el sistema de distribución conectado por cable o de manera inalámbrica.
- Modo ad-hoc: los dispositivos se conectan directamente entre sí sin necesidad de un punto de acceso.

2.1.2. MQTT

MQTT es un protocolo de mensajería estándar internacional OASIS [11] para IoT. Está diseñado como un transporte de mensajería de publicación/suscripción **extremadamente** ligero, ideal para conectar dispositivos remotos con un consumo de código reducido y un ancho de banda de red **mínimo**.

Basado en TCP/IP [12], MQTT implementa un modelo de **comunicación de publicación/suscripción**, en el cual:

- Broker: funciona como un servidor central que recibe los mensajes de los clientes y los distribuye a los suscriptores correspondientes, actúa como intermediario en la comunicación.
- Cliente: puede ser un dispositivo que publica mensajes en un **tópico** o que recibe mensajes al estar suscrito a un **tópico**.

- Cliente: puede ser un dispositivo que publica mensajes en un tópico o que recibe mensajes al estar suscrito a un tópico.
- Tópico: es la dirección a la que se envían los mensajes en MQTT. El broker se encarga de distribuirlos a los clientes suscritos. Los temas se organizan en una estructura jerárquica de tópicos.

TLS es un protocolo de seguridad criptográfica diseñado para garantizar la privacidad y la integridad de los datos en comunicaciones sobre redes, como Internet [13]. Opera sobre la capa de transporte y permite autenticación, cifrado de datos y protección contra manipulación.

TLS se utiliza para garantizar la confidencialidad de los protocolos de aplicación (MQTT [11], HTTP [14] y WebSocket [15]) [16].

2.2. Componentes de hardware

En esta sección se describen los diferentes elementos de hardware utilizados en el desarrollo del trabajo.

2.2.1. Microcontrolador

El microcontrolador ESP-WROOM-32 (figura 2.1), es un chip de tipo SoC (del inglés, *System on Chip*) de bajo costo y bajo consumo de energía que integra WiFi, Bluetooth y Bluetooth LE en un solo paquete. El ESP-WROOM-32 [17] es un microcontrolador de 32 bits con una arquitectura Xtensa LX6 de doble núcleo, lo que le permite ejecutar dos hilos de ejecución simultáneos. Además, cuenta con una amplia gama de periféricos, como UART, I2C, SPI y ADC, que lo hace ideal para aplicaciones de IoT.



FIGURA 2.1. Microcontrolador ESP-WROOM-32¹.

2.2.2. Sensor de temperatura ambiente, humedad relativa y presión atmosférica

El BME280 (figura 2.2) es un sensor digital de alta precisión para la medición de temperatura ambiente, humedad relativa y presión atmosférica. Se comunica a través de las interfaces I2C y SPI y ofrece una precisión de $\pm 1^{\circ}\text{C}$ para la temperatura ambiente, $\pm 3\%$ para la humedad relativa y $\pm 1 \text{ hPa}$ para la presión atmosférica [18].

¹Imagen tomada de [Nodemcu Esp32 Wifi HobbyTronica](#).

- Tópico: es la dirección a la que se envían los mensajes en MQTT. El broker se encarga de distribuirlos a los clientes suscritos. Los temas se organizan en una estructura jerárquica de tópicos.

2.1.3. TLS

TLS es un protocolo de seguridad criptográfica diseñado para garantizar la privacidad y la integridad de los datos en comunicaciones sobre redes, como Internet [13]. Opera sobre la capa de transporte y permite autenticación, cifrado de datos y protección contra manipulación.

TLS se utiliza para garantizar la confidencialidad de los protocolos de aplicación (MQTT [11], HTTP [14] y WebSocket [15]) [16].

2.2. Componentes de hardware

En esta sección se describen los diferentes elementos de hardware utilizados en el desarrollo del trabajo.

2.2.1. Microcontrolador

El microcontrolador ESP-WROOM-32 (figura 2.1), es un chip de tipo SoC (del inglés, *System on Chip*) de bajo costo y bajo consumo de energía que integra WiFi, Bluetooth y Bluetooth LE en un solo paquete. El ESP-WROOM-32 [17] es un microcontrolador de 32 bits con una arquitectura Xtensa LX6 de doble núcleo, lo que le permite ejecutar dos hilos de ejecución simultáneos. Además, cuenta con una amplia gama de periféricos, como UART, I2C, SPI y ADC, que lo hace ideal para aplicaciones de IoT.



FIGURA 2.1. Microcontrolador ESP-WROOM-32¹.

2.2.2. Sensor de temperatura ambiente, humedad relativa y presión atmosférica

El BME280 (figura 2.2) es un sensor digital de alta precisión para la medición de temperatura ambiente, humedad relativa y presión atmosférica. Se comunica a través de las interfaces I2C y SPI y ofrece una precisión de $\pm 1^{\circ}\text{C}$ para la temperatura ambiente, $\pm 3\%$ para la humedad relativa y $\pm 1 \text{ hPa}$ para la presión atmosférica [18].

¹Imagen tomada de [Nodemcu Esp32 Wifi HobbyTronica](#).

2.2. Componentes de hardware

9

FIGURA 2.2. Sensor BME280².

2.2.3. Sensor de luz digital

El BH1750 (figura 2.3) es un sensor digital de intensidad **luminosa** que mide la iluminación ambiental en lux. Utiliza la interfaz I2C para la comunicación y puede medir niveles de luz en un rango de 1 a 65.535 lux, con una precisión de 1 lux [19].

FIGURA 2.3. Sensor BH1750³.

2.2.4. Sensor de dióxido de carbono

El sensor MH-Z19C (figura 2.4) es un detector de CO_2 por NDIR (del inglés, *Non Dispersive Infrared Detector*). Se comunica a través de la interfaz UART y es capaz de medir la concentración de CO_2 en un rango de 0 a 5000 ppm con una precisión de 50 ppm [20].

FIGURA 2.4. Sensor MH-Z19C⁴.

2.2.5. Sensor de detección de pH

El sensor PH-4502C (figura 2.5) mide la acidez o alcalinidad del líquido mediante un electrodo de vidrio. Se comunica a través de la interfaz analógica y es capaz de medir el pH en un rango de 0 a 14 [21].

²Imagen tomada de [Sensor BME280 Naylamp Mechatronics](#).

³Imagen tomada de [Sensor BH1750 HobbyTronica](#).

⁴Imagen tomada de [MH-Z19C PartsCabi.net](#).

2.2. Componentes de hardware

9

FIGURA 2.2. Sensor BME280².

2.2.3. Sensor de luz digital

El BH1750 (figura 2.3) es un sensor digital de intensidad **luminosa** que mide la iluminación ambiental en lux. Utiliza la interfaz I2C para la comunicación y puede medir niveles de luz en un rango de 1 a 65.535 lux, con una precisión de 1 lux [19].

FIGURA 2.3. Sensor BH1750³.

2.2.4. Sensor de dióxido de carbono

El sensor MH-Z19C (figura 2.4) es un detector de CO_2 por NDIR (del inglés, *Non Dispersive Infrared Detector*). Se comunica a través de la interfaz UART y es capaz de medir la concentración de CO_2 en un rango de 0 a 5000 ppm con una precisión de 50 ppm [20].

FIGURA 2.4. Sensor MH-Z19C⁴.

2.2.5. Sensor de detección de pH

El sensor PH-4502C (figura 2.5) mide la acidez o alcalinidad del líquido mediante un electrodo de vidrio. Se comunica a través de la interfaz analógica y es capaz de medir el pH en un rango de 0 a 14 [21].

²Imagen tomada de [Sensor BME280 Naylamp Mechatronics](#).

³Imagen tomada de [Sensor BH1750 HobbyTronica](#).

⁴Imagen tomada de [MH-Z19C PartsCabi.net](#).

FIGURA 2.5. Sensor PH-4502C⁵.

2.2.6. Sensor de conductividad eléctrica

El sensor de CE (figura 2.6) mide la capacidad de una solución para conducir electricidad, lo que depende de la presencia de iones. A mayor concentración de iones, mayor es la conductividad [22]. Este sensor se comunica a través de una interfaz analógica y puede medir la conductividad en un rango de 0 a 20 mS/cm [23].

FIGURA 2.6. Sensor CE⁶.

2.2.7. Sensor de sólidos disueltos totales

El sensor TDS (figura 2.7) mide la cantidad de sales, minerales y metales que se encuentran disueltos en la solución [24]. Se comunica a través de la interfaz analógica y es capaz de medir la concentración de TDS en un rango de 0 a 1000 ppm [25].

FIGURA 2.7. Sensor TDS⁷.

2.2.8. Sensor de temperatura digital sumergible

El DS18B20 (figura 2.8) es un sensor digital de temperatura sumergible. Se comunica a través de la interfaz OneWire y puede medir la temperatura en un rango de -55 °C a 125 °C con una precisión de ±0.5 °C [26].

⁵Imagen tomada de [Sensor PH-4502C Mercado Libre Static](#).

⁶Imagen tomada de [Sensor CE Amazon](#).

⁷Imagen tomada de [Sensor TDS Aliexpress](#).

FIGURA 2.5. Sensor PH-4502C⁵.

2.2.6. Sensor de conductividad eléctrica

El sensor de CE (figura 2.6) mide la capacidad de una solución para conducir electricidad, lo que depende de la presencia de iones. A mayor concentración de iones, mayor es la conductividad [22]. Este sensor se comunica a través de una interfaz analógica y puede medir la conductividad en un rango de 0 a 20 mS/cm [23].

FIGURA 2.6. Sensor CE⁶.

2.2.7. Sensor de sólidos disueltos totales

El sensor TDS (figura 2.7) mide la cantidad de sales, minerales y metales que se encuentran disueltos en la solución [24]. Se comunica a través de la interfaz analógica y es capaz de medir la concentración de TDS en un rango de 0 a 1000 ppm [25].

FIGURA 2.7. Sensor TDS⁷.

2.2.8. Sensor de temperatura digital sumergible

El DS18B20 (figura 2.8) es un sensor digital de temperatura sumergible. Se comunica a través de la interfaz OneWire y permite medir la temperatura en un rango de -55 °C a 125 °C con una precisión de ±0.5 °C [26].

⁵Imagen tomada de [Sensor PH-4502C Mercado Libre Static](#).

⁶Imagen tomada de [Sensor CE Amazon](#).

⁷Imagen tomada de [Sensor TDS Aliexpress](#).

2.2. Componentes de hardware

11

FIGURA 2.8. Sensor de temperatura DS18B20⁸.

2.2.9. Sensor ultrasónico

El sensor HC-SR04 (figura 2.9) mide distancias por ultrasonido en un rango de 2 cm a 400 cm con una precisión de 3 mm. Se comunica a través de la interfaz GPIO [27].

FIGURA 2.9. Sensor HC-SR04⁹.

2.2.10. Sensor de medición de consumo eléctrico

El sensor PZEM-004T (figura 2.10) es un módulo de medición de parámetros eléctricos que mide la tensión, corriente, potencia activa y energía consumida. Se comunica a través de la interfaz UART y es capaz de medir la tensión en un rango de 80 a 260 V, la corriente en un rango de 0 a 100 A, y la potencia en un rango de 0 a 22 kW [28].

FIGURA 2.10. Sensor de medición de consumo eléctrico¹⁰.

2.2.11. Módulo Relay

El módulo Relay (figura 2.11) es un actuador eléctrico de **dos canales optocopladores** que permite el control de encendido y apagado de dispositivos eléctricos. Se comunica a través de la interfaz GPIO y es capaz de controlar dispositivos de hasta 10 A y 250 VAC [29].

⁸Imagen tomada de [Sensor DS18B20 Mercado Libre](#).

⁹Imagen tomada de [Sensor HC-SR04 Aliexpress](#).

¹⁰Imagen adaptada de [Sensor PZEM-004T Mercado Libre](#).

2.2. Componentes de hardware

11

FIGURA 2.8. Sensor de temperatura DS18B20⁸.

2.2.9. Sensor ultrasónico

El sensor HC-SR04 (figura 2.9) mide distancias por ultrasonido en un rango de 2 cm a 400 cm con una precisión de 3 mm. Se comunica a través de la interfaz GPIO [27].

FIGURA 2.9. Sensor HC-SR04⁹.

2.2.10. Sensor de medición de consumo eléctrico

El sensor PZEM-004T (figura 2.10) es un módulo de medición de parámetros eléctricos que mide la tensión, corriente, potencia activa y energía consumida. Se comunica a través de la interfaz UART y es capaz de medir la tensión en un rango de 80 a 260 V, la corriente en un rango de 0 a 100 A, y la potencia en un rango de 0 a 22 kW [28].

FIGURA 2.10. Sensor de medición de consumo eléctrico¹⁰.

2.2.11. Módulo de relés

El módulo de **relés** (figura 2.11) es un actuador eléctrico de canales optocoplados que permite el control de encendido y apagado de dispositivos eléctricos. Se comunica a través de la interfaz GPIO y es capaz de controlar dispositivos de hasta 10 A y 250 VAC [29].

⁸Imagen tomada de [Sensor DS18B20 Mercado Libre](#).

⁹Imagen tomada de [Sensor HC-SR04 Aliexpress](#).

¹⁰Imagen adaptada de [Sensor PZEM-004T Mercado Libre](#).

FIGURA 2.11. Relay de 2 Canales 5 V 10 A¹¹.

2.3. Desarrollo de firmware

En esta sección se describe la herramienta de software utilizada para la programación de los microcontroladores ESP32.

2.3.1. MicroPython

MicroPython es una implementación optimizada de Python 3 para microcontroladores y sistemas embebidos. Está diseñado para ejecutarse en dispositivos con recursos limitados, como el ESP32, y proporciona una forma sencilla de programar microcontroladores con un lenguaje de alto nivel como Python [30].

Su facilidad de uso, la amplia disponibilidad de bibliotecas y la reducción del tiempo de desarrollo lo convierten en una opción eficiente. Además, al ser un lenguaje interpretado, posibilita la ejecución interactiva de pruebas y depuración, facilitando la identificación y corrección de errores en el código [31].

2.4. Desarrollo backend y API

En esta sección se presentan las herramientas de software utilizadas en el desarrollo del backend y la API REST.

2.4.1. FastAPI

FastAPI es un framework moderno para la construcción de APIs REST rápidas y escalables en Python. Está diseñado para ser fácil de usar, rápido de desarrollar y altamente eficiente en términos de rendimiento. FastAPI utiliza Python 3.6+ y aprovecha las características de tipado estático de Python para proporcionar una API autodocumentada y con validación de tipos integrada [32].

2.4.2. MongoDB

MongoDB es una base de datos NoSQL (del inglés, *Not Only SQL*) de código abierto y orientada a documentos que proporciona una forma flexible y escalable de almacenar y recuperar datos. Utiliza un modelo de datos basado en documentos que almacena datos en un formato similar a JSON (del inglés, *JavaScript Object Notation*) llamado BSON (del inglés, *Binary JSON*) que permite almacenar datos de forma anidada y sin esquema fijo, lo que facilita la manipulación y consulta de datos no estructurados [33].

¹¹Imagen tomada de [Relay de 2 Canales Amazon](#).

FIGURA 2.11. Relé de 2 Canales 5 V 10 A¹¹.

2.3. Desarrollo de firmware

En esta sección se describe la herramienta de software utilizada para la programación de los microcontroladores ESP32.

2.3.1. MicroPython

MicroPython es una implementación optimizada de Python 3 para microcontroladores y sistemas embebidos. Está diseñado para ejecutarse en dispositivos con recursos limitados, como el ESP32, y proporciona una forma sencilla de programar microcontroladores con un lenguaje de alto nivel como Python [30].

Su facilidad de uso, la amplia disponibilidad de bibliotecas y la reducción del tiempo de desarrollo lo convierten en una opción eficiente. Además, al ser un lenguaje interpretado, posibilita la ejecución interactiva de pruebas y depuración, lo que permite identificar y corregir errores en el código [31].

2.4. Desarrollo backend y API

En esta sección se presentan las herramientas de software utilizadas en el desarrollo del backend y la API REST.

2.4.1. FastAPI

FastAPI es un framework moderno para la construcción de APIs REST rápidas y escalables en Python. Está diseñado para ser fácil de usar, rápido de desarrollar y altamente eficiente en términos de rendimiento. FastAPI utiliza Python 3.6+ y aprovecha las características de tipado estático de Python para proporcionar una API autodocumentada y con validación de tipos integrada [32].

2.4.2. MongoDB

MongoDB es una base de datos NoSQL (del inglés, *Not Only SQL*) de código abierto y orientada a documentos que proporciona una forma flexible y escalable de almacenar y recuperar datos. Utiliza un modelo de datos basado en documentos que almacena datos en un formato similar a JSON (del inglés, *JavaScript Object Notation*) llamado BSON (del inglés, *Binary JSON*) que permite almacenar datos de forma anidada y sin esquema fijo, lo que facilita la manipulación y consulta de datos no estructurados [33].

¹¹Imagen tomada de [Relé de 2 Canales Amazon](#).

2.5. Desarrollo frontend

En esta sección se presentan las herramientas de software utilizadas en el desarrollo del frontend.

2.5.1. React

React es una biblioteca de JavaScript de código abierto para construir interfaces de usuario interactivas y reutilizables. Desarrollada por Facebook, React permite crear componentes de interfaz de usuario que se actualizan de forma eficiente cuando cambian los datos, lo que facilita la creación de aplicaciones web rápidas y dinámicas [34].

2.6. Infraestructura y despliegue

En esta sección se presentan las herramientas de software utilizadas en la infraestructura y despliegue del sistema.

2.6.1. Docker

Docker es una plataforma de código abierto que permite a los desarrolladores **y a los equipos de operaciones** construir, empaquetar y desplegar aplicaciones en contenedores. Los **contenedores** son unidades de software ligeros y portátiles que incluyen todo lo necesario para ejecutar una aplicación, incluidas las bibliotecas, las dependencias y el **código** [35].

Docker facilita la creación de entornos de desarrollo y despliegue consistentes y reproducibles, lo que garantiza que las aplicaciones se ejecuten de la misma manera en cualquier entorno.

2.6.2. AWS IoT Core

AWS IoT Core es un servicio de AWS (del inglés, *Amazon Web Services*) que permite a los dispositivos conectarse de forma segura a la nube y comunicarse entre sí a través de protocolos de comunicación estándar como MQTT y HTTP. Proporciona una infraestructura escalable y segura para la gestión de dispositivos, la recopilación de datos y la integración con otros servicios de AWS [36]. Utiliza TLS para cifrar la comunicación entre los dispositivos y la nube, para garantizar la confidencialidad y la integridad de los datos.

2.6.3. AWS EC2

Amazon EC2 (del inglés, *Elastic Compute Cloud*) es un servicio de AWS que proporciona capacidad informática escalable en la nube. Permite a los usuarios lanzar instancias virtuales en la nube con diferentes configuraciones de CPU, memoria, almacenamiento y red, lo que facilita la implementación de aplicaciones escalables y de alta disponibilidad [37].

2.5. Desarrollo frontend

En esta sección se presentan las herramientas de software utilizadas en el desarrollo del frontend.

2.5.1. React

React es una biblioteca de JavaScript de código abierto para construir interfaces de usuario interactivas y reutilizables. Desarrollada por Facebook, React permite crear componentes de interfaz de usuario que se actualizan de forma eficiente cuando cambian los datos, lo que facilita la creación de aplicaciones web rápidas y dinámicas [34].

2.6. Infraestructura y despliegue

En esta sección se presentan las herramientas de software utilizadas en la infraestructura y despliegue del sistema.

2.6.1. Docker

Docker es una plataforma de código abierto que permite a los desarrolladores construir, empaquetar y desplegar aplicaciones en contenedores. Los **contenedores** son unidades de software ligeros y portátiles que incluyen todo lo necesario para ejecutar una aplicación, incluidas las bibliotecas, las dependencias y el **código** [35].

Docker facilita la creación de entornos de desarrollo y despliegue consistentes y reproducibles, lo que garantiza que las aplicaciones se ejecuten de la misma manera en cualquier entorno.

2.6.2. AWS IoT Core

AWS IoT Core es un servicio de AWS (del inglés, *Amazon Web Services*) que permite a los dispositivos conectarse de forma segura a la nube y comunicarse entre sí a través de protocolos de comunicación estándar como MQTT y HTTP. Proporciona una infraestructura escalable y segura para la gestión de dispositivos, la recopilación de datos y la integración con otros servicios de AWS [36]. Utiliza TLS para cifrar la comunicación entre los dispositivos y la nube, para garantizar la confidencialidad y la integridad de los datos.

2.6.3. AWS EC2

Amazon EC2 (del inglés, *Elastic Compute Cloud*) es un servicio de AWS que proporciona capacidad informática escalable en la nube. Permite a los usuarios lanzar instancias virtuales en la nube con diferentes configuraciones de CPU, memoria, almacenamiento y red, lo que facilita la implementación de aplicaciones escalables y de alta disponibilidad [37].

2.7. Herramientas de desarrollo

En esta sección se presentan las herramientas de software utilizadas en el desarrollo del sistema.

2.7.1. Visual Studio Code

Visual Studio Code, comúnmente abreviado como VS Code, es un entorno de desarrollo integrado (IDE, del inglés, *Integrated Development Environment*) de código abierto, altamente extensible y multiplataforma compatible con Windows, macOS y Linux [38].

VS Code es un editor de código ligero y rápido con soporte para muchos lenguajes de programación y extensiones que permiten personalizar y mejorar la funcionalidad del editor. Además, cuenta con herramientas de depuración integradas, control de versiones y terminal integrada.

2.7.2. Postman

Postman es una plataforma de colaboración para el desarrollo de APIs que permite a los desarrolladores diseñar, probar y documentar de forma rápida. Proporciona una interfaz gráfica intuitiva para enviar solicitudes HTTP a un servidor y visualizar las respuestas, lo que facilita la depuración y el desarrollo de APIs [39].

2.7.3. GitHub

GitHub es una plataforma de alojamiento de repositorios Git [40] que permite a los desarrolladores colaborar en proyectos de software de forma distribuida. Proporciona herramientas para gestionar el código fuente, realizar seguimiento de los cambios, revisar el código, realizar integración continua y despliegue automático [41].

2.7. Herramientas de desarrollo

En esta sección se presentan las herramientas de software utilizadas en el desarrollo del sistema.

2.7.1. Visual Studio Code

Visual Studio Code, comúnmente abreviado como VS Code, es un entorno de desarrollo integrado (IDE, del inglés, *Integrated Development Environment*) de código abierto, altamente extensible y multiplataforma compatible con Windows, macOS y Linux [38].

VS Code es un editor de código ligero y rápido con soporte para muchos lenguajes de programación y extensiones que permiten personalizar y mejorar la funcionalidad del editor. Además, cuenta con herramientas de depuración integradas, control de versiones y terminal integrada.

2.7.2. Postman

Postman es una plataforma de colaboración para el desarrollo de APIs que permite a los desarrolladores diseñar, probar y documentar de forma rápida. Proporciona una interfaz gráfica intuitiva para enviar solicitudes HTTP y WebSocket a un servidor y visualizar las respuestas, lo que facilita la depuración y el desarrollo de APIs [39].

2.7.3. GitHub

GitHub es una plataforma de alojamiento de repositorios Git [40] que permite a los desarrolladores colaborar en proyectos de software de forma distribuida. Proporciona herramientas para gestionar el código fuente, realizar seguimiento de los cambios, revisar el código, realizar integración continua y despliegue automático [41].

Capítulo 3

Diseño e implementación

En este capítulo se describe el diseño y la implementación del sistema de monitoreo y control de invernaderos. Se detallan los componentes principales del sistema, las decisiones de diseño tomadas y los pasos seguidos para su implementación.

3.1. Arquitectura del sistema

La figura 3.1 ilustra la arquitectura general del sistema y la interacción entre los diferentes componentes.

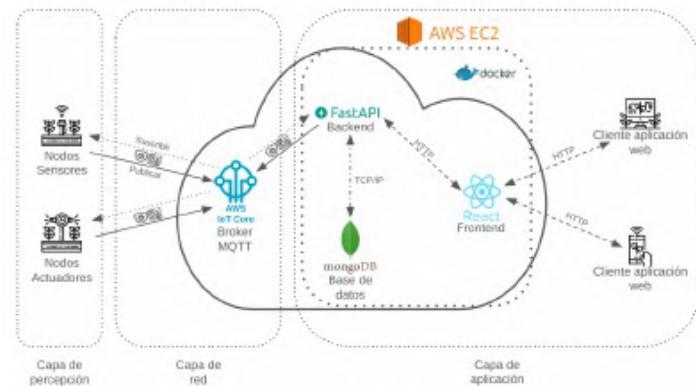


FIGURA 3.1. Arquitectura de la solución propuesta.

La arquitectura planteada para el desarrollo del trabajo sigue el modelo de tres capas típico de un sistema IoT: percepción, red y aplicación.

3.1.1. Capa de percepción

La capa de percepción está constituida por los nodos sensores y actuadores, que se encargan de recopilar datos del entorno y ejecutar acciones específicas en función de los parámetros configurados.

Capítulo 3

Diseño e implementación

Este capítulo describe el diseño y la implementación del sistema de monitoreo y control de invernaderos. Se detallan los componentes principales del sistema, las decisiones de diseño tomadas y los pasos seguidos para su implementación.

3.1. Arquitectura del sistema

La figura 3.1 ilustra la arquitectura general del sistema y la interacción entre los diferentes componentes.

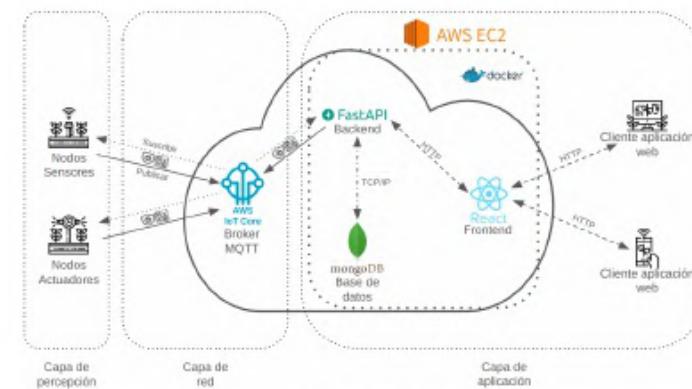


FIGURA 3.1. Arquitectura de la solución propuesta.

La arquitectura planteada para el desarrollo del trabajo sigue el modelo de tres capas típico de un sistema IoT: percepción, red y aplicación.

3.1.1. Capa de percepción

La capa de percepción está constituida por los nodos sensores y actuadores, que se encargan de recopilar datos del entorno y ejecutar acciones específicas en función de los parámetros configurados.

Cada nodo sensor incluye un microcontrolador ESP-WROOM-32, este se conecta a diversos sensores que miden parámetros como temperatura ambiente, humedad relativa, presión atmosférica, luminosidad, concentración de CO_2 , pH, conductividad eléctrica, temperatura de la solución nutritiva, nivel de líquidos, consumo eléctrico, entre otros. Los nodos actuadores, por su parte, cuentan con relés para controlar dispositivos como ventiladores, iluminación y sistemas de recirculación de nutrientes.

Los nodos **están** conectados a una red Wi-Fi local, lo que les **permite** establecer comunicación con la red y transmitir los datos **de los sensores** hacia el servidor IoT. La transmisión de datos se realiza **con** el protocolo MQTT.

3.1.2. Capa de red

La capa de red está **compuesta** por la infraestructura que gestiona la comunicación entre los nodos sensores y actuadores y la plataforma de backend. Los nodos sensores y actuadores se conectan a la red Wi-Fi local, lo que les permite acceder a internet y a la infraestructura de la nube. Una vez conectados, los dispositivos transmiten los datos a través del protocolo MQTT.

La comunicación entre los nodos y el broker MQTT se asegura mediante el uso de certificados de seguridad, los que garantizan la autenticación de los dispositivos y el cifrado de los datos.

El broker MQTT utilizado en este trabajo es AWS IoT Core, un servicio completamente gestionado que permite establecer una conexión segura y escalable entre los dispositivos IoT y la nube. Este broker actúa como intermediario para la transmisión de datos entre los nodos y la capa de aplicación.

3.1.3. Capa de aplicación

La capa de aplicación es responsable del procesamiento, almacenamiento y visualización de los datos recopilados por los nodos. Para esta capa, se implementó el servidor IoT en la nube **utilizando el servicio AWS EC2**, que permite ejecutar aplicaciones y servicios en instancias virtuales.

El procesamiento **y la gestión** de datos se realiza a través de un backend **desarrollado** con FastAPI, mientras que la base de datos MongoDB se utiliza para el almacenamiento de la información. Además, se implementó una interfaz gráfica de usuario en React para la visualización y gestión de los datos. Todos estos servicios fueron desplegados a través de contenedores Docker.

3.2. Modelo de datos

En esta sección se presenta el modelo de datos implementado en el sistema.

La figura 3.2 permite visualizar las principales colecciones y sus relaciones dentro de la base de datos. El diseño del modelo de datos se desarrolló en base a los tipos de datos proporcionados por los sensores, así como los **requerimientos técnicos establecidos** para el sistema.

Cada nodo sensor incluye un microcontrolador ESP-WROOM-32, este se conecta a diversos sensores que miden parámetros como temperatura ambiente, humedad relativa, presión atmosférica, luminosidad, concentración de CO_2 , pH, conductividad eléctrica, temperatura de la solución nutritiva, nivel de líquidos, consumo eléctrico, entre otros. Los nodos actuadores, por su parte, cuentan con relés para controlar dispositivos como ventiladores, iluminación y sistemas de recirculación de nutrientes.

Los nodos **fueron** conectados a una red Wi-Fi local, lo que les **permittió** establecer comunicación con la red y transmitir los datos hacia el servidor IoT mediante el protocolo MQTT.

3.1.2. Capa de red

La capa de red está **conformada** por la infraestructura que gestiona la comunicación entre los nodos del sistema (sensores y actuadores) y la plataforma **en** la nube. Una vez **integrados** a la red local, los nodos emplearon el protocolo MQTT para publicar y recibir datos de manera eficiente.

El broker MQTT utilizado en este trabajo es AWS IoT Core, un servicio completamente gestionado que permite establecer una conexión segura y escalable entre los dispositivos IoT y la nube. Este broker actúa como intermediario para la transmisión de datos entre los nodos y la capa de aplicación.

La comunicación entre los nodos y el broker MQTT se aseguró mediante el uso de certificados generados en el propio broker, los cuales garantizaron la autenticación de los dispositivos y el cifrado de los datos.

3.1.3. Capa de aplicación

La capa de aplicación es responsable del procesamiento, almacenamiento y visualización de los datos recopilados por los nodos. Para esta capa, se implementó el servidor IoT en la nube **a través** del servicio AWS EC2, que permite ejecutar aplicaciones y servicios en instancias virtuales.

El procesamiento de los datos se llevó a cabo mediante un backend **desarrollado** con FastAPI, **encargado** de gestionar las solicitudes e interactuar con la base de datos MongoDB, utilizada para el almacenamiento de la información.

Además, se **desarrolló** una interfaz de usuario en React que permite la visualización y gestión de los datos de manera intuitiva y accesible desde cualquier dispositivo con conexión a internet.

Todos los servicios fueron desplegados **en** contenedores Docker y orquestados mediante Docker Compose [42], lo que permitió una gestión eficiente, modular y escalable de los distintos componentes del sistema.

3.2. Modelo de datos

En esta sección se presenta el modelo de datos implementado en el sistema.

La figura 3.2 permite visualizar las principales colecciones y sus relaciones dentro de la base de datos. El diseño del modelo de datos se desarrolló en base a los tipos

3.2. Modelo de datos

17

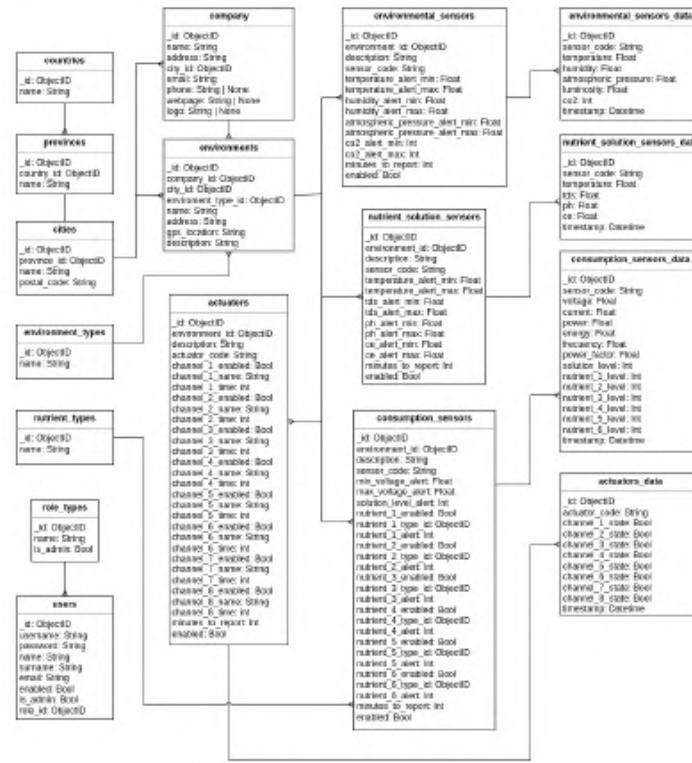


FIGURA 3.2. Modelo de datos implementado.

El modelo de datos se estructuró en colecciones dentro de MongoDB, organizadas en las siguientes categorías principales:

3.2.1. Parametrización del sistema

- Colecciones de configuración básica: países, provincias, ciudades, empresa.
- Colecciones para gestión de espacios: ambientes y tipos de ambientes.
- Colecciones específicas del dominio: tipos de nutrientes.

3.2.2. Gestión de usuarios y roles

- Colección de usuarios y roles: almacena información de los usuarios y sus credenciales.
- Colección de roles: se plantea como funcionalidad futura, para poder parametrizar permisos para cada tipo de rol.

3.2. Modelo de datos

17

de datos proporcionados por los sensores, así como los requerimientos técnicos establecidos para el sistema.

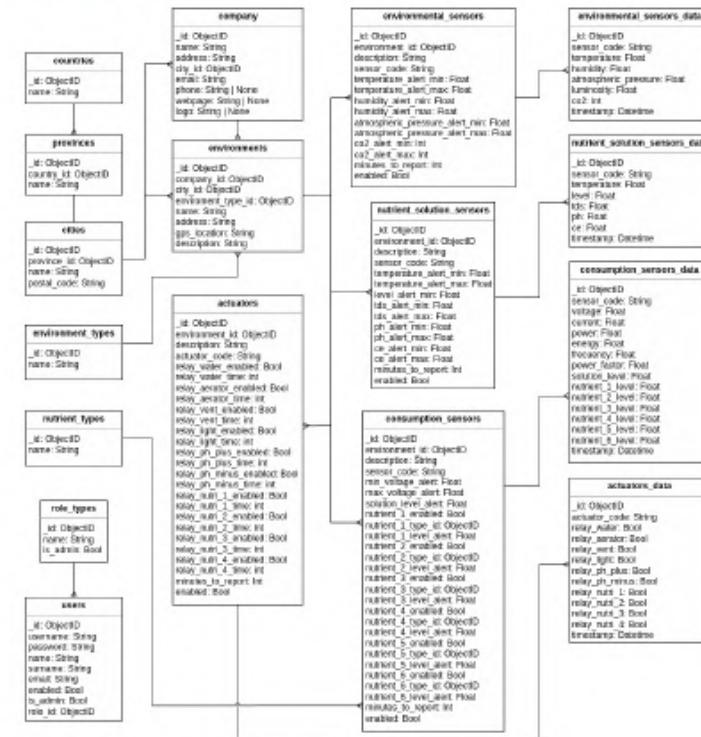


FIGURA 3.2. Modelo de datos implementado.

El modelo de datos se estructuró en colecciones dentro de MongoDB, organizadas en las siguientes categorías principales:

3.2.1. Parametrización del sistema

- Colecciones de configuración básica: países, provincias, ciudades, empresa.
- Colecciones para gestión de espacios: ambientes y tipos de ambientes.
- Colecciones específicas del dominio: tipos de nutrientes.

3.2.2. Gestión de usuarios y roles

- Colección de usuarios: almacena información de los usuarios y sus credenciales.

3.2.3. Sensores y actuadores

- Colecciones de sensores y actuadores: almacena la información de cada tipo de dispositivo, los parámetros de alerta y la frecuencia de muestreo.
- Colecciones de datos históricos: registran las mediciones vinculadas a cada dispositivo mediante identificadores únicos.

3.2.4. Auditoría y seguimiento

- Colecciones de logs: permiten registrar los cambios en las configuraciones de sensores y actuadores realizados por los usuarios.

El modelo de datos completo del sistema, puede consultarse en el apéndice A.

3.3. Servidor IoT

En esta sección se presenta la arquitectura del sistema y se detallan las tecnologías utilizadas y la arquitectura del servidor.

3.3.1. Arquitectura del servidor

La arquitectura del servidor está compuesta por tres componentes principales: backend, capa de datos y frontend, como se muestra en la figura 3.3.

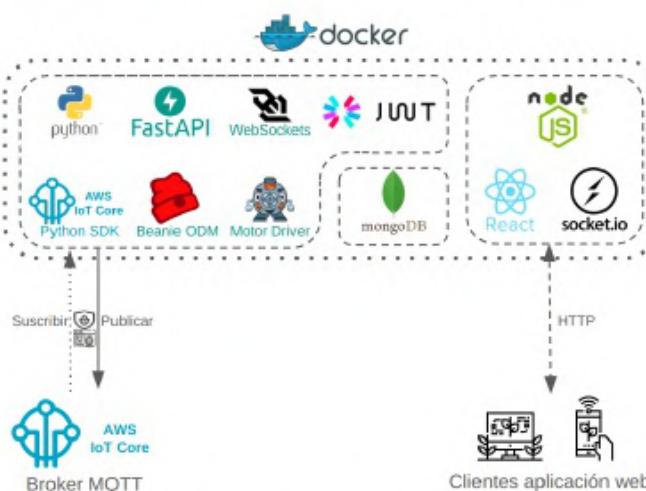


FIGURA 3.3. Arquitectura del servidor del sistema IoT.

A continuación, se describe brevemente cada uno de estos componentes:

- Backend: implementado con FastAPI, expone una API REST que permite gestionar sensores, actuadores, ambientes y usuarios. Incluye autenticación

- Colección de roles: se plantea como futura funcionalidad, para poder parametrizar permisos para cada tipo de rol.

3.2.3. Sensores y actuadores

- Colecciones de sensores y actuadores: almacena la información de cada tipo de dispositivo, los parámetros de alerta y la frecuencia de muestreo.
- Colecciones de datos históricos: registran las mediciones vinculadas a cada dispositivo mediante identificadores únicos.

3.2.4. Auditoría y seguimiento

- Colecciones de logs: permiten registrar los cambios en las configuraciones de sensores y actuadores realizados por los usuarios.

El modelo de datos completo del sistema, puede consultarse en el apéndice A.

3.3. Servidor IoT

En esta sección se presenta la arquitectura del sistema y se detallan las tecnologías utilizadas y la arquitectura del servidor.

3.3.1. Arquitectura del servidor

La arquitectura del servidor está compuesta por tres componentes principales: backend, almacenamiento y frontend, como se muestra en la figura 3.3.

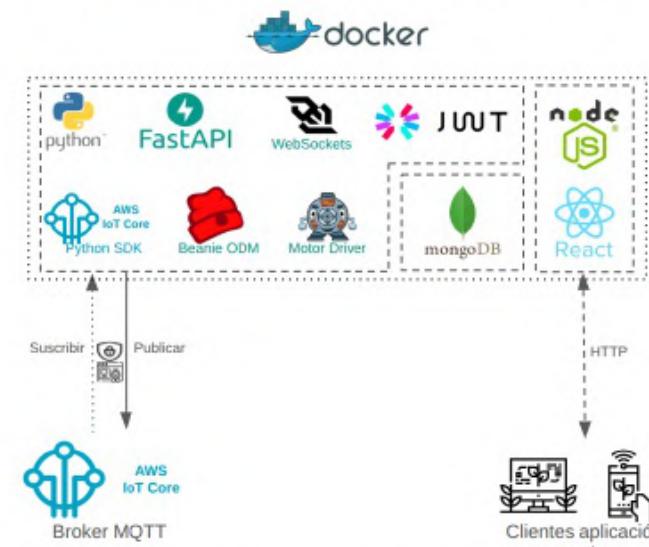


FIGURA 3.3. Arquitectura del servidor del sistema IoT.

3.4. Desarrollo del backend

19

y autorización basada en JWT, integración con MongoDB a través de Beanie [42] y Motor [43], conexión con el broker MQTT para la comunicación con los dispositivos IoT implementada con el SDK (del inglés, *Software Development Kit*) de AWS IoT para Python [44] y soporte para comunicaciones en tiempo real con clientes mediante WebSocket [45].

- **Capa de datos:** utiliza MongoDB como base de datos NoSQL. La información se organiza en colecciones que almacenan datos de sensores, actuadores, usuarios, ambientes y registros de configuración.
- Frontend: desarrollado en React, permite a los usuarios visualizar datos en tiempo real y configurar el sistema. Se conecta con la API REST del backend y utiliza WebSocket [46] para actualizaciones en tiempo real.

3.4. Desarrollo del backend

En esta sección se detallan los aspectos clave en el diseño y desarrollo del servidor backend, así como la lógica de negocio implementada.

3.4.1. Diseño de la API

El diseño se estructuró en base a las necesidades del sistema y los requerimientos funcionales y no funcionales establecidos. Se organizaron los archivos en carpetas de acuerdo a su funcionalidad.

La tabla 3.1 presenta un resumen de los principales endpoints de la API, junto con una breve descripción de la acción y el método HTTP utilizado.

TABLA 3.1. Resumen de principales endpoints de la API.

Método	Endpoint	Acción
GET	/mqqt/test	Test conexión cliente MQTT
POST	/mqqt/publish	Publicar en tópico MQTT
POST	/login	Login de usuarios
GET	/renew-token	Renovar token
GET	/users/	Obtener usuarios
GET	/environments/	Obtener ambientes
GET	/actuators/	Obtener actuadores
GET	/sensors/environmental/	Obtener sensores
GET	/sensors/nutrients/solution/	Obtener sensores
GET	/sensors/consumption/	Obtener sensores
GET	/actuators/data/	Obtener datos históricos
GET	/sensors/environmental/data/	Obtener datos históricos
GET	/sensors/consumption/data/	Obtener datos históricos
GET	/sensors/nutrients/solution/data/	Obtener datos históricos

El listado completo de endpoints de la API se puede consultar en el apéndice B.

3.4. Desarrollo del backend

19

A continuación, se describe brevemente cada uno de estos componentes:

- Backend: implementado en FastAPI, expone una API REST que permite gestionar sensores, actuadores, ambientes y usuarios. Incluye autenticación y autorización basada en JWT, integración con MongoDB a través de Beanie [43] y Motor [44], conexión con el broker MQTT implementada con el SDK (del inglés, *Software Development Kit*) de AWS IoT para Python [45] y soporte para comunicaciones en tiempo real con clientes mediante WebSocket [46].
- Almacenamiento: utiliza MongoDB como base de datos NoSQL. La información se organiza en colecciones para almacenar datos de sensores, actuadores, usuarios, ambientes y registros de configuración.
- Frontend: desarrollado en React, permite a los usuarios visualizar datos en tiempo real, reportes y configurar el sistema. Se conecta con la API REST del backend y utiliza WebSocket para actualizaciones en tiempo real.

3.4. Desarrollo del backend

En esta sección se detallan los aspectos clave en el diseño y desarrollo del servidor backend, así como la lógica de negocio implementada.

3.4.1. Diseño de la API

El diseño se estructuró en base a las necesidades del sistema y los requerimientos funcionales y no funcionales establecidos. Se organizaron los archivos en carpetas de acuerdo a su funcionalidad.

La tabla 3.1 presenta un resumen de los principales endpoints de la API, junto con una breve descripción de la acción y el método HTTP utilizado.

TABLA 3.1. Resumen de principales endpoints de la API.

Método	Endpoint	Acción
GET	/mqqt/test	Test conexión cliente MQTT.
POST	/mqqt/publish	Publicar en tópico MQTT.
POST	/login	Login de usuarios.
GET	/renew-token	Renovar token.
GET	/users/	Obtener usuarios.
GET	/environments/	Obtener ambientes.
GET	/actuators/	Obtener actuadores.
GET	/sensors/environmental/	Obtener sensores.
GET	/sensors/nutrients/solution/	Obtener sensores.
GET	/sensors/consumption/	Obtener sensores.
GET	/actuators/data/	Obtener datos históricos.
GET	/sensors/environmental/data/	Obtener datos históricos.
GET	/sensors/consumption/data/	Obtener datos históricos.
GET	/sensors/nutrients/solution/data/	Obtener datos históricos.

3.4.2. Autenticación y autorización

Se implementó un sistema de autenticación basado en JWT, que permite a los usuarios acceder a la API de manera segura. La autenticación se realiza mediante el envío de las credenciales del usuario en el cuerpo de la solicitud, y el servidor responde con un token JWT que se utiliza para autenticar las solicitudes posteriores.

El token JWT contiene la información del usuario, este token se envía en el **encabezado** de las solicitudes a la API. El servidor verifica la validez del token y permite o deniega el acceso a los recursos solicitados. El token se **diseñó** para que tuviera vencimiento, por lo que se implementó un **sistema** de renovación que permite a los usuarios mantener **su** sesión activa sin necesidad de autenticarse nuevamente con sus credenciales.

El código de la implementación de la autenticación y autorización se puede consultar en el apéndice C.

La figura 3.4 muestra el esquema de autenticación, autorización y renovación de tokens implementado en el sistema.

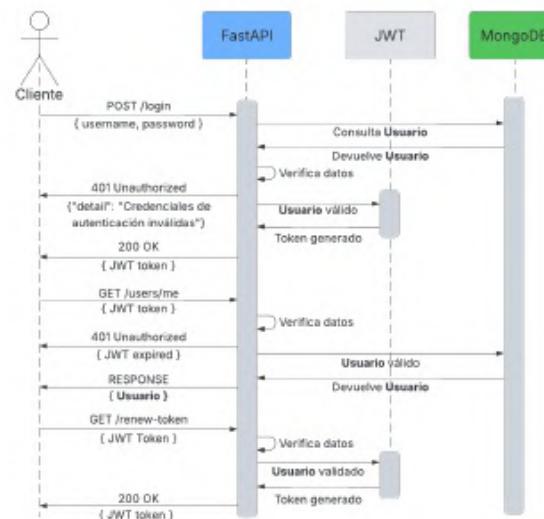


FIGURA 3.4. Esquema de autenticación y autorización.

3.4.3. Persistencia de datos

En FastAPI, cada modelo representa una colección en la base de datos e incluye los campos necesarios para almacenar la información requerida. La comunicación entre el backend y la base de datos se realizó a través de la biblioteca Motor, que proporciona una interfaz asíncrona para interactuar con MongoDB. Además se utilizó el ODM (del inglés, *Object Document Mapper*), a través de la biblioteca

El listado completo de endpoints de la API se puede consultar en el apéndice B.

3.4.2. Autenticación y autorización

Se implementó un sistema de autenticación basado en JWT, que permite a los usuarios acceder a la API de manera segura. La autenticación se realiza mediante el envío de las credenciales del usuario en el cuerpo de la solicitud, y el servidor responde con un token JWT que se utiliza para autenticar las solicitudes posteriores.

El token JWT contiene la información del **usuario** y se envía en el **encabezado** de las solicitudes a la API. El servidor verifica la validez del token y permite o deniega el acceso a los recursos solicitados. El **servidor** verifica su **valididad** y, en función de ello, permite o deniega el acceso a los recursos solicitados.

Dado que el token fue diseñado con un **tiempo de expiración**, se implementó un **mecanismo** de renovación que permite a los usuarios mantener la **sesión activa** sin necesidad de **volver** a autenticarse con sus credenciales.

La figura 3.4 muestra el esquema de autenticación, autorización y renovación de tokens implementado en el sistema.

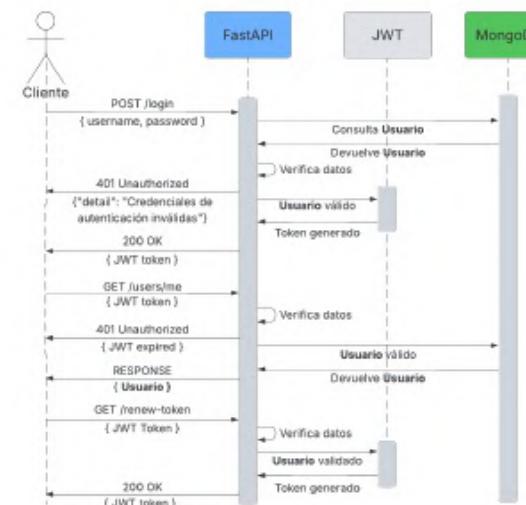


FIGURA 3.4. Esquema de autenticación y autorización.

3.4.3. Persistencia de datos

En FastAPI, cada modelo representa una colección en la base de datos e incluye los campos necesarios para almacenar la información requerida. La comunicación entre el backend y la base de datos se realizó a través de la biblioteca Motor, que proporciona una interfaz asíncrona para interactuar con MongoDB. Además se utilizó el ODM (del inglés, *Object Document Mapper*), a través de la biblioteca

3.4. Desarrollo del backend

21

Beanie, que permite definir modelos y realizar consultas y operaciones sobre la base de datos de manera sencilla.

La figura 3.5 muestra un ejemplo de la relación entre los modelos implementados en el sistema y los métodos HTTP de la colección EnvironmentalSensor.

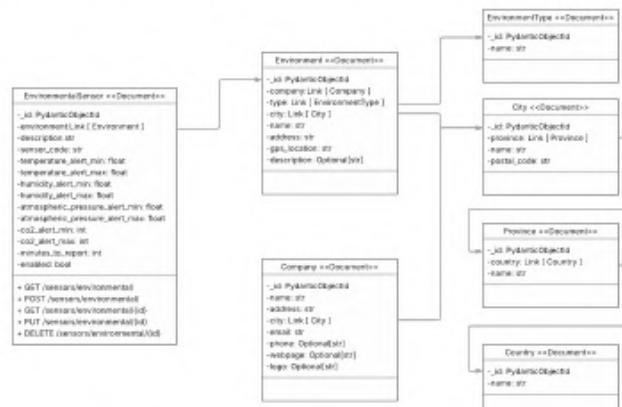


FIGURA 3.5. Diagrama de clases de los modelos implementados.

Como se mencionó anteriormente, los datos se almacenan en MongoDB. Para establecer la conexión con la base de datos, se utiliza el cliente asíncrono de la biblioteca Motor, mientras que la inicialización de los modelos se realiza mediante la función `init_beanie` del ODM Beanie. Esta función configura los modelos y establece la conexión con la base de datos. En la cadena de conexión se especifica el nombre de usuario, la contraseña y la dirección del servidor de MongoDB.

La figura 3.6 ilustra los pasos necesarios para establecer la conexión entre FastAPI y MongoDB.

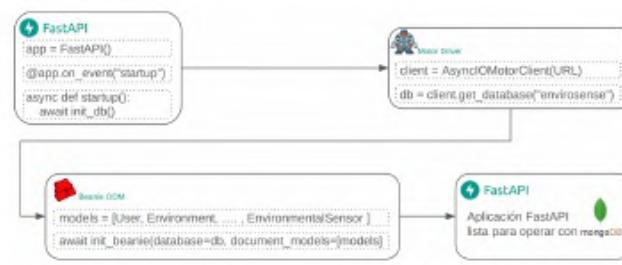


FIGURA 3.6. Pasos para la conexión de FastAPI y MongoDB

El código para establecer la conexión y la inicialización de los modelos se puede consultar en el apéndice D.

3.4. Desarrollo del backend

21

Beanie, que permite definir modelos y realizar consultas y operaciones sobre la base de datos de manera sencilla.

La figura 3.5 muestra un ejemplo de la relación entre los modelos implementados en el sistema y los métodos HTTP de la colección EnvironmentalSensor.

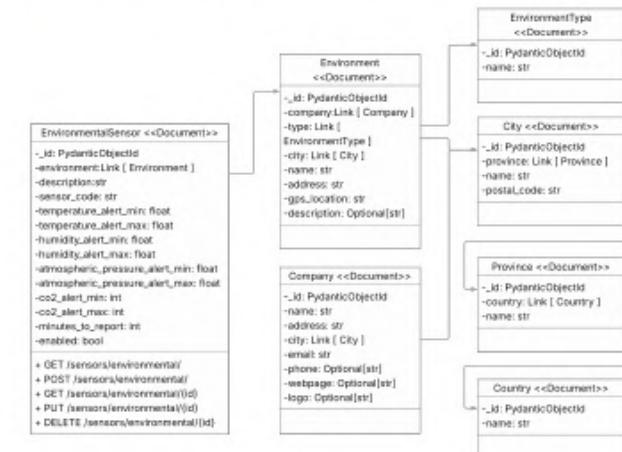


FIGURA 3.5. Diagrama de clases de los modelos implementados.

Como se mencionó anteriormente, los datos se almacenan en MongoDB. Para establecer la conexión con la base de datos, se utiliza el cliente asíncrono de la biblioteca Motor, mientras que la inicialización de los modelos se realiza mediante la función `init_beanie` del ODM Beanie. Esta función configura los modelos y establece la conexión con la base de datos. En la cadena de conexión se especifica el nombre de usuario, la contraseña y la dirección del servidor de MongoDB.

La figura 3.6 ilustra los pasos necesarios para establecer la conexión entre FastAPI y MongoDB.

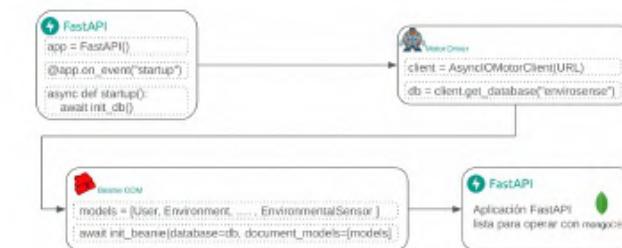


FIGURA 3.6. Pasos para la conexión de FastAPI y MongoDB

3.4.4. Comunicación con el broker MQTT

A continuación, se describe la implementación de la comunicación con el broker MQTT.

Pasos en AWS IoT Core

Este apartado detalla los pasos realizados en AWS IoT Core para crear y configurar el objeto *Thing* y los certificados de seguridad necesarios para establecer la conexión con el broker MQTT.

1. Se creó un objeto *Thing* en AWS IoT Core, que representa un dispositivo IoT. Este objeto se utiliza para gestionar la conexión y la comunicación con el broker.
2. Se generaron certificados de seguridad y claves privadas para el objeto *Thing*, y se descargó el certificado raíz de Amazon. Estos elementos son necesarios para autenticar la conexión, garantizar el cifrado de los datos transmitidos y establecer una comunicación segura con el broker MQTT.
3. Se definieron las políticas de acceso necesarias para el objeto *Thing*, a fin de permitir publicar y suscribirse a los tópicos correspondientes.

Políticas de acceso: determinan los permisos del cliente para publicar, suscribirse y recibir mensajes en los tópicos. Se aplican a los certificados generados para el cliente y controlan el acceso a los recursos de AWS IoT Core.

Definidas en formato JSON, estas políticas garantizan que solo los clientes autorizados puedan interactuar con los recursos y operar en los tópicos correspondientes. Además, pueden configurarse de manera específica para cada *Thing*, lo que permite ejercer un control granular sobre los permisos de acceso.

En el Apéndice E se puede visualizar un ejemplo de política de acceso definida para el objeto *Thing*.

Implementación de MQTT en FastAPI

Una vez que se creó y se configuró el objeto *Thing* en AWS IoT Core, se procedió a la implementación de la conexión del broker con FastAPI. Para ello, se utilizó la SDK de AWS IoT para Python, que proporciona una interfaz sencilla para conectarse al broker y gestionar la comunicación con los dispositivos IoT.

Se implementó un cliente MQTT que se conecta al broker con los certificados generados previamente. Este cliente permite publicar y suscribirse a los tópicos correspondientes, lo que facilitó la comunicación entre el servidor y los dispositivos IoT.

En la aplicación FastAPI se definieron dos rutas clave para la chequear la comunicación con AWS IoT Core:

1. Una ruta para verificar la conexión con el broker MQTT.
2. Una ruta para enviar mensajes a un tópico y comprobar la comunicación entre el servidor y el broker.

La figura 3.7 muestra los pasos realizados para verificar la conexión con el broker MQTT.

3.4.4. Comunicación con el Broker MQTT

A continuación, se detalla el proceso de implementación de la comunicación con el broker MQTT en AWS IoT Core.

Configuración del Thing y Certificados de Seguridad

Para establecer una conexión segura con el broker MQTT, se realizaron los siguientes pasos en AWS IoT Core:

1. Creación del Thing: se creó un objeto *Thing* en AWS IoT Core, que representa un dispositivo IoT. Este objeto se utiliza para gestionar la conexión y la comunicación con el broker.
2. Generación de certificados:
 - Se emitieron certificados de seguridad y claves privadas específicas para el *Thing*.
 - Se descargó el certificado raíz de Amazon (CA) para validar la autenticidad del broker.
- Estos elementos permiten:
 - Autenticación mutua entre dispositivo y broker.
 - Cifrado de extremo a extremo para la comunicación.
3. Asignación de políticas: se definieron las políticas de acceso necesarias para el objeto *Thing*, a fin de permitir publicar y suscribirse a los tópicos correspondientes.

Gestión de Políticas de Acceso

Las políticas de acceso en AWS IoT Core cumplen estas funciones clave:

- Control granular: definen permisos específicos (publicación/suscripción) para cada *Thing* mediante reglas JSON.
- Seguridad por diseño:
 - Restringen operaciones a tópicos autorizados.
 - Pueden limitarse por dispositivo, usuario o tipo de operación.
- Escalabilidad: permiten gestionar flotas de dispositivos mediante plantillas de políticas reutilizables.

Estas políticas garantizan que solo dispositivos autenticados con certificados válidos puedan intercambiar datos a través del broker MQTT.

Implementación de MQTT en FastAPI

Una vez que se creó y se configuró el objeto *Thing* en AWS IoT Core, se procedió a la implementación de la conexión del broker con FastAPI. Para ello, se utilizó la SDK de AWS IoT para Python, que proporciona una interfaz sencilla para conectarse al broker y gestionar la comunicación con los dispositivos IoT.

3.4. Desarrollo del backend

23

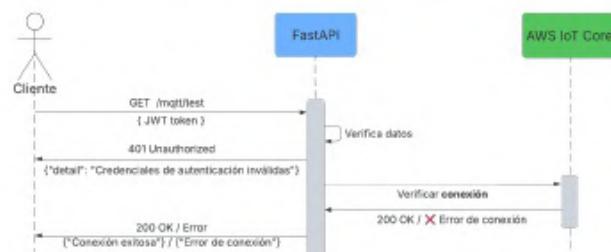


FIGURA 3.7. Pasos para verificar la conexión con el broker MQTT.

La figura 3.8 muestra los pasos realizados para publicar un mensaje en un tópico específico.

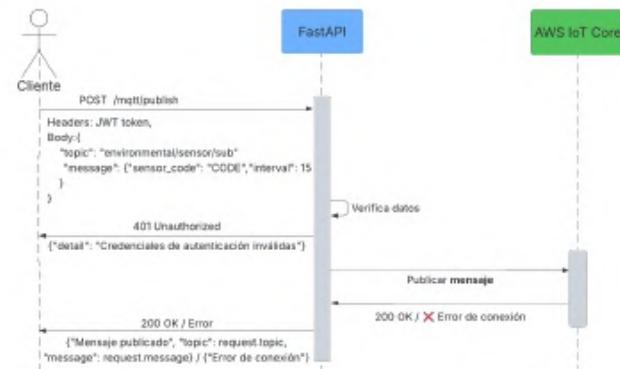


FIGURA 3.8. Pasos para publicar un mensaje en un tópico específico.

Comunicación con MQTT en FastAPI

Además de las rutas anteriores, en FastAPI se implementaron métodos para suscribirse a los tópicos, recibir mensajes de los nodos y publicar en los tópicos correspondientes.

Al iniciar el servidor, el cliente MQTT establece la conexión con el broker y se suscribe a los tópicos definidos. Los mensajes recibidos son procesados, almacenados en MongoDB y enviados al frontend mediante WebSocket, lo que permite actualizar la interfaz en tiempo real. Este cliente, que maneja la conexión, publicación y suscripción, se inicializa junto con FastAPI para asegurar una comunicación eficiente.

La figura 3.9 muestra los pasos de cómo la aplicación FastAPI se conecta al broker MQTT y se suscribe a los tópicos solicitados.

3.4. Desarrollo del backend

23

Se implementó un cliente MQTT que se conecta al broker con los certificados generados previamente. Este cliente permite publicar y suscribirse a los tópicos correspondientes, lo que facilitó la comunicación entre el servidor y los dispositivos IoT.

En la aplicación FastAPI se definieron dos rutas clave para la chequear la comunicación con AWS IoT Core:

1. Una ruta para verificar la conexión con el broker MQTT.
2. Una ruta para enviar mensajes a un tópico y comprobar la comunicación entre el servidor y el broker.

La figura 3.7 muestra los pasos realizados para verificar la conexión con el broker MQTT.

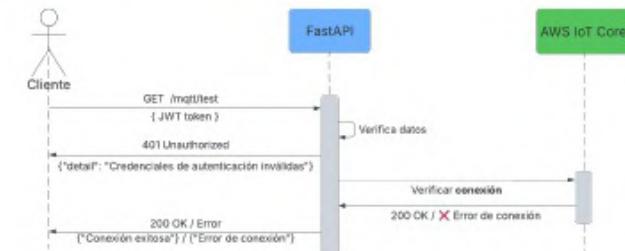


FIGURA 3.7. Pasos para verificar la conexión con el broker MQTT.

La figura 3.8 muestra los pasos realizados para publicar un mensaje en un tópico específico.

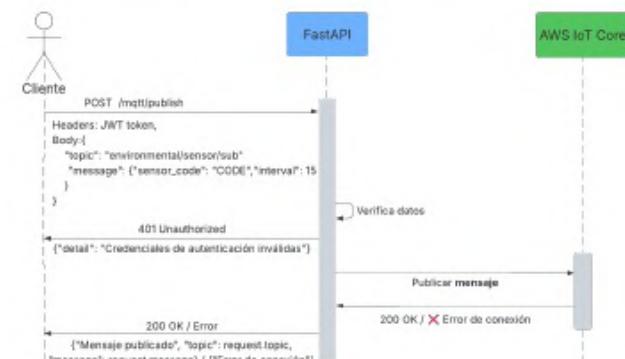


FIGURA 3.8. Pasos para publicar un mensaje en un tópico específico.



FIGURA 3.9. Pasos para la conexión del cliente MQTT.

El código completo de la implementación de la conexión con el broker MQTT se encuentra en el Apéndice E.

3.4.5. Implementación de WebSockets

3.5. Desarrollo del frontend

3.6. Desarrollo de nodos sensores y actuadores

3.7. Despliegue del sistema

Comunicación con MQTT en FastAPI

Además de las rutas anteriores, en FastAPI se implementaron métodos para suscribirse a los tópicos, recibir mensajes de los nodos y publicar en los tópicos correspondientes.

Al iniciarse el servidor, el cliente MQTT establece la conexión con el broker y se suscribe a los tópicos definidos. Los mensajes recibidos son procesados, almacenados en MongoDB y enviados al frontend mediante WebSocket, lo que permite actualizar la interfaz en tiempo real. Este cliente, que maneja la conexión, publicación y suscripción, se inicializa junto con FastAPI para asegurar una comunicación eficiente.

La figura 3.9 muestra los pasos de cómo la aplicación FastAPI se conecta al broker MQTT y se suscribe a los tópicos solicitados.



FIGURA 3.9. Pasos para la conexión del cliente MQTT.

3.4.5. Implementación de WebSocket

La comunicación en tiempo real se implementó mediante el módulo `websockets` de FastAPI, con una ruta específica para que los clientes se conecten y reciban datos actualizados de sensores y actuadores. La conexión permanece activa mientras sea válida, y se cierra en caso de error o falta de autorización.

El cliente debe enviar un token válido por `Authorization`. Si es válido, el servidor acepta la conexión y la gestiona a través de la clase `WebSocketManager`.

Esta clase administra las conexiones activas, clientes y el envío de datos en tiempo real. Además, mantiene un caché con los últimos valores por tipo de sensor para optimizar el rendimiento. Al recibir nuevos datos, el servidor actualiza el caché y los transmite a todos los clientes conectados.

La figura 4.4 ilustra el proceso de conexión.

Apéndice A

Modelo de datos implementado en el trabajo

La figura A.1 muestra el modelo de datos implementado en el trabajo.

3.5. Desarrollo del frontend

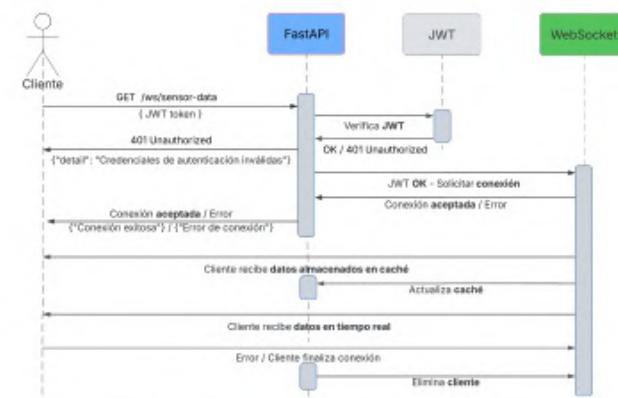


FIGURA 3.10. Pasos para establecer la conexión WebSocket.

3.5. Desarrollo del frontend

En esta sección se describe el diseño y desarrollo de la interfaz de usuario, enfocada en la visualización de datos en tiempo real y la gestión de dispositivos a través de una aplicación web.

3.5.1. Tecnologías utilizadas

Como se mencionó anteriormente, el frontend se desarrolló a través de la biblioteca React [34]. Para los estilos y la disposición visual se utilizó la biblioteca React Bootstrap [47]. Para los iconos de la interfaz se utilizó Bootstrap Icons [48], que proporciona una amplia variedad de iconos gratuitos personalizables y fáciles de usar.

La comunicación con el servidor se implementó mediante la biblioteca Axios [49] para realizar solicitudes HTTP, mientras que la recepción de datos en tiempo real se logró con la implementación de un WebSocket nativo, a través de un hook personalizado desarrollado en React.

La gestión del estado de autenticación y control de sesiones se implementó con Redux Toolkit [50] y la navegación entre páginas se gestionó a través de la biblioteca React Router [51].

Para la visualización de los gráficos, se empleó la librería Recharts [52], mientras que para la representación de tablas se utilizó la biblioteca React Data Table Component [53].

3.5.2. Arquitectura de la interfaz

La aplicación se estructuró en torno a un componente principal de layout, que organiza la navegación mediante un sidebar lateral y un navbar superior.

Apéndice A. Modelo de datos implementado en el trabajo

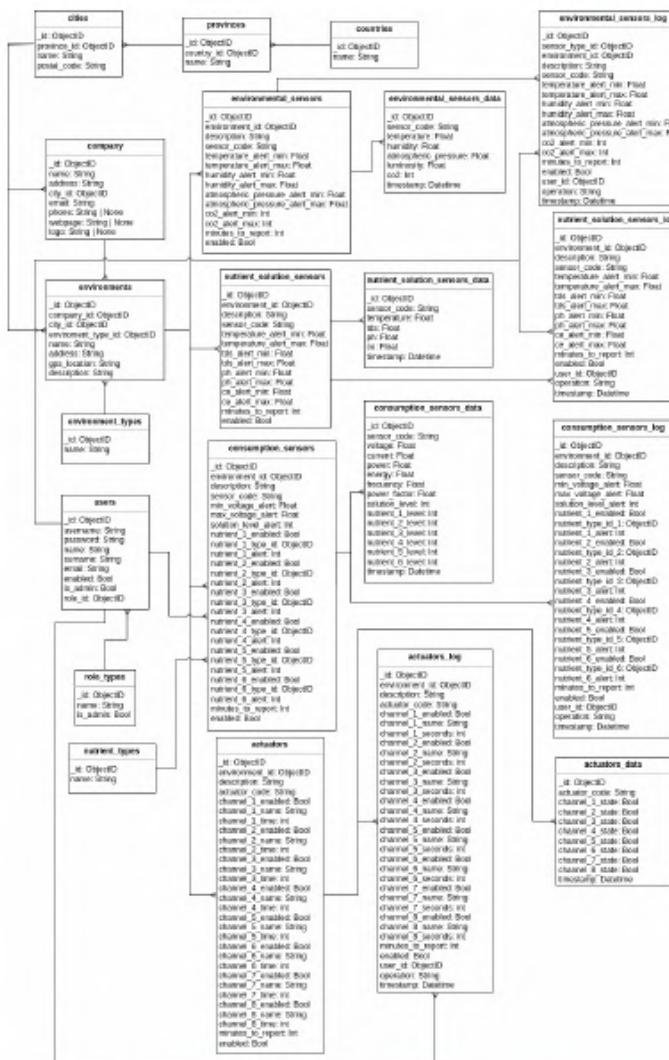


FIGURA A.1. Modelo de datos implementado en el trabajo.

Capítulo 3. Diseño e implementación

Cada sección de la interfaz de usuario corresponde a un módulo de funcionalidad específica (como visualización en tiempo real, reportes, configuración de ambientes, dispositivos, parámetros y administración de usuarios), que son renderizados dinámicamente en función de la ruta activa.

3.5.3. Componentes principales de la interfaz de usuario

A continuación, se presenta un detalle de los principales componentes desarrollados para la construcción de la interfaz.

Gestión de autenticación

La autenticación se implementó mediante un hook denominado `useAuthStore`, basado en Redux Toolkit. Este módulo gestiona el inicio de sesión, la validación del token y el cierre de sesión de los usuarios, para asegurar el acceso restringido a las funcionalidades de la plataforma.

Al iniciar sesión, se almacena el token en el `localStorage` del navegador, lo que permite mantener la sesión activa y acceder a las funcionalidades de la aplicación. El token se envía en cada solicitud a la API para autenticar al usuario y garantizar la seguridad de la comunicación.

La figura 4.6 muestra la pantalla de inicio de sesión, donde los usuarios ingresan sus credenciales para acceder a la aplicación web.



FIGURA 3.11. Pantalla de Login

Layout

El componente `layout` organiza la estructura general de la aplicación e incluye:

- **sidebar**: componente que muestra el menú lateral, con opciones de navegación entre los distintos módulos.
- **Navbar**: barra superior que permite colapsar o expandir el menú lateral y contiene el botón de cierre de sesión.

Apéndice B

Resumen de endpoints de la API

Las siguientes tablas presentan un resumen de los endpoints implementados en la API, junto con una breve descripción de la acción y el método HTTP utilizado.

TABLA B.1. Resumen de principales endpoints de la API

Método	Endpoint	Acción
GET	/api/	Ruta por defecto
GET	/mqtt/test	Test conexión cliente MQTT
POST	/mqtt/publish	Publicar en tópico MQTT
POST	/login	Login de usuarios
GET	/renew-token	Renovar token
GET	/roles/	Obtener roles
POST	/roles/	Crear rol
GET	/roles/{id}	Obtener un rol
PUT	/roles/{id}	Actualizar rol
DELETE	/roles/{id}	Eliminar rol
GET	/users/	Obtener usuarios
POST	/users/	Crear usuario
PUT	/users/	Actualizar usuario
GET	/users/{id}	Obtener un usuario
DELETE	/users/{id}	Eliminar usuario
PATCH	/users/	Actualizar username
PATCH	/users/password	Actualizar password
GET	/users/me	Obtener un usuario
PATCH	/users/change/password	Actualizar 'username'
GET	/countries/	Obtener países
POST	/countries/	Crear país
GET	/countries/{id}	Obtener un país
PUT	/countries/{id}	Actualizar país
DELETE	/countries/{id}	Eliminar país
GET	/provinces/	Obtener provincias
POST	/provinces/	Crear provincia
GET	/provinces/{id}	Obtener una provincia
PUT	/provinces/{id}	Actualizar provincia
DELETE	/provinces/{id}	Eliminar provincia

La figura 3.12 muestra la pantalla principal de la aplicación, donde se puede observar la barra de navegación superior y el menú lateral desde el rol admin.

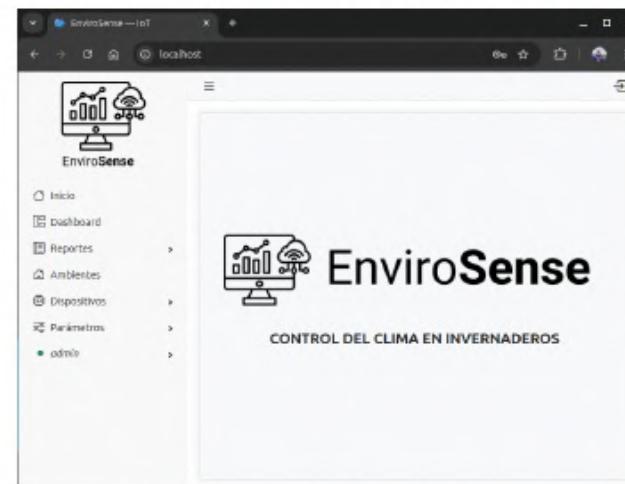


FIGURA 3.12. Estructura del componente layout

Además, esta pantalla permite visualizar los módulos disponibles en la aplicación, como dashboard, reportes, ambientes, dispositivos, parámetros y panel de usuario.

Arquitectura de navegación y control de acceso

La aplicación implementa un sistema de navegación que gestiona dinámicamente las rutas públicas y privadas mediante la biblioteca React Router. Utiliza los componentes `Routes` para definir la estructura de navegación y `Outlet` para renderizar sub-rutas.

Además, se implementa un flujo de acceso controlado por:

- Rutas públicas: accesibles sin autenticación, como la página de inicio de sesión.
- Rutas privadas: requieren autenticación y autorización, como el acceso a los módulos de la aplicación.
- Rutas restringidas: accesibles solo para usuarios con los permisos adecuados, como reportes, la gestión de usuarios, ambientes, y dispositivos.

Este esquema garantiza que cada usuario solo pueda acceder a las funcionalidades habilitadas según sus permisos. Por ejemplo, como se muestra en la figura 3.13, el usuario `martin`, con el rol de usuario estándar, tiene acceso únicamente al dashboard, a los reportes y a su panel de usuario, donde puede consultar su información personal.

TABLA B.2. Resumen de principales endpoints de la API

Método	Endpoint	Acción
GET	/cities/	Obtener ciudades
POST	/cities/	Crear ciudad
GET	/cities/{id}	Obtener una ciudad
PUT	/cities/{id}	Actualizar ciudad
DELETE	/cities/{id}	Eliminar ciudad
GET	/company/	Obtener empresas
POST	/company/	Crear empresa
GET	/company/{id}	Obtener una empresa
PUT	/company/{id}	Actualizar empresa
DELETE	/company/{id}	Eliminar empresa
POST	/company/uploadLogo	subir logo empresa
GET	/environments/types/	Obtener tipos de ambientes
POST	/environments/types/	Crear tipo de ambiente
GET	/environments/types/{id}	Obtener un tipo de ambiente
PUT	/environments/types/{id}	Actualizar tipo de ambiente
DELETE	/environments/types/{id}	Eliminar tipo de ambiente
GET	/environments/	Obtener ambientes
POST	/environments/	Crear ambiente
GET	/environments/{id}	Obtener un ambiente
PUT	/environments/{id}	Actualizar ambiente
DELETE	/environments/{id}	Eliminar ambiente
GET	/actuators/	Obtener actuadores
POST	/actuators/	Crear actuador
GET	/actuators/{id}	Obtener un actuador
PUT	/actuators/{id}	Actualizar actuador
DELETE	/actuators/{id}	Eliminar actuador
GET	/actuators/log/	Obtener logs de actuadores
POST	/actuators/log/	Crear log de actuador
GET	/actuators/data/	Obtener datos históricos
POST	/actuators/data/	Crear dato histórico
GET	/actuators/data/{id}	obtener un dato histórico
GET	/nutrients/types/	Obtener tipos de nutrientes
POST	/nutrients/types/	Crear tipo de nutriente
GET	/nutrients/types/{id}	Obtener un tipo de nutriente
PUT	/nutrients/types/{id}	Actualizar tipo de nutriente
DELETE	/nutrients/types/{id}	Eliminar tipo de nutriente
GET	/sensors/consumption/	Obtener sensores
POST	/sensors/consumption/	Crear sensor
GET	/sensors/consumption/{id}	Obtener un sensor
PUT	/sensors/consumption/{id}	Actualizar sensor
DELETE	/sensors/consumption/{id}	Eliminar sensor
GET	/sensors/consumption/log/	Obtener logs de sensor
POST	/sensors/consumption/log/	Crear log de sensor
GET	/sensors/consumption/data/	Obtener datos históricos
POST	/sensors/consumption/data/	Crear dato histórico
GET	/sensors/consumption/data/{id}	Obtener un dato histórico

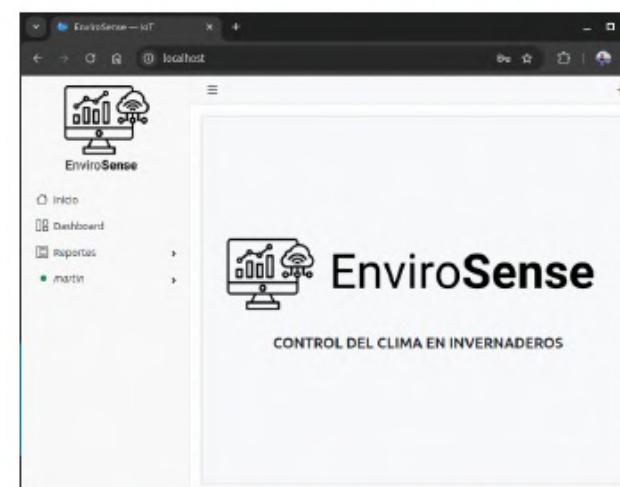


FIGURA 3.13. Interfaz de navegación según permisos de usuario estándar

3.5.4. Visualización de datos en tiempo real

Se desarrolló una página que permite visualizar en tiempo real los datos recopilados por los sensores y el estado de los actuadores de un ambiente específico. La información se actualiza automáticamente mediante el uso de WebSocket a medida que se reciben nuevos datos.

La figura 4.13 ilustra el dashboard de la aplicación, donde se pueden observar los datos de los sensores y actuadores en tiempo real de un ambiente específico.

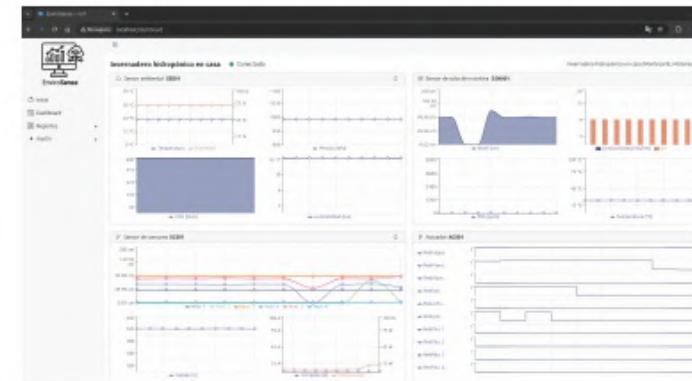


FIGURA 3.14. Visualización de datos en tiempo real

TABLA B.3. Resumen de principales endpoints de la API

Método	Endpoint	Acción
GET	/sensors/environmental/	Obtener sensores
POST	/sensors/environmental/	Crear sensor ambiental
GET	/sensors/environmental/{id}	Obtener un sensor
PUT	/sensors/environmental/{id}	Actualizar sensor
DELETE	/sensors/environmental/{id}	Eliminar sensor
GET	/sensors/environmental/log/	Obtener logs de sensor
POST	/sensors/environmental/log/	Crear log de sensor
GET	/sensors/environmental/data/	Obtener datos históricos
POST	/sensors/environmental/data/	Crear dato histórico
GET	/sensors/environmental/data/{id}	Obtener un dato histórico
GET	/sensors/nutrients/solution/	Obtener sensores
POST	/sensors/nutrients/solution/	Crear sensor
GET	/sensors/nutrients/solution/{id}	Obtener un sensor
PUT	/sensors/nutrients/solution/{id}	Actualizar sensor
DELETE	/sensors/nutrients/solution/{id}	Eliminar sensor
GET	/sensors/nutrients/solution/log/	Obtener logs de sensor
POST	/sensors/nutrients/solution/log/	Crear log de sensor
GET	/sensors/nutrients/solution/data/	Obtener datos históricos
POST	/sensors/nutrients/solution/data/	Crear dato histórico
GET	/sensors/nutrients/solution/data/{id}	Obtener un dato histórico

3.5.5. Reportes de datos históricos

Se desarrollaron un conjunto de reportes que permiten visualizar los datos históricos de los sensores y actuadores. Los diferentes tipos de reportes están organizados en módulos independientes.

Cada reporte permite:

- Filtrar los datos por dispositivo, rango de fechas y nivel de agregación temporal.
 - Actuadores: el reporte permite visualizar la cantidad de veces que se activó y el tiempo total de activación de cada canal del actuador.
 - Sensores: el reporte permite visualizar el promedio de los datos recolectados por cada sensor.
- Alternar entre una vista tabular y una vista gráfica.
- Página de resultados en la vista tabular, para mejorar la navegación cuando hay grandes volúmenes de datos.

Los reportes consumen los datos procesados a través de la API del backend, lo que permite mantener actualizada la información consultada.

La figura 3.15 presenta la pantalla de reportes, donde se visualiza la tabla con los datos históricos de un sensor de consumos.

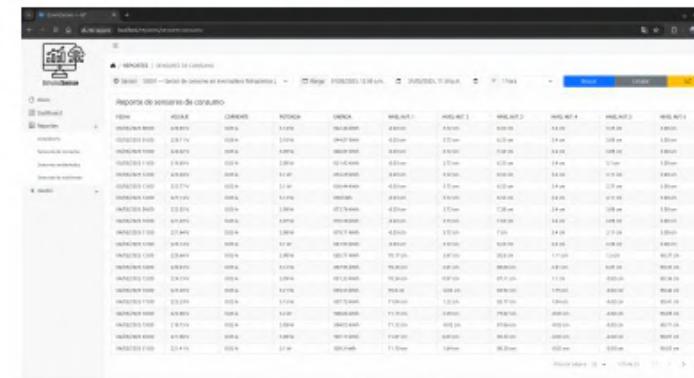


FIGURA 3.15. Pantalla de reportes

La figura 3.16 muestra la pantalla de reportes, donde se exhiben los gráficos con los datos históricos de un sensor de consumos.

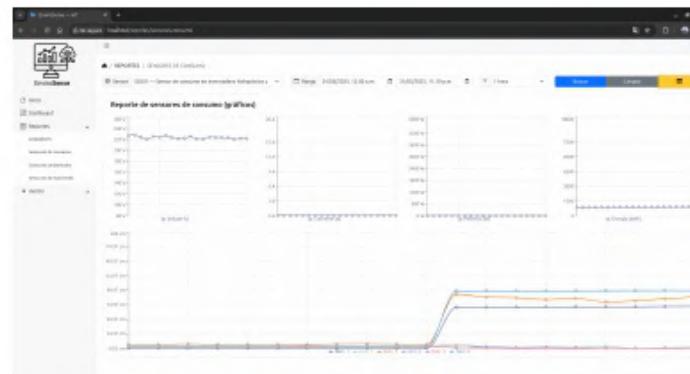


FIGURA 3.16. Pantalla de reporte

La figura 3.17 muestra la pantalla de reportes, donde se puede observar la tabla con los datos históricos de un actuador.

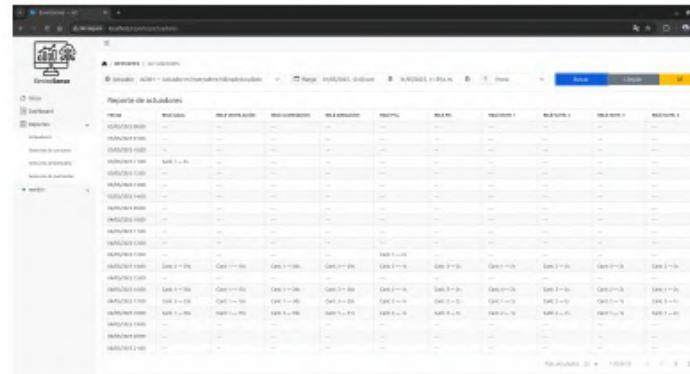


FIGURA 3.17. Pantalla de reporte

Finalmente, en la figura 3.18 se observa la interfaz de la aplicación web en un dispositivo móvil, específicamente la pantalla de visualización de datos en tiempo real. La interfaz se adapta de forma automática a diferentes tamaños de pantalla, lo que permite una experiencia de usuario optimizada tanto en teléfonos móviles como en tablets.

Apéndice C

Autenticación con JWT

El control de acceso se realiza mediante la verificación de un token JWT que se envía en las solicitudes. Si el token es válido, se permite el acceso a los recursos protegidos; de lo contrario, se devuelve un error de autenticación.

El código utiliza la librería `passlib` para el manejo de contraseñas y `bcrypt` para el cifrado. Además, se utiliza `fastapi.security` para manejar la autenticación y autorización.

El siguiente código es un ejemplo de cómo implementar el control de acceso en una API REST utilizando FastAPI y JWT.

```

1 from fastapi import APIRouter, Depends, HTTPException, status
2 from fastapi.security import OAuth2PasswordBearer
3 from passlib.context import CryptContext
4 import jwt
5 import bcrypt
6 from models.user import User
7 KEY = "colocar_clave_secreta_aquí"
8 ALG = "HS256"
9 ACCESS_TOKEN_EXPIRE_MINUTES = 30
10 oauth2 = OAuth2PasswordBearer(tokenUrl="login")
11 crypt = CryptContext(schemes=["bcrypt"], deprecated="auto")
12 async def auth_user(token: str = Depends(oauth2)):
13     exception = HTTPException(
14         status_code=status.HTTP_401_UNAUTHORIZED,
15         detail="Credenciales de autenticación inválidas",
16         headers={"WWW-Authenticate": "Bearer"}
17     )
18     try:
19         user = jwt.decode(token, KEY, algorithms=[ALG]).get("username")
20         if user is None:
21             raise exception
22     except:
23         raise exception
24     user = await User.find_one({"username": user})
25     if not user:
26         raise exception
27     return user
28 async def current_user(user: User = Depends(auth_user)):
29     if not user.enabled:
30         raise HTTPException(
31             status_code=status.HTTP_400_BAD_REQUEST, detail="Usuario
32 deshabilitado"
33         )
34     return user

```

CÓDIGO C.1. Pseudocódigo del control de acceso

3.6. Desarrollo del firmware de los dispositivos IoT

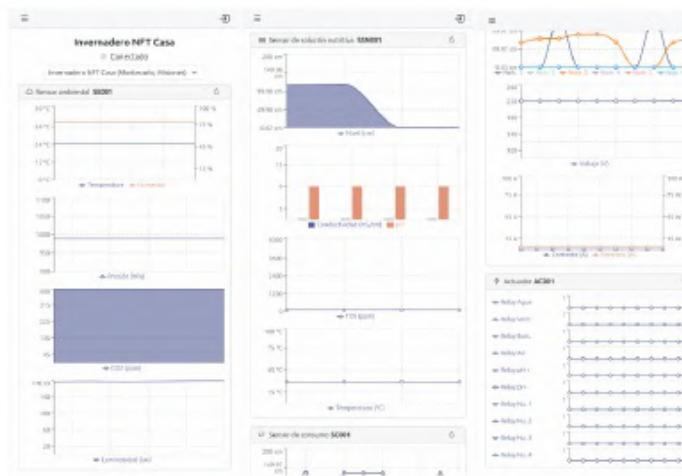


FIGURA 3.18. Visualización de datos en tiempo real desde un dispositivo móvil

3.6. Desarrollo del firmware de los dispositivos IoT

Esta sección presenta el desarrollo del firmware implementado en MicroPython para los nodos del sistema. Se describe la arquitectura general del software embutido y la estructura del código fuente.

3.6.1. Arquitectura general del firmware

Todos los nodos comparten una arquitectura de firmware similar, diseñada para realizar:

- Adquisición de datos: lectura de sensores y actuadores.
- Control de actuadores: gestión de dispositivos conectados.
- Comunicación inalámbrica: conexión a la red Wi-Fi y broker MQTT.
- Gestión de eventos: manejo de interrupciones y temporizadores.
- Interacción con el broker MQTT: publicación y suscripción a tópicos.

3.6.2. Estructura del código

Para el desarrollo de los nodos, se adoptó un esquema de organización que facilitó la modularidad, la reutilización de código y la facilidad de mantenimiento.

La estructura es la siguiente:

1. Importación de librerías: importación de módulos para sensores, Wi-Fi, MQTT, gestión del tiempo y otras funcionalidades.

2. Configuración inicial: definición de variables y constantes (pines, parámetros de conexión, intervalos).
3. Inicialización: configuración de periféricos del ESP32, inicialización de sensores/relés y manejo de errores.
4. Funciones auxiliares: definición de funciones para la conexión Wi-Fi, sincronización NTP (del inglés, *Network Time Protocol*) y carga de certificados TLS.
5. Conexión MQTT: establecimiento de la conexión con el broker MQTT con certificados TLS.
6. Callback de mensajes MQTT: función para procesar mensajes MQTT (configuración remota, acciones específicas).
7. Función de lectura/control: implementación de la lógica para leer sensores o controlar actuadores.
8. Bucle principal:
 - a) Verificación de la conexión Wi-Fi.
 - b) Lectura de datos de los sensores o control de los actuadores.
 - c) Envío de datos al broker MQTT.
 - d) Recepción y procesamiento de mensajes MQTT.
 - e) Gestión de errores y reinicio del sistema en caso de fallos.

La Figura 3.19 muestra el flujo general del firmware, desde la inicialización hasta el bucle principal, con manejo de comandos MQTT y acciones sobre sensores y actuadores.

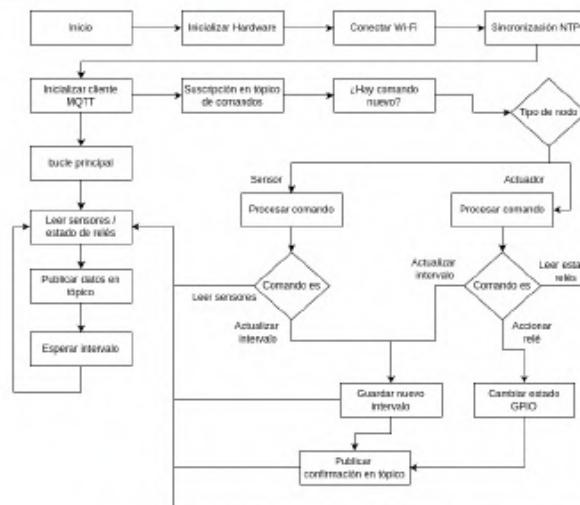


FIGURA 3.19. Flujo de ejecución del código en los nodos

Apéndice D

Conexión de la aplicación FastAPI con la base de datos MongoDB

La aplicación FastAPI se conecta con la base de datos MongoDB a través de la biblioteca Motor y el ODM Beanie. La conexión a la base de datos MongoDB se establece con la URL de conexión, que incluye el nombre de usuario, la contraseña y la dirección del servidor MongoDB. La URL de conexión se define en la configuración de la aplicación FastAPI y se utiliza para crear una instancia de Motor y Beanie.

El código D.1 ilustra la conexión a la base de datos MongoDB con Motor y Beanie.

```

1  from fastapi import FastAPI
2  from pymongo import MongoClient
3  from beanie import init_beanie
4  from motor.motor_asyncio import AsyncIOMotorClient
5
6  # Método asíncrono para inicializar la conexión a la base de datos
7  async def init_db():
8      client = AsyncIOMotorClient("mongodb://USER:PASSWORD@URL:PORT/?"
9      authSource=admin")
10     db = client.get_database("envirosense")
11     await init_beanie(database=db, document_models=[User, Role, Country, Province, City, Company,
12     EnvironmentType, Environment, NutrientType, ConsumptionSensor,
13     ConsumptionSensorData, ConsumptionSensorLog,
14     EnvironmentalSensor, EnvironmentalSensorData,
15     EnvironmentalSensorLog, NutrientSolutionSensor,
16     NutrientSolutionSensorData, NutrientSolutionSensorLog,
17     Actuator, ActuatorData, ActuatorLog])
18
19
20  # Iniciar FastAPI
21  app = FastAPI()
22
23  # Al inicializar la aplicación FastAPI, se ejecuta la función
24  startup
25  @app.on_event("startup")
26
27  # Método asíncrono que se ejecuta al iniciar la aplicación
28  # Se utiliza el método await para esperar a que la conexión se
29  # establezca antes de continuar con la ejecución de la aplicación.
30  await init_db()
```

CÓDIGO D.1. Cliente MQTT

3.6. Desarrollo del firmware de los dispositivos IoT

3.6.3. Gestión de la configuración

La configuración de los parámetros operativos de cada nodo se almacenó en archivos locales en el ESP32, lo que garantiza la persistencia de la configuración incluso después de reinicios.

A continuación, se detallan los archivos de configuración utilizados en el firmware de los nodos:

Archivo de configuración

El archivo config.py contiene constantes y variables de configuración específicas de cada nodo. Incluye:

- Códigos de identificación: código único que permite identificar de manera única a cada dispositivo.
- Tópicos MQTT: define los tópicos utilizados para la publicación de datos y la recepción de comandos. Cada nodo cuenta con un tópico de envío y otro de recepción.
- Parámetros de conexión AWS IoT Core: identificador del cliente y el endpoint del broker MQTT.

Archivo de configuración del intervalo

El archivo interval.conf almacena el intervalo de muestreo de sensores y actuadores. Al iniciar el dispositivo, se intenta leer este valor; si el archivo no existe o está corrupto, se usa un valor por defecto. El intervalo puede actualizarse remotamente mediante MQTT, se sobrescribe el archivo para conservar la nueva configuración tras reinicios.

Archivo de configuración Wi-Fi

El archivo wifi.dat almacena los parámetros de conexión Wi-Fi (SSID y contraseña), permite guardar múltiples configuraciones de red. Si el nodo no logra conectarse, se inicia un servidor web para que el usuario seleccione el SSID, ingrese la contraseña y defina la zona horaria. Al configurarse con éxito, los datos se guardan en el archivo y el dispositivo se reinicia.

Archivo de configuración de la zona horaria

La zona horaria se obtiene desde el archivo timezone.conf para garantizar la marca de tiempo correcta. Esta funcionalidad se utiliza en los nodos sensores y actuadores con el fin de incluir la hora local en los mensajes enviados al broker MQTT. En caso de error en la lectura del archivo, se asigna una zona horaria por defecto, en este caso, la de Argentina (UTC-3).

3.6.4. Comunicación inalámbrica

Los nodos desarrollados utilizan la comunicación Wi-Fi para conectarse a la red local y enviar datos al broker MQTT a través de Internet.

A continuación, se describen las librerías utilizadas para gestionar la comunicación inalámbrica:

Wi-Fi

Para gestionar la conexión Wi-Fi, se empleó la biblioteca Wifi Manager [54]. Al iniciar, los nodos intentan conectarse a la red configurada y realizan verificaciones periódicas para asegurar su disponibilidad.

La biblioteca se modificó para adaptarla a los requerimientos del trabajo. Se agregó la capacidad de almacenar datos de redes cuyo SSID contenga espacios en blanco y se implementó un menú desplegable para la selección de la zona horaria.

MQTT

Para desarrollar la comunicación MQTT, se utilizó la biblioteca `umqtt.robust` de MicroPythonLib [55], que permitió implementar un cliente MQTT en los nodos. Se estableció la conexión al broker MQTT de AWS IoT Core mediante certificados TLS almacenados localmente en cada dispositivo.

La Figura 3.20 ilustra el flujo de comunicación bidireccional entre los nodos, el broker MQTT de AWS IoT Core y el backend en FastAPI, mediante conexiones TLS y tópicos específicos. Los nodos envían datos y reciben comandos, mientras que el backend centraliza la gestión de las comunicaciones.

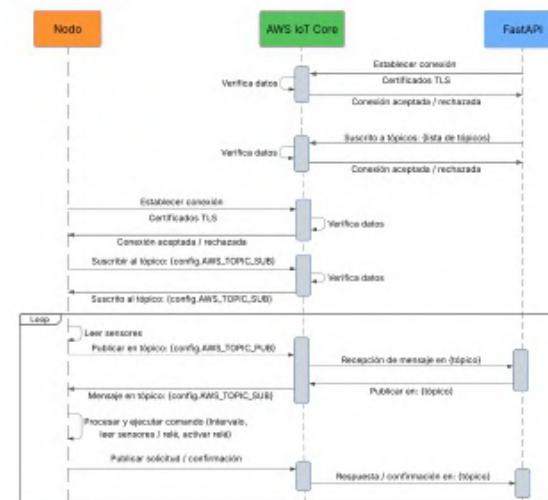


FIGURA 3.20. Flujo de comunicación entre nodos y broker MQTT

Comunicación y formato de mensajes

Los datos de sensores, actuadores y los comandos de control se transmiten mediante MQTT, en formato JSON, y se publican en los tópicos definidos en el archivo `config.py`. Se implementó un esquema estandarizado para simplificar el procesamiento tanto en los nodos como en el backend.

Apéndice E

Conexión al broker MQTT con FastAPI y AWS IoT SDK para Python

La conexión al broker MQTT se realiza para permitir la comunicación entre el servidor IoT y el broker MQTT.

El código E.1 muestra una política de acceso en AWS IoT Core que habilita al cliente a conectarse, publicar, suscribirse y recibir mensajes en cualquier tópico.

```

1
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Action": [
7                 "iot:Connect",
8                 "iot:Publish",
9                 "iot:Subscribe",
10                "iot:Receive"
11            ],
12            "Resource":
13                "arn:aws:iot:*:*:*"
14        }
15    ]

```

CÓDIGO E.1. Ejemplo de política de acceso en AWS IoT Core

El código E.2 muestra el proceso de conexión a AWS IoT Core y la implementación de la lógica de publicación y suscripción a los tópicos.

En este código, se definen los métodos para conectar al broker, publicar y suscribirse a tópicos, y manejar los mensajes recibidos. Se implementó un cliente MQTT que interactúa con AWS IoT Core, gestionando la comunicación con los nodos sensores y actuadores. Además, se incorporaron métodos para recibir datos de estos dispositivos, enviarles comandos y almacenar la información en la base de datos MongoDB.

```

1     import asyncio
2     import json
3     import os

```

3.6. Desarrollo del firmware de los dispositivos IoT

3.6.5. Gestión de tareas y concurrencia

La ejecución concurrente de tareas en MicroPython se logró mediante la implementación de:

- Un bucle principal para tareas como:
 - Conexión Wi-Fi.
 - Procesamiento de mensajes MQTT.
 - Lectura de datos de sensores.
- Temporizadores hardware (Timers) del ESP32 para:
 - Ejecutar acciones periódicas (por ejemplo: control de actuadores).
 - Procesar tareas en paralelo sin afectar el flujo principal.

3.6.6. Manejo de errores

Se implementaron mecanismos en todo el firmware para garantizar la robustez y confiabilidad de los nodos frente a posibles fallos. A continuación, se detallan las principales estrategias implementadas para la gestión de errores:

Manejo de excepciones

Se utilizaron bloques `try-except` para capturar y manejar excepciones durante la ejecución. De este modo, el programa puede continuar operativo incluso ante errores como fallos en la lectura de sensores o problemas de conexión a la red.

Reintentos

Para operaciones críticas, como la conexión Wi-Fi y la sincronización de la hora con el servidor NTP, se implementaron mecanismos de reinicio. Si una operación falla, el programa intenta repetirla un número preestablecido de veces antes de considerarla fallida.

Logging

Se implementaron mensajes de registro (`print`) para informar sobre el estado del programa y los errores que se producen. Esto facilitó la depuración y el mantenimiento del firmware.

Optimización de recursos

Se empleó la función `gc.collect()` para liberar memoria de forma periódica y optimizar el rendimiento del sistema. Esta función se invoca, por ejemplo, luego de la lectura de sensores o del envío de datos al broker MQTT.

3.6.7. Resumen de nodos sensores y actuadores

La tabla 3.2 resume las funciones y descripción general del firmware de los nodos sensores y actuadores, diseñados para adquirir datos del entorno o controlar dispositivos.

```

1 from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
2 from models.sensor_environmental_data import EnvironmentalSensorData
3 from mqtt.aws_config import AWS_ENDPOINT, MQTT_CLIENT_ID
4
5 # Método asíncrono para insertar datos en la base de datos
6 async def insert_sensor_data(payload):
7     try:
8         sensor_data = EnvironmentalSensorData(**payload)
9         await sensor_data.insert()
10        print("Datos guardados en la base de datos.")
11    except Exception as e:
12        print("Error al guardar datos en la base de datos:", e)
13
14 # Método para procesar mensajes de sensores y actuadores
15 async def process_sensor_message_pub(topic, payload):
16     try:
17         print(f"Mensaje recibido desde {topic}")
18         print(f"Data: {payload}")
19
20         if topic in [
21             "environmental/sensor/pub",
22             "nutrient_solution/sensor/pub",
23             "consumption/sensor/pub",
24             "actuators/pub",
25         ]:
26             print(f"Procesando mensaje de {topic}")
27             await insert_sensor_data(payload)
28         except Exception as e:
29             print(f"Error inesperado: {e}")
30
31 # Método para procesar mensajes de control de sensores
32 def process_sensor_message_sub(topic, payload):
33     try:
34         sensor_code = payload.get("sensor_code")
35         interval = payload.get("interval")
36
37         print(f"Mensaje recibido desde {topic}")
38         print(f"Sensor: {sensor_code}, Intervalo: {interval}")
39
40         if str(interval) == "OK":
41             print("Nuevo intervalo de tiempo establecido")
42         else:
43             print("Error al establecer el intervalo de tiempo")
44     except Exception as e:
45         print(f"Error inesperado: {e}")
46
47 class AWSMQTTClient:
48     def __init__(self):
49         CERTS_DIR = os.getenv("CERTS_DIR", "mqtt/certificates")
50         self.ROOT_CRT = os.path.join(CERTS_DIR, "root.crt")
51         self.CLIENT_CRT = os.path.join(CERTS_DIR, "client.crt")
52         self.CLIENT_KEY = os.path.join(CERTS_DIR, "client.key")
53
54         # Verificar existencia de certificados
55         for cert in [self.ROOT_CRT, self.CLIENT_CRT, self.CLIENT_KEY]:
56
57             if not os.path.exists(cert):
58                 raise FileNotFoundError(f"El archivo {cert} no fue"
59                                     f"encontrado.")
60
61         # Configurar cliente MQTT
62         self.client = AWSIoTMQTTClient(MQTT_CLIENT_ID)
63         self.client.configureEndpoint(AWS_ENDPOINT, 8883)

```

TABLA 3.2. Resumen de nodos: funciones y firmware.

Nodo	Funciones principales	Descripción del firmware
Sensor ambiental	Medición de temperatura, humedad y presión (BME280). Medición de luminosidad (BH1750). Medición de concentración de CO ₂ (MH-Z19C). Conexión Wi-Fi. Conexión broker MQTT. Configuración remota de intervalo. Indicador LED. Botón de reinicio.	Comunicación I ² C para BME280. UART para MH-Z19C. Medición inicial de CO ₂ . Envío de datos en JSON vía MQTT. Sincronización NTP. Reconexión Wi-Fi automática.
Sensor de consumos	Medición de voltaje, corriente y potencia (PZEM-004T). Medición de nivel de líquidos (HC-SR04). Conexión Wi-Fi. Conexión broker MQTT. Configuración remota de intervalo. Indicador LED. Botón de reinicio.	Uso de UART para PZEM-004T. GPIO para HC-SR04. Envío de datos en formato JSON vía MQTT. Sincronización NTP. Reconexión Wi-Fi automática.
Sensor de solución nutritiva	Medición de temperatura (DS18B20). Medición de pH, CE y TDS. Medición de nivel de líquidos (HC-SR04). Conexión Wi-Fi. Conexión broker MQTT. Configuración remota de intervalo. Indicador LED. Botón de reinicio.	Lectura OneWire para DS18B20. Medición de CE,TDS y pH. Promedio de muestras. Envío de datos en JSON vía MQTT. Sincronización NTP. Reconexión Wi-Fi automática.
Actuador	Control de relés para dispositivos (bombas, aireadores, luces, dosificadores). Ejecución simultánea de comandos. Conexión Wi-Fi. Conexión broker MQTT. Configuración remota. Indicador LED. Botón de reinicio.	Control de salidas GPIO. Recepción de comandos MQTT. Ejecución de temporizadores con Timers ESP32. Manejo de estados de relés en JSON. Sincronización NTP. Reconexión Wi-Fi automática.

```

43     self.client.configureCredentials(self.ROOT_CRT, self.
44         CLIENT_KEY, self.CLIENT_CRT)
45     self.client.configureConnectDisconnectTimeout(10)
46     self.loop = asyncio.get_event_loop()
47
48     def connect(self):
49         print("Estableciendo conexión con AWS IoT Core...")
50         self.client.connect()
51         print("Cliente MQTT Conectado")
52
53     def disconnect(self):
54         self.client.disconnect()
55         print("Cliente Desconectado")
56
56     def publish(self, topic, message):
57         payload = json.dumps(message) if isinstance(message, dict)
58     else str(message)
59         self.client.publish(topic, payload, 0)
60         print(f'Mensaje enviado a {topic}: {payload}')
61
62     def subscribe(self, topic, callback):
63         def wrapper(client, userdata, message):
64             try:
65                 payload_str = message.payload.decode()
66                 payload = json.loads(payload_str)
67                 if topic in [ "environmental/sensor/sub",
68                     "nutrient_solution/sensor/sub",
69                     "consumption/sensor/sub",
70                     "actuators/sub" ]:
71                     process_sensor_message_sub(topic, payload)
72                 else:
73                     self.loop.create_task(process_sensor_message_pub(
74                         message.topic, payload))
75             except json.JSONDecodeError:
76                 print("Error al decodificar el mensaje JSON.")
77             except Exception as e:
78                 print(f'Error inesperado en wrapper: {e}')
79
80         self.client.subscribe(topic, 1, wrapper)
81         print(f'Suscripto a {topic}')
82
83     def unsubscribe(self, topic):
84         self.client.unsubscribe(topic)
85         print(f'Desuscripto de {topic}')
86

```

CÓDIGO E.2. Cliente MQTT

El código E.3 muestra la integración del cliente MQTT con la aplicación en FastAPI. Se define un cliente MQTT y se suscribe a los tópicos de publicación de datos de sensores y actuadores y tópicos de envío de parámetros a sensores y actuadores. Además, se definen los endpoints para probar la conexión MQTT y publicar mensajes desde la aplicación FastAPI.

```

1 # Fragmento de código para la integración del cliente MQTT con FastAPI
2
3 # Inicializar cliente MQTT
4 mqtt_client = AWSMQTTClient()
5
6 @app.on_event("startup")
7 async def startup():
8

```

3.7. Despliegue del sistema

En esta sección se describe el proceso de implementación y configuración del sistema en un entorno de prueba basado en una instancia EC2 AWS. Se detalla la instalación de los componentes, los pasos para realizar la clonación y ejecución del sistema, así como la configuración del servicio para su ejecución automática.

3.7.1. Entorno de prueba

Creación de la instancia EC2

El sistema fue desplegado en una instancia EC2 de AWS con las siguientes características:

- Nombre: envirosense-app.
- Imagen de máquina de Amazon: Ubuntu Server 24.04 LTS (64-bit x86).
- Tipo de instancia: t2.micro (1 vCPU, 1 GB de RAM).
- Almacenamiento: 8 GB tipo gp3.
- Región: us-east-1.
- Grupo de seguridad: puertos abiertos para HTTP (80), HTTPS (443), SSH (22), WebSocket (8000) y acceso a la base de datos (27017).

El acceso a la instancia se realizó mediante SSH (del inglés, *Secure Shell*). Para ello, se generó un par de claves RSA (Rivest, Shamir y Adleman) y se descargó el archivo `envirosense-app-key.pem`, necesario para establecer la conexión.

Acceso remoto y configuración inicial

Una vez lanzada la instancia, se accedió a la misma mediante SSH y se procedió a actualizar los paquetes del sistema operativo.

A continuación, se instalaron las siguientes herramientas:

- Docker: para los contenedores de la aplicación.
- Docker Compose: para la orquestación de los contenedores.
- DuckDNS [56]: para la gestión de la IP pública de la instancia.

3.7.2. Instalación de componentes del sistema

Docker y Docker Compose

Como se mencionó en la sección 3.3.1, se utilizó Docker para generar tres contenedores:

- Backend: contenedor con la imagen base `python` que ejecuta la API desarrollada en FastAPI.
- Frontend: contenedor que utiliza imágenes base de `node` y `nginx` para ejecutar un servidor HTTP con NGINX [57] y servir la aplicación web desarrollada en React.

```

1  await init_db()
2  mqtt_client.connect()
3
4  # Suscripción a tópicos de publicación de sensores y actuadores
5  mqtt_client.subscribe("environmental/sensor/pub",
6  process_sensor_message_pub)
7  mqtt_client.subscribe("nutrient_solution/sensor/pub",
8  process_sensor_message_pub)
9  mqtt_client.subscribe("consumption/sensor/pub",
10 process_sensor_message_pub)
11 mqtt_client.subscribe("actuators/pub", process_sensor_message_pub)
12
13 # Suscripción a tópicos de envío de parámetros a sensores y
14 actuadores
15 mqtt_client.subscribe("environmental/sensor/sub",
16 process_sensor_message_sub)
17 mqtt_client.subscribe("nutrient_solution/sensor/sub",
18 process_sensor_message_sub)
19 mqtt_client.subscribe("consumption/sensor/sub",
20 process_sensor_message_sub)
21 mqtt_client.subscribe("actuators/sub", process_sensor_message_sub)
22
23 @app.on_event("shutdown")
24 async def shutdown():
25     mqtt_client.disconnect()
26
27 # Clase para manejar la publicación de mensajes MQTT
28 class PublishRequest(BaseModel):
29     topic: str
30     message: dict
31
32 # Endpoint para publicar mensajes MQTT
33 @app.post("/mqtt/publish")
34 def publish_message(request: PublishRequest, user: dict = Depends(
35 current_user)):
36     mqtt_client.publish(request.topic, request.message)
37     return {"status": "Mensaje publicado", "topic": request.topic, "message": request.message}
38
39 # Endpoint para probar la conexión MQTT
40 @app.get("/mqtt/test")
41 def test_mqtt_connection(user: dict = Depends(current_user)):
42     try:
43         mqtt_client.connect()
44         return {"status": "Conexión exitosa"}
45     except Exception as e:
46         return {"status": "Error de conexión", "error": str(e)}

```

CÓDIGO E.3. Cliente MQTT en FastAPI

- Base de datos: contenedor basado en la imagen oficial de mongo, que ejecuta una instancia de MongoDB para almacenar los datos de la aplicación.

Mediante el archivo docker-compose.yml, se definieron y orquestaron los contenedores con Docker Compose. Se establecieron parámetros de configuración para la comunicación entre contenedores, persistencia de datos, variables de entorno, puertos y redes.

Los contenedores del backend y el frontend cuentan con sus respectivos Dockerfile, donde se especificaron las instrucciones para construir las imágenes de cada servicio. Estos archivos definen el entorno, dependencias y comandos para ejecutar cada servicio.

Cliente de actualización dinámica de DuckDNS

Dado que se trata de un entorno de prueba, no fue necesario utilizar el servicio de IP elástica de AWS para mantener una dirección fija. En su lugar, se optó por emplear el servicio DuckDNS, que permite asociar un subdominio gratuito a la instancia EC2.

Se configuró un cliente de actualización dinámica, que se ejecuta en la instancia cada cinco minutos para actualizar automáticamente la IP pública. Esto facilita el acceso remoto al sistema sin necesidad de recordar una IP dinámica.

Clonación del repositorio y preparación

El código fuente del sistema fue clonado en la máquina virtual desde un repositorio público de GitHub [58]. Dentro del directorio del sistema, se preparó un archivo de variables de entorno (.env) que contiene información sensible y parámetros de configuración del sistema. Este archivo no forma parte del repositorio, por lo que fue creado manualmente en la instancia. En él se definieron las siguientes variables necesarias para la ejecución del sistema:

- Configuración de la base de datos: host, usuario, contraseña y nombre de la base de datos para establecer la conexión con MongoDB.
- Configuración del backend: host, CORS (del inglés, Cross-Origin Resource Sharing), clave secreta, algoritmo de cifrado y duración del token JWT.
- Configuración del frontend: URL de la API del backend y del cliente WebSocket.

Configuración de certificados

Para establecer una conexión segura entre el broker MQTT de AWS IoT Core y el backend, fue necesario configurar los certificados requeridos. Para ello, se creó una carpeta denominada certificates en el directorio raíz del sistema, donde se almacenaron los certificados correspondientes:

- Certificado del cliente (archivo client.cert).
- Clave privada del cliente (archivo client.key).
- Certificado root de AWS (archivo client.cert).

Estos archivos fueron generados previamente y luego transferidos a la máquina virtual.

Bibliografía

- [1] Global Agricultural Productivity (GAP). 2016 *Global Agricultural Productivity Report*. Inf. tén. Documento en línea. Global Agricultural Productivity, 2016. URL: https://globalagriculturalproductivity.org/wp-content/uploads/2019/01/2016_GAP_Report.pdf (visitado 20-03-2025).
- [2] Raquel Salazar-Moreno, Abraham Rojano-Aguilas e Ireneo Lorenzo López-Cruz. «La eficiencia en el uso del agua en la agricultura controlada». En: *Tecnología y ciencias del agua* 5.2 (2014). Documento en línea, págs. 177-183. URL: http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S2007-242220140002&lng=es&tlang=es (visitado 20-03-2025).
- [3] Misiones Online. *Horticultura en Misiones*. URL: <https://misionesonline.net/2024/06/14/horticultura-en-misiones-2/> (visitado 20-03-2025).
- [4] Primera Edición. *Misiones: la hidroponía, cada vez más presente*. Primera Edición. URL: <https://www.primeraedicion.com.ar/nota/100627758/misiones-la-hidroponia-cada-vez-mas-presente/> (visitado 20-03-2025).
- [5] Lucas A. Garibaldi et al. «Seguridad alimentaria, medio ambiente y nuestros hábitos de Consumo». En: *Ecología Austral* 28.3 (2018), págs. 572-580. URL: https://www.scielo.org.ar/scielo.php?script=sci_arttext&pid=S1667-782X2018000400011&lng=es&tlang=es (visitado 20-03-2025).
- [6] Hidropónia FIL. URL: <https://hidroponiafil.com.ar/> (visitado 20-03-2025).
- [7] Hidrosense. URL: <https://www.hidrosense.com.br/> (visitado 20-03-2025).
- [8] iPonia. URL: <https://iponia.com.br/> (visitado 20-03-2025).
- [9] Growcast. URL: <https://www.growcast.io/> (visitado 20-03-2025).
- [10] Shanna Li. «Comparative analysis of infrastructure and Ad-Hoc wireless networks». En: *ITM Web of Conferences*. Proceedings of the International Conference on Intelligent Computing, Communication & Information Technologies (ICICCI 2018) 25 (2019). Article number 01009, pág. 01009. ISSN: 2271-2097. doi: <https://doi.org/10.1051/itmconf/20192501009>. URL: https://www itm-conferences.org/articles/itmconf/pdf/2019/02/itmconf_icicci2018_01009.pdf.
- [11] OASIS Open. *Foundational IoT Messaging Protocol MQTT Becomes International OASIS Standard*. OASIS Open. URL: <https://www.oasis-open.org/2014/11/13/foundational-iot-messaging-protocol-mqtt-becomes-international-oasis-standard/> (visitado 25-03-2025).
- [12] Amazon Web Services. ¿Qué es MQTT? AWS. URL: <https://aws.amazon.com/es/what-is/mqtt/> (visitado 25-03-2025).
- [13] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Internet Requests for Comments. RFC. doi: <10.17487/RFC8446>. URL: <https://datatracker.ietf.org/doc/html/rfc8446>.
- [14] IBM Corporation. *Protocols TCP/IP*. International Business Machines (IBM). URL: <https://docs.aws.amazon.com/iot/latest/developerguide/transport-security.html> (visitado 25-03-2025).
- [15] I. Fette y A. Melnikov. *The WebSocket Protocol*. Inf. tén. IETF, doi: <10.17487/RFC6455>. URL: <https://tools.ietf.org/html/rfc6455>.

3.7. Despliegue del sistema

Construcción y ejecución de los servicios

Después de realizar las configuraciones, se ejecutó el comando `docker-compose up -build` para construir las imágenes y lanzar los servicios definidos en el archivo `docker-compose.yml`.

Con el sistema en ejecución en EC2, se verificó el funcionamiento de la aplicación web mediante el acceso a la url <http://envirosense.duckdns.org> desde un navegador web.

3.7.3. Configuración del sistema como servicio

Para garantizar la disponibilidad tras reinicios, se configuró el sistema como un servicio gestionado por `systemd`. Se creó un archivo de definición con las instrucciones para iniciar y detener los contenedores mediante `Docker Compose`, en el que se especifican el directorio de trabajo, el usuario, el tipo de inicio y las políticas de reinicio. Finalmente, el servicio fue habilitado para ejecutarse automáticamente al iniciar el sistema operativo.

3.7.4. Resumen del proceso de despliegue

En la tabla 3.3 se presenta un resumen de las etapas realizadas durante el despliegue del sistema.

TABLA 3.3. Resumen del despliegue del sistema.

Etapa	Descripción	Acciones realizadas
Creación instancia EC2	Creación de instancia virtual de AWS.	Datos instancia: t2.micro con Ubuntu 24.04; configuración de red y claves SSH.
Configuración inicial	Preparación de máquina virtual.	Acceso por SSH, actualización del sistema, instalación de Docker, Docker Compose y DuckDNS.
Gestión dinámica de DNS	Asociación de un subdominio DuckDNS para acceso remoto.	Configuración de cliente de actualización de IP con dominio envirosense.duckdns.org.
Preparación del entorno	Clonación del código y configuración de parámetros.	Clonación desde GitHub, creación manual del archivo <code>.env</code> con variables de entorno.
Certificados MQTT	Configuración para conexión con AWS IoT Core.	Transferencia y ubicación de certificados en la instancia EC2.
Ejecución del sistema	Construcción y puesta en marcha del sistema.	Ejecución del sistema con <code>docker-compose</code> y validación por navegador web.
Ejecución como servicio	Automatización del inicio del sistema tras reinicio.	Creación de servicio en <code>systemd</code> , configuración y habilitación.

- [16] Amazon Web Services. *Transport Security in AWS IoT Core*. URL: <https://docs.aws.amazon.com/iot/latest/developerguide/transport-security.html> (visitado 25-03-2025).
- [17] Espressif Systems (Shanghai) Co., Ltd. *ESP32-WROOM-32 Datasheet*. URL: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf (visitado 26-03-2025).
- [18] Bosch Sensortec. *BME280 - Combined humidity, pressure and temperature sensor*. URL: <https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/> (visitado 26-03-2025).
- [19] ROHM Semiconductor. *BH1750 - Ambient Light Sensor (ALS) - Datasheet*. URL: https://www.mouser.com/catalog/specsheets/Rohm_11162017_ROHMS34826-1.pdf (visitado 26-03-2025).
- [20] Zhengzhou Winsen Electronics Technology Co., Ltd. *MH-Z19C NDIR CO₂ Sensor - Terminal Type Manual*. URL: https://www.mouser.com/datasheet/2/1398/Soldered_108994_co2_sensor_mh_z19-3532446.pdf (visitado 26-03-2025).
- [21] DFRobot. *Datos técnicos del sensor de pH líquido PH-4502C*. URL: <https://image.dfrobot.com/image/data/SEN0161/PH%20composite%20electrode%20manual.pdf> (visitado 26-03-2025).
- [22] METTLER TOLEDO. *Sensores de Conductividad*. METTLER TOLEDO International Inc. URL: <https://www.mt.com/es/es/home/products/Process-Analytics/conductivity-resistivity-analyzers/conductivity-sensor.html> (visitado 26-03-2025).
- [23] EC Buying. *Datos técnicos del sensor de CE*. URL: https://es.aliexpress.com/item/1005003479288815.html?spm=a2g0o.order_list.order_list_main.40.42e8194dYjUbr1&gatewayAdapt=glo2esp (visitado 26-03-2025).
- [24] Hach Company. *Sólidos (totales y disueltos)*. URL: <https://es.hach.com/parameters/solids/> (visitado 26-03-2025).
- [25] DFRobot. *Datos técnicos Sensor TDS*. URL: <https://www.dfrobot.com/product-1662.html?srsltid> (visitado 26-03-2025).
- [26] Dallas Semiconductor. *Datos técnicos Sensor DS18B20*. URL: <https://cdn.sparkfun.com/datasheets/Sensors/Temp/DS18B20.pdf> (visitado 26-03-2025).
- [27] Sparkfun Electronics. *Datos técnicos del sensor ultrasónico*. URL: <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf> (visitado 26-03-2025).
- [28] Unit Electronics. *Datos técnicos del sensor de energía eléctrica PZEM-004T*. URL: <https://uelectronics.com/wp-content/uploads/2024/06/AR4189-AR4190-Medidor-de-Energia-Electrica-AC-100A-Manual.pdf> (visitado 26-03-2025).
- [29] Songle Relay. *Datos técnicos del Relay Songle*. URL: https://datasheet4u.com/pdf-down/S/R/D/SRD-12VDC-xx-x_ETC.pdf (visitado 26-03-2025).
- [30] Damien P. George y contributors. *MicroPython*. URL: <https://micropython.org/> (visitado 26-03-2025).
- [31] CTA Electronics. *MicroPython - Recursos y Guías*. URL: <https://www.ctaelectronics.com/es/micropython/> (visitado 26-03-2025).
- [32] Tiangolo. *FastAPI*. URL: <https://fastapi.tiangolo.com/> (visitado 26-03-2025).
- [33] MongoDB, Inc. *MongoDB Documentation*. URL: <https://www.mongodb.com/> (visitado 26-03-2025).
- [34] Meta (formerly Facebook) and contributors. *React: Biblioteca de JavaScript para interfaces de usuario*. URL: <https://es.react.dev/> (visitado 26-03-2025).

Las capturas de pantalla del proceso y los comandos utilizados durante cada etapa del despliegue del sistema pueden consultarse en el apéndice C.

3.8. Código fuente del sistema

El código fuente completo del sistema, que incluye el backend, el frontend y el firmware para dispositivos IoT, se encuentra disponible públicamente en el repositorio de GitHub: <https://github.com/martinlacheski/EnviroSenseloT>. El proyecto se distribuye bajo la licencia MIT.

- [35] Docker, Inc. *Docker: Desarrollo acelerado de aplicaciones en contenedores*. URL: <https://www.docker.com/> (visitado 26-03-2025).
- [36] Amazon.com, Inc. *AWS IoT Core*. URL: <https://aws.amazon.com/es/iot-core/> (visitado 26-03-2025).
- [37] Amazon.com, Inc. *EC2, Nube de cómputo elástica de Amazon*. URL: <https://aws.amazon.com/es/ec2/> (visitado 26-03-2025).
- [38] Microsoft Corporation. *Visual Studio Code*. URL: <https://code.visualstudio.com/> (visitado 26-03-2025).
- [39] Postman, Inc. *Postman API Platform*. URL: <https://www.postman.com/> (visitado 26-03-2025).
- [40] Git. *Sistema de control de versiones Git*. URL: <https://git-scm.com/> (visitado 26-03-2025).
- [41] GitHub, Inc. *GitHub*. URL: <https://github.com/> (visitado 26-03-2025).
- [42] Beanie: *Asynchronous Python ODM for MongoDB*. URL: <https://beanie-odm.dev/> (visitado 01-04-2025).
- [43] Inc. MongoDB. Motor: *Asynchronous Python driver for MongoDB*. URL: <https://motor.readthedocs.io/en/stable/index.html> (visitado 01-04-2025).
- [44] Amazon Web Services. *AWS IoT SDKs - Guía del desarrollador*. URL: https://docs.aws.amazon.com/es_es/iot/latest/developerguide/iot-sdks.html (visitado 02-04-2025).
- [45] FastAPI WebSockets Reference. URL: <https://fastapi.tiangolo.com/reference/websockets/> (visitado 02-04-2025).
- [46] Socket.IO: *Real-time application framework*. URL: <https://socket.io/> (visitado 02-04-2025).

Capítulo 4

Ensayos y resultados

Este capítulo presenta los ensayos realizados para validar la funcionalidad de los componentes del sistema, tanto de manera individual como en conjunto. Se presentan los resultados obtenidos y su análisis.

4.1. Banco de pruebas

Se construyó un banco de prueba físico con el objetivo de validar el rendimiento, la funcionalidad y la integración de los sensores, actuadores y módulos del sistema. Este entorno controlado permitió montar los dispositivos, realizar el conexión correspondiente y simular condiciones cercanas a las reales de operación, lo que facilitó la evaluación del comportamiento general del sistema. El banco de prueba incluyó los siguientes componentes:

- Sensores y actuadores:
 - Nodo ambiental: incluye los sensores BMP280, BH1750 y MH-Z19C.
 - Nodo de consumos: compuesto por un módulo PZEM-004T y seis sensores HC-SR04.
 - Nodo solución nutritiva: equipado con un sensor de temperatura DS18B20, sensores de pH, CE, TDS y un sensor HC-SR04.
 - Nodo actuador: integrado por un módulo de relés de 4 canales (5 V, 10 A) y tres módulos de relés de 2 canales (5 V, 10 A).
- Microcontroladores:
 - Cuatro módulos ESP-WROOM-32, uno por cada nodo.
 - Cuatro placas base [59] para los módulos ESP-WROOM-32 para facilitar la conexión de los módulos a los sensores y actuadores.
- Fuentes de alimentación:
 - Una fuente de 5 V, 2 A para los relés.
 - Cuatro fuentes de 5 V, 1.5 A para los microcontroladores.
- Servidor MQTT:
 - El servicio se implementó en AWS IoT Core.
- Servidor web: