

Resumen

La presente memoria describe el desarrollo de un prototipo para monitorear y controlar de manera remota las condiciones climáticas en los invernaderos de la Facultad de Ciencias Forestales de la Universidad Nacional de Misiones. La solución propuesta integra sensores y actuadores conectados a un servidor remoto a través de una red inalámbrica y un protocolo de mensajería ligero, así como una aplicación web que permite la supervisión y el control a distancia del sistema.

Este trabajo permite optimizar el uso de recursos, mejorar la productividad y reducir costos operativos. Su implementación requirió conocimientos en sistemas embebidos, sensores, protocolos de comunicación, desarrollo de software, técnicas de seguridad y la implementación de soluciones cloud.

Resumen

La presente memoria describe el desarrollo de un prototipo de sistema para monitorear y controlar de manera remota las condiciones climáticas en los invernaderos de la Facultad de Ciencias Forestales de la Universidad Nacional de Misiones. La solución propuesta integra sensores y actuadores conectados a un servidor remoto a través de una red inalámbrica y un protocolo de mensajería ligero, así como una aplicación web que permite la supervisión y el control a distancia del sistema.

Este trabajo permite optimizar el uso de recursos, mejorar la productividad y reducir costos operativos. Su implementación requirió conocimientos en sistemas embebidos, sensores, protocolos de comunicación, desarrollo de software, técnicas de seguridad y la implementación de soluciones cloud.

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.

Agradecimientos

A Dios, por darme la fuerza y la sabiduría necesaria 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.

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

2.7.1.	Visual Studio Code	14
2.7.2.	PyMakr	14
2.7.3.	MongoDB Compass	14
2.7.4.	Postman	14
2.7.5.	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

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

Manejo de excepciones	35
Reintentos	35
Logging	35
Optimización de recursos	35
3.6.7. Resumen de nodos sensores y actuadores	35
Herramientas de despliegue y monitoreo	36
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
Documentación del proceso de despliegue	40
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	47
4.4. Pruebas de frontend	49
4.4.1. Pruebas de autenticación y autorización	50
Pruebas de autenticación	50
Pruebas de autenticación y autorización	50
4.4.2. Pruebas de funcionalidad	51
Validación de formularios	51
Validación de tablas y gráficos	52
Validación de gráficos con WebSocket	53
Pruebas de usabilidad	54
4.5. Pruebas de componentes	55
4.5.1. Pruebas de sensores	55
Prueba de sensor ambiental	55
Prueba de sensor de solución nutritiva	56
Prueba de sensor de consumos	56
4.5.2. Pruebas de actuadores	57
4.6. Pruebas de comunicación	57
4.6.1. Pruebas en sensor ambiental	57
4.6.2. Pruebas en sensor de consumos	58
4.6.3. Pruebas en sensor de solución nutritiva	59
4.6.4. Pruebas de comunicación en el actuador	59
4.7. Prueba integral del sistema	60

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

5. Conclusiones	63
5.1. Conclusiones generales	63
5.2. Próximos pasos	64
A. Modelo de datos implementado en el trabajo	65
B. Resumen de endpoints de la API	67
C. Despliegue del sistema en instancia EC2	71
C.1. Introducción	71
C.2. Creación de la instancia EC2	71
C.3. Instalación y configuración del servidor EC2	73
C.3.1. Conectarse a la instancia EC2	73
C.3.2. Instalación de Docker	74
C.3.3. Instalación de Docker Compose	74
C.3.4. Instalación del cliente DuckDNS	75
C.3.5. Descarga y configuración del repositorio EnviroSense	76
C.3.6. Configuración del sistema como servicio	77
Bibliografía	79

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

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. Ejemplo de diagrama de clase de la colección EnvironmentalSensor.....	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 inicio de sesión.....	26
3.12. Pantalla principal de la aplicación web.....	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 visualización tabular de consumos registrados.....	29
3.16. Pantalla de visualización de gráficos de consumos registrados.....	30
3.17. Pantalla de visualización tabular de actuadores.....	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 en frontend.....	46
4.5. Pruebas de WebSocket en Postman.....	47
4.6. Pruebas de WebSocket SSH EC2.....	47
4.7. Creación de usuario y validación en MongoDB.....	48
4.8. Validación de almacenamiento en MongoDB.....	49

x

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

x

4.9. Pruebas de autenticación	50
4.10. Autenticación y autorización rol administrador	51
4.11. Autenticación y autorización rol usuario	51
4.12. Validación de formularios	52
4.13. Verificación de creación de usuario	52
4.14. Pruebas de funcionalidad en tablas	53
4.15. Pruebas de funcionalidad en gráficos	53
4.16. Pruebas de funcionalidad en el dashboard	54
4.17. Interfaz de ambientes en dispositivo móvil	54
4.18. Listado en formato apaisado desde dispositivo móvil	55
4.19. Mediciones de sensor ambiental	56
4.20. Mediciones de sensor de solución nutritiva	56
4.21. Mediciones de sensor de consumos	56
4.22. Estados de actuadores	57
4.23. Pruebas de encendido de relés	57
4.24. Pruebas de comunicación en sensor ambiental	58
4.25. Solicitud de datos en sensor ambiental	58
4.26. Pruebas de comunicación en sensor de consumos	58
4.27. Solicitud de datos en sensor de consumos	58
4.28. Pruebas de comunicación en sensor de solución nutritiva	59
4.29. Solicitud de datos en sensor de solución nutritiva	59
4.30. Pruebas de comunicación en actuador	59
4.31. Solicitud de datos en actuador	60
4.32. Envío de comando en actuador	60
4.33. Envío de comando de activación al microcontrolador	60
4.34. Salida por consola del frontend	61
4.35. Salida por consola del backend	61
4.36. Salida por consola del microcontrolador	61
4.37. Salida por consola del backend	62
4.38. Verificación de persistencia en MongoDB Compass	62
4.39. Actualización del estado de los relés en el frontend	62
A.1. Modelo de datos implementado en el trabajo	66
C.1. Resumen de la instancia EC2 creada	72
C.2. Conexión SSH a la instancia EC2	73
C.3. Acceso a la aplicación web EnviroSenseloT	78
C.4. Visualización de datos en tiempo real	78

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

Índice de tablas

1.1. Características de la competencia	2
3.1. Resumen de los 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. Endpoints básicos	67
B.2. Endpoints de operación	68
B.3. Endpoints de monitoreo	69

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.

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 **utilización** de prácticas ineficientes.

1.2. Motivación

La motivación de este trabajo radica en el desarrollo e implementación de un sistema basado en IoT (del inglés, *Internet of Things*) y de bajo costo, que permite monitorear en tiempo real y controlar de manera remota los invernaderos de la Facultad de Ciencias Forestales (FCF) de la Universidad Nacional de Misiones (UNaM).

Este sistema posibilita el registro continuo de diversas variables de interés, como temperatura ambiente, humedad relativa, dióxido de carbono (CO_2), niveles de nutrientes, y consumo de agua y energía, entre otros. Los datos generados están disponibles para docentes, estudiantes e investigadores, para su uso en la realización de tesis, investigaciones y trabajos académicos.

Así, el trabajo no solo tiene un impacto directo en la producción, sino también en la formación académica y el avance científico. Proporciona una plataforma de datos para el análisis y el desarrollo de nuevas soluciones tecnológicas, alineadas con las demandas actuales de sostenibilidad ambiental y seguridad alimentaria [5].

1.3. Estado del arte

En el mercado actual, existen diversas empresas que ofrecen soluciones comerciales para optimizar la gestión de invernaderos. Estas herramientas permiten el control automatizado de variables clave como temperatura, humedad, ventilación y circulación de nutrientes o riego. La tabla 1.1 presenta una comparación de algunas de las soluciones disponibles y sus características más relevantes.

TABLA 1.1. Características de la competencia.

Empresa	Características
Hidroponía FIL [6]	Ofrece servicios en comodato de sensores y actuadores para monitorear y controlar en tiempo real variables críticas como temperatura ambiente, humedad relativa, conductividad eléctrica, pH, riego e iluminación.
Hidrosense [7]	Ofrece productos para automatizar la inyección de nutrientes en el sistema de riego a través del control del nivel de la conductividad eléctrica, la temperatura y el nivel de pH. Ofrece una plataforma para la visualización del estado, reportes y el envío de alertas.
iPONIA [8]	Ofrece productos y una plataforma para monitorear y controlar el invernadero hidropónico. Integra sensores para medir el nivel de pH, conductividad eléctrica, temperatura de la solución, temperatura ambiente y humedad relativa del aire. También ofrece dosificadores para inyectar los fertilizantes a la solución nutritiva.
Growcast [9]	Ofrece productos y una plataforma para controlar cultivos a través de sensores y actuadores que procesan y reportan datos en tiempo real. Integra sensores para medir temperatura ambiente, humedad relativa y CO_2 . Realiza el control del riego, la iluminación y la ventilación.

1.2. Motivación

La motivación de este trabajo radica en el desarrollo e implementación de un sistema basado en IoT (del inglés, *Internet of Things*) y de bajo costo, que permite monitorear en tiempo real y controlar de manera remota los invernaderos de la Facultad de Ciencias Forestales (FCF) de la Universidad Nacional de Misiones (UNaM).

Este sistema posibilita el registro continuo de diversas variables de interés, como temperatura ambiente, humedad relativa, dióxido de carbono (CO_2), niveles de nutrientes, y consumo de agua, nutrientes y energía, entre otros. Los datos generados están disponibles para docentes, estudiantes e investigadores, para su uso en la realización de tesis, investigaciones y trabajos académicos.

Así, el trabajo no solo tiene un impacto directo en la producción, sino también en la formación académica y el avance científico. Proporciona una plataforma de datos para el análisis y el desarrollo de nuevas soluciones tecnológicas, alineadas con las demandas actuales de sostenibilidad ambiental y seguridad alimentaria [5].

1.3. Estado del arte

En el mercado actual, existen diversas empresas que ofrecen soluciones comerciales para optimizar la gestión de invernaderos. Estas herramientas permiten el control de variables clave como temperatura, humedad, ventilación y circulación de nutrientes o sistemas de riego. La tabla 1.1 presenta una comparación de algunas de las soluciones disponibles y sus características más relevantes.

TABLA 1.1. Características de la competencia.

Empresa	Características
Hidroponía FIL [6]	Ofrece servicios en comodato de sensores y actuadores para monitorear y controlar en tiempo real variables críticas como temperatura ambiente, humedad relativa, conductividad eléctrica, pH, riego e iluminación.
Hidrosense [7]	Ofrece productos para automatizar la inyección de nutrientes en el sistema de riego a través del control del nivel de la conductividad eléctrica, la temperatura y el nivel de pH. Ofrece una plataforma para la visualización del estado, reportes y el envío de alertas.
iPONIA [8]	Ofrece productos y una plataforma para monitorear y controlar el invernadero hidropónico. Integra sensores para medir el nivel de pH, conductividad eléctrica, temperatura de la solución, temperatura ambiente y humedad relativa del aire. También ofrece dosificadores para inyectar los nutrientes a la solución nutritiva.
Growcast [9]	Ofrece productos y una plataforma para controlar cultivos a través de sensores y actuadores que procesan y reportan datos en tiempo real. Integra sensores para medir temperatura ambiente, humedad relativa y CO_2 . Realiza el control del riego, la iluminación y la ventilación.

- Desarrollo de una aplicación web responsive para la visualización de datos en tiempo real y el control remoto de actuadores.
- Entregables.
 - Código fuente completo del sistema (sensores, actuadores, servidor IoT, API y aplicación web).
 - Guías de instalación, configuración y operación.

El trabajo no incluyó:

- Armado de PCB.
- Desarrollo de una aplicación móvil compatible con iOS y Android.

La figura 1.1 muestra el diagrama en bloques del sistema, que evidencia la integración de hardware, software y servicios en la nube.

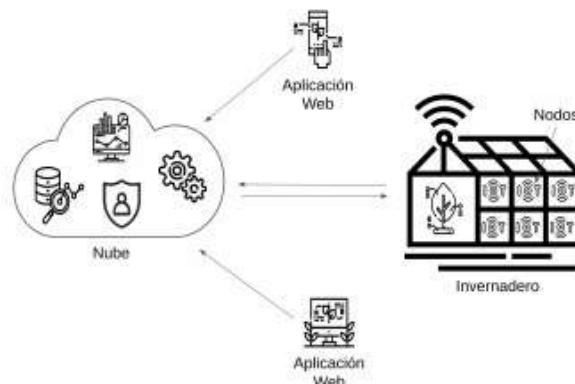


FIGURA 1.1. Diagrama en bloques del sistema.

1.5. Requerimientos

A continuación, se detallan los requerimientos técnicos asociados a los diferentes componentes del sistema.

1. Requerimientos de los nodos:
 - a) Utilizar microcontroladores basados en ESP32.
 - b) Implementar certificados TLS para seguridad en las comunicaciones.
 - c) Permitir conexión Wi-Fi.
 - d) Identificador único por nodo dentro del sistema.
 - e) Configuración remota del intervalo de envío de datos.
 - f) Los nodos sensores deben transmitir al servidor IoT.

- Desarrollo de una aplicación web responsive para la visualización de datos en tiempo real y el control remoto de actuadores.
- Entregables.
 - Código fuente completo del sistema (sensores, actuadores, servidor IoT, API y aplicación web).
 - Guías de instalación, configuración y operación.

El trabajo no incluyó:

- Armado de PCB.
- Desarrollo de una aplicación móvil compatible con iOS y Android.

La figura 1.1 muestra el diagrama en bloques del sistema, que evidencia la integración de hardware, software y servicios en la nube.

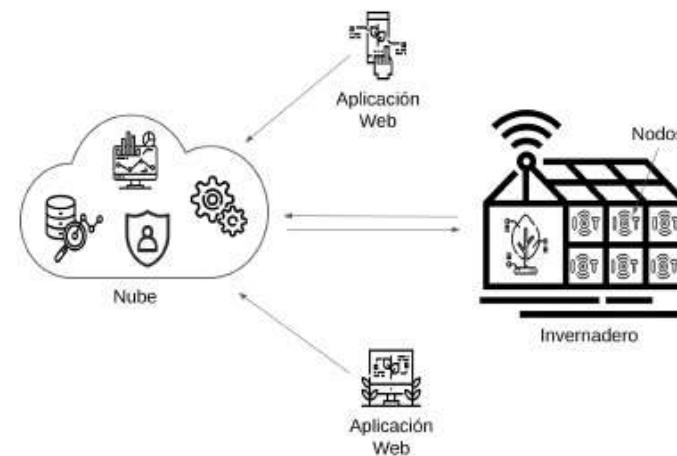


FIGURA 1.1. Diagrama en bloques del sistema.

1.5. Requerimientos

A continuación, se detallan los requerimientos técnicos asociados a los diferentes componentes del sistema.

1. Requerimientos de los nodos:
 - a) Utilizar microcontroladores basados en ESP32.
 - b) Implementar certificados TLS para seguridad en las comunicaciones.
 - c) Permitir conexión Wi-Fi.
 - d) Identificador único por nodo dentro del sistema.

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 (del inglés, *Create, Read, Update, Delete*).
 - 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).

1.5. Requerimientos

5

- e) Configuración remota del intervalo de envío de datos.
- f) Los nodos sensores deben transmitir al servidor IoT:
 - 1) Nodos ambientales: temperatura ambiente, humedad relativa, presión atmosférica, nivel de iluminación y nivel de CO_2 .
 - 2) Nodos de solución nutritiva: valores de **potencial de Hidrógeno** (**pH**), 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 (del inglés, *Create, Read, Update, Delete*).
 - 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

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.

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 detallan los diferentes protocolos de comunicación empleados 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 AP.

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 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.

- 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 Wi-Fi, 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 implementados 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 Wi-Fi, Bluetooth y BLE (del inglés, *Bluetooth Low Energy*) 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.

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 $\pm 0,5$ °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 $\pm 0,5$ °C [26].

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

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

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

FIGURA 2.11. Relé de 2 Canales 5 V 10 A³¹.

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].

³¹Imagen tomada de Relé de 2 Canales Amazon.

FIGURA 2.11. Relé de 2 Canales 5 V 10 A³¹.

2.3. Desarrollo de firmware

En esta sección se **expone** la herramienta de software **empleada** para la programación de los microcontroladores ESP32.

2.3.1. MicroPython

MicroPython es una implementación optimizada de Python 3, **diseñada** para ejecutarse en sistemas **embebidos** y dispositivos con recursos limitados, como el **ESP32**. Proporciona una forma sencilla de **programarlos** 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 **aplicadas** 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].

³¹Imagen 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 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 detalla la herramienta de software adoptada en la construcción 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 describen las herramientas de software empleadas 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 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].

2.7. Herramientas de desarrollo

En esta sección se detallan 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. PyMakr

PyMakr es un complemento para VS Code que permite programar y corregir errores en el código de MicroPython que se ejecuta en microcontroladores. Desarrollado por Pycom, proporciona una interfaz intuitiva para cargar y ejecutar scripts en el dispositivo, además de herramientas para depuración y monitoreo en tiempo real. Facilita la conexión entre el entorno de desarrollo y el microcontrolador, lo que permite a los desarrolladores escribir, transferir y depurar código de manera eficiente [39].

2.7.3. MongoDB Compass

MongoDB Compass es una herramienta con interfaz gráfica que permite interactuar de forma visual con bases de datos MongoDB. Facilita la exploración y el análisis de los datos almacenados, así como la creación de consultas, la visualización de datos y la gestión de índices. Compass permite a desarrolladores y administradores trabajar con MongoDB sin necesidad de utilizar la línea de comandos, lo que simplifica notablemente la interacción con la base de datos [40].

2.7.4. 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 [41].

2.7.5. GitHub

GitHub es una plataforma de alojamiento de repositorios Git [42] que permite a los desarrolladores trabajar y 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 [43].

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.

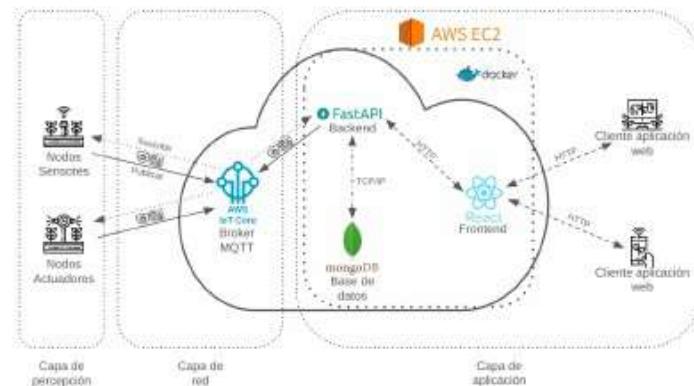


FIGURA 3.1. Arquitectura de la solución propuesta.

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.

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.

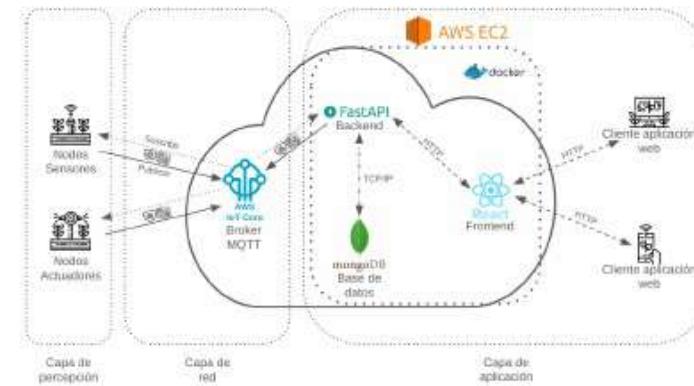


FIGURA 3.1. Arquitectura de la solución propuesta.

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, nivel de iluminación, concentración de CO_2 ,

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 permitió 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

pH, CB, 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 permitió 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 [44], 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 de datos proporcionados por los sensores, así como los requerimientos técnicos establecidos para el sistema.

3.2. Modelo de datos

1

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

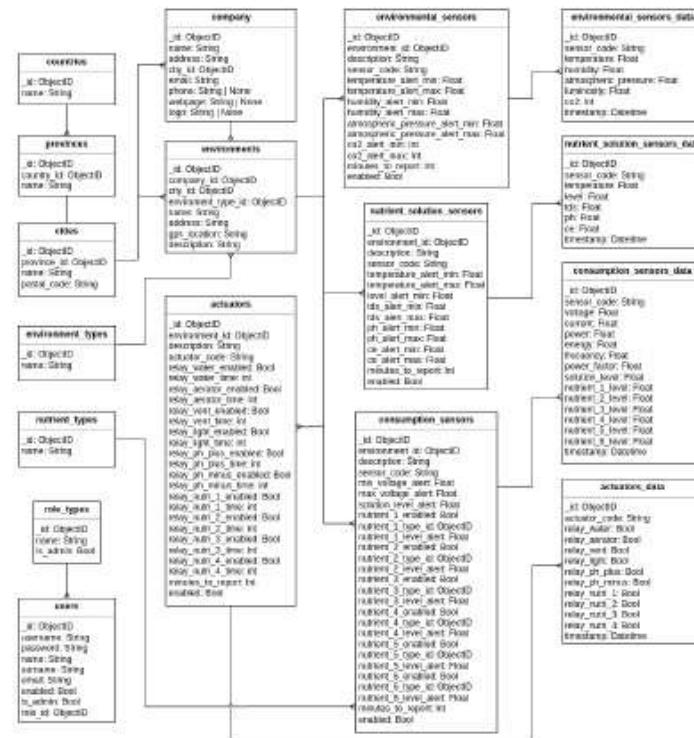


FIGURA 3.2. Modelo de datos implementado

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, empresas.
 - 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. *Modelo de dato*

17

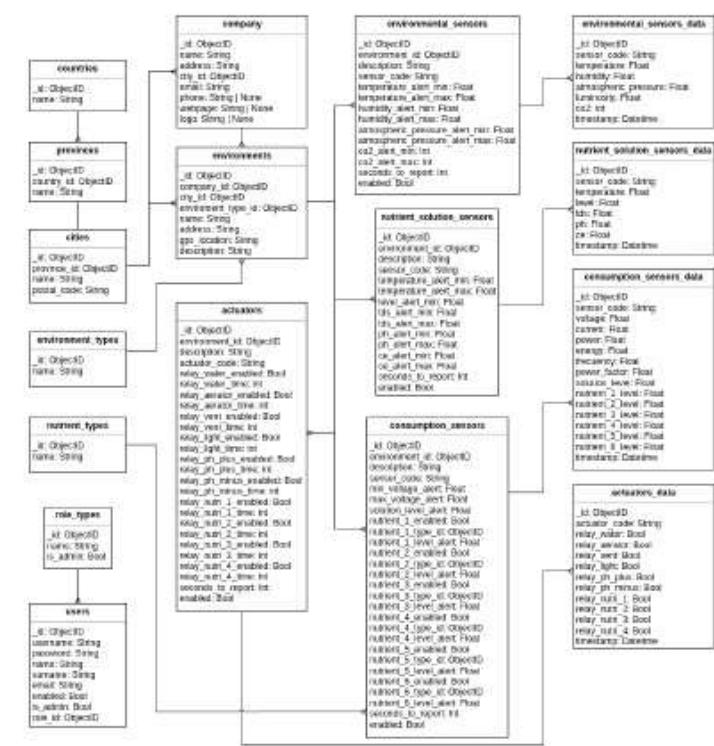


FIGURA 3.2. Modelo de datos implementado

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

- 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

 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.

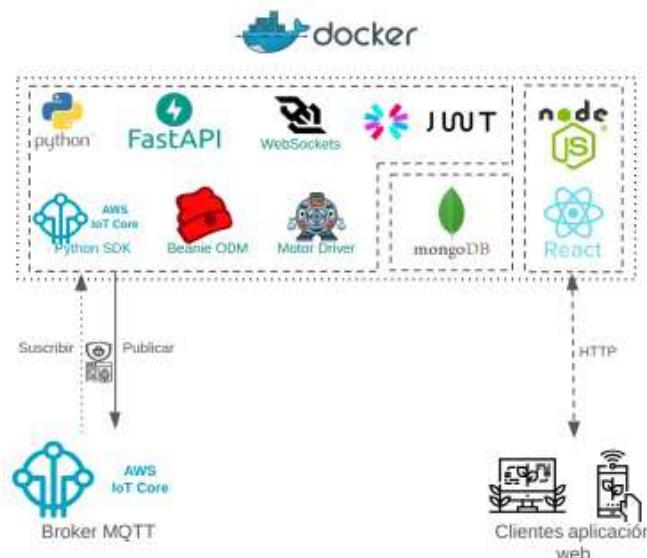


FIGURA 3.3. Arquitectura del servidor del sistema IoT.

3.2.3. 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 del servidor, junto con las tecnologías utilizadas.

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.

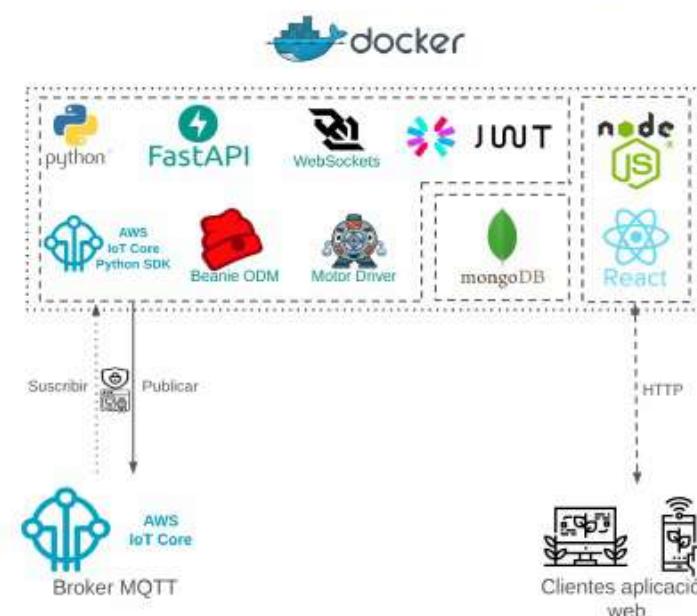


FIGURA 3.3. Arquitectura del servidor del sistema IoT.

3.4. Desarrollo del backend

19

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. 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 [45] y Motor [46], conexión con el broker MQTT implementada con el SDK (del inglés, *Software Development Kit*) de AWS IoT para Python [47] y soporte para comunicaciones en tiempo real con clientes mediante WebSocket [48].
- 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 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 los 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.

3.4.2. 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 validez 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.

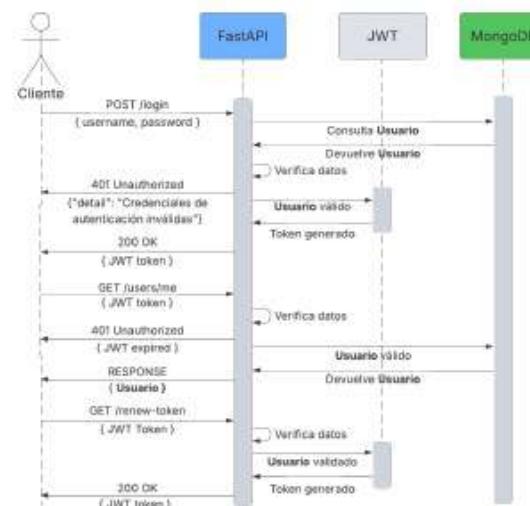


FIGURA 3.4. Esquema de autenticación y autorización.

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

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 contiene la información del usuario y se envía en el encabezado de las solicitudes a la API. El servidor verifica su validez 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.

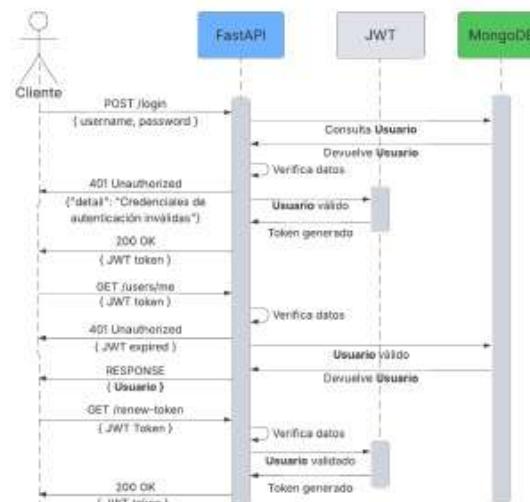


FIGURA 3.4. Esquema de autenticación y autorización.

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.

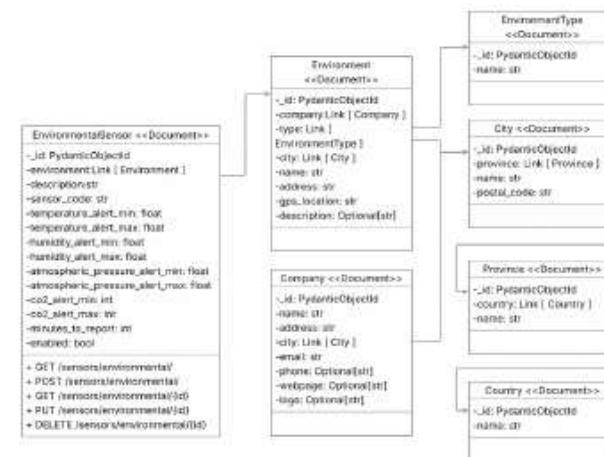


FIGURA 3.5. Diagrama de clases de los modelos implementados.

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.

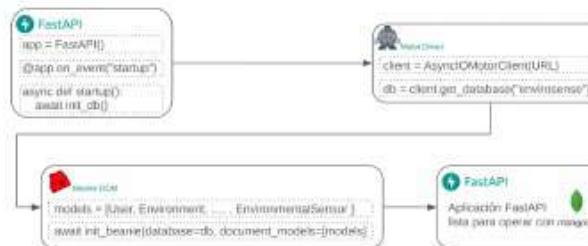


FIGURA 3.6. Pasos para la conexión de FastAPI y MongoDB.

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.

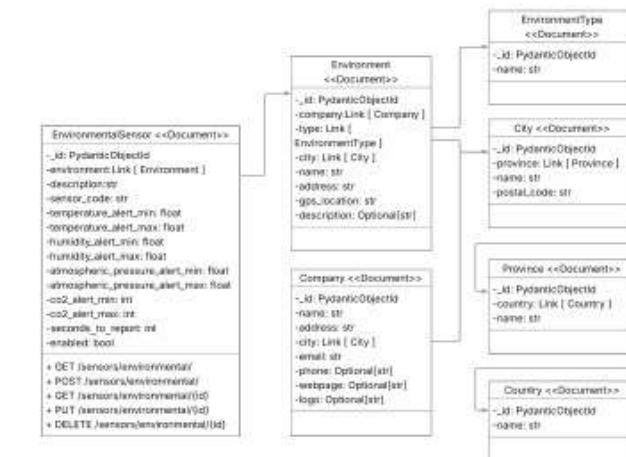


FIGURA 3.5. Ejemplo de diagrama de clase de la colección EnvironmentalSensor.

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.

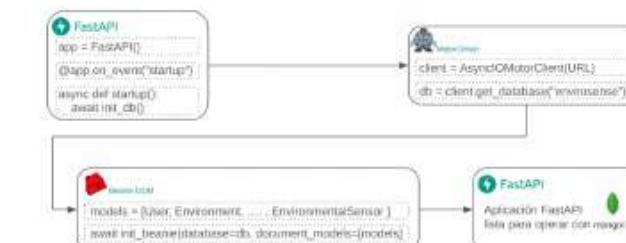


FIGURA 3.6. Pasos para la conexión de FastAPI y MongoDB.

3.4.4. Comunicación con el Broker MQTT

A continuación, se 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.

3.4.4. 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 de 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

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.

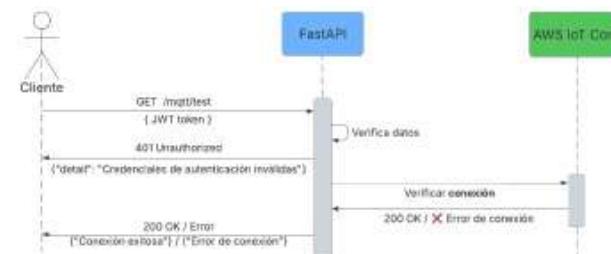


FIGURA 3.7. Pasos para verificar la conexión con el broker MQTT.

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

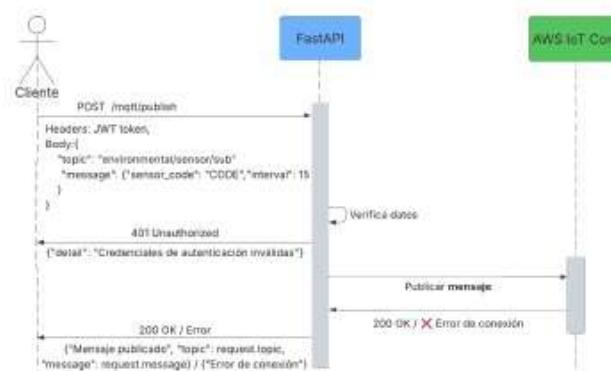


FIGURA 3.8. Pasos para publicar un mensaje en un tópico específico.

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 determinado.

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

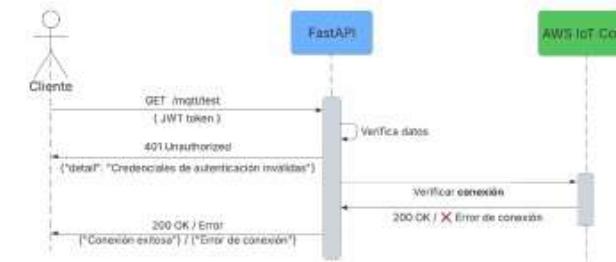


FIGURA 3.7. Pasos para verificar la conexión con el broker MQTT.

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

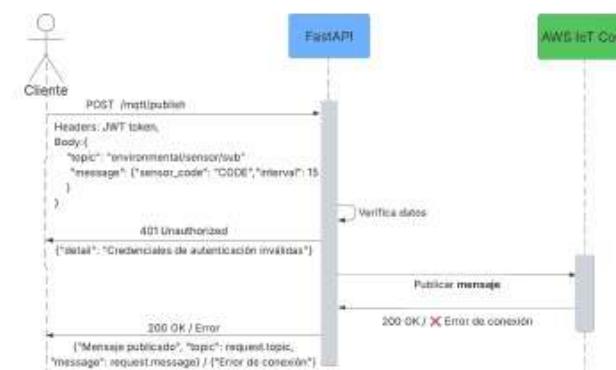


FIGURA 3.8. Pasos para publicar un mensaje en un tópico específico.

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.

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 describe la secuencia de acciones que permite a la aplicación FastAPI conectarse al broker MQTT y suscribirse a los tópicos requeridos.



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.

3.5. Desarrollo del frontend

25

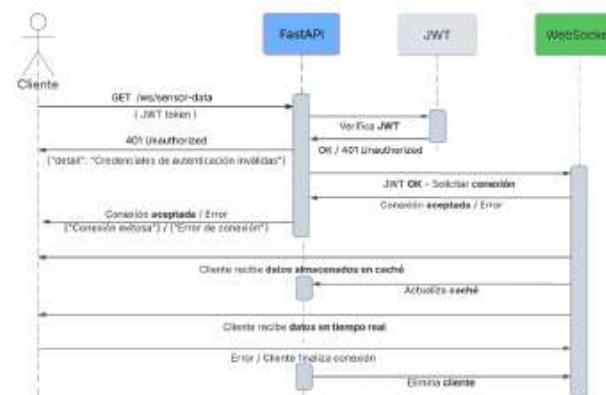


FIGURA 3.10. Pasos para establecer la conexión WebSocket

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.

3.5. Desarrollo del frontend

25

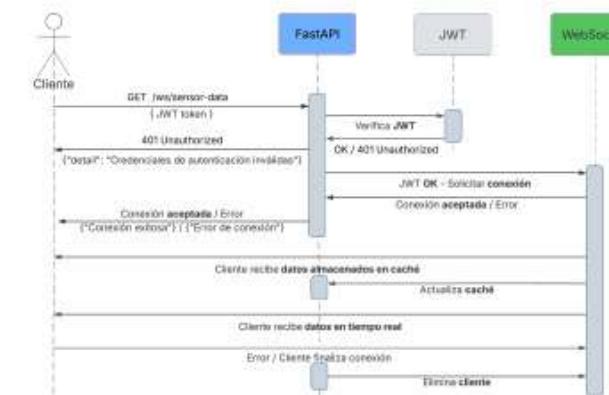


FIGURA 3.10. Pasos para establecer la conexión WebSocket

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 [49]. Para los iconos de la interfaz se utilizó Bootstrap Icons [50], que proporciona una amplia variedad de iconos gratuitos personalizables y fáciles de usar.

La comunicación con el servidor se implementó mediante la biblioteca Axios [51] 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 [52] y la navegación entre páginas se gestionó a través de la biblioteca React Router [53].

Para la visualización de los gráficos, se empleó la biblioteca Recharts [54], mientras que para la representación de tablas se utilizó la biblioteca React Data Table Component [55].

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.

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.

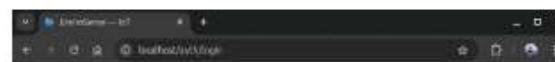


FIGURA 4.6. Pantalla de .

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.

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 el 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 3.11 muestra la pantalla de inicio de sesión, donde los usuarios ingresan sus credenciales para acceder a la aplicación web.

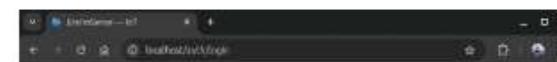


FIGURA 3.11. Pantalla de .

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.

3.5. Desarrollo del frontend

27

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

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.

3.5. Desarrollo del frontend

27

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.

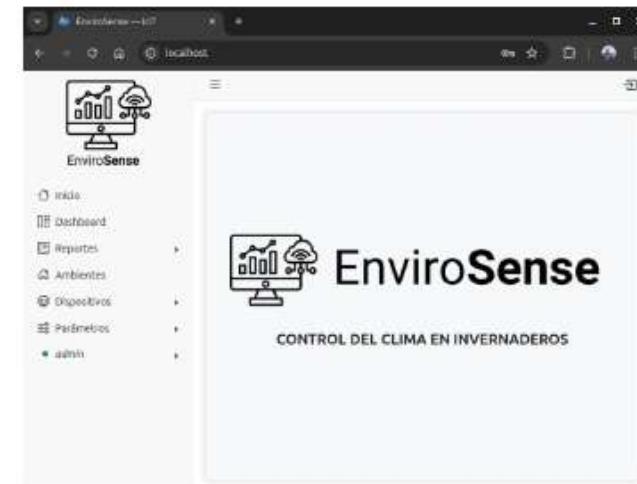


FIGURA 3.12. Pantalla principal de la aplicación web.

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, destinadas a **usuarios registrados** que acceden a funcionalidades generales de la aplicación.
- Rutas restringidas: requieren autenticación y permisos específicos, habilitadas solo para ciertos roles (como administradores), por ejemplo para acceder a 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.



FIGURA 3.13. Interfaz de navegación según permisos de usuario



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

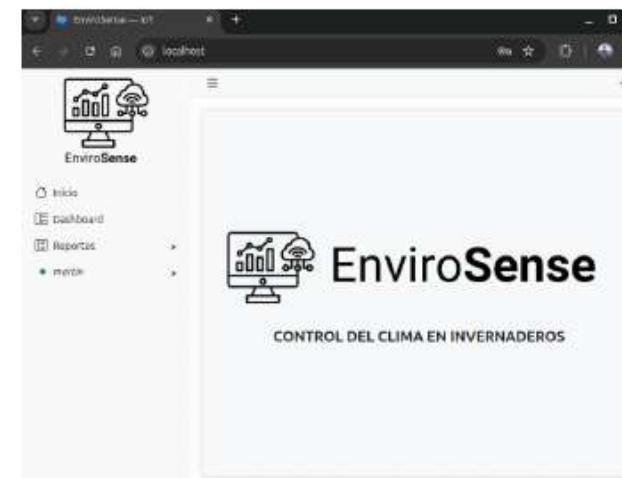


FIGURA 3.13. Interfaz de navegación según permisos de usuario estándar.

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

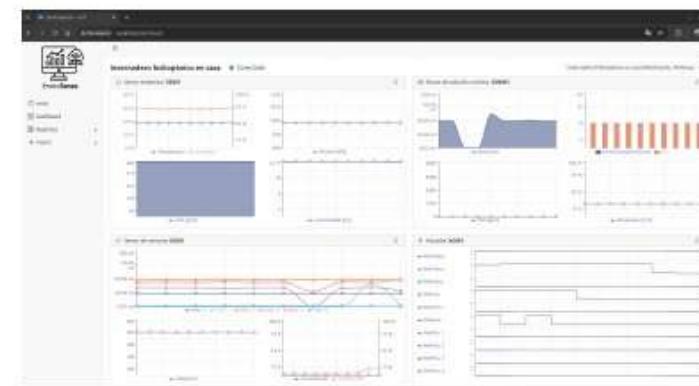


FIGURA 3.14. Visualización de datos en tiempo real.

3.5. Desarrollo del frontend

24

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 recopilados por cada sensor.
 - Alternar entre una vista tabular y una vista gráfica.
 - Paginación 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 te

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



3.5. Desarrollo del frontend

29

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 obtienen la información procesada desde la API del backend, lo que garantiza que los datos mostrados estén siempre actualizados.

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 visualización tabular de consumos registrados.

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

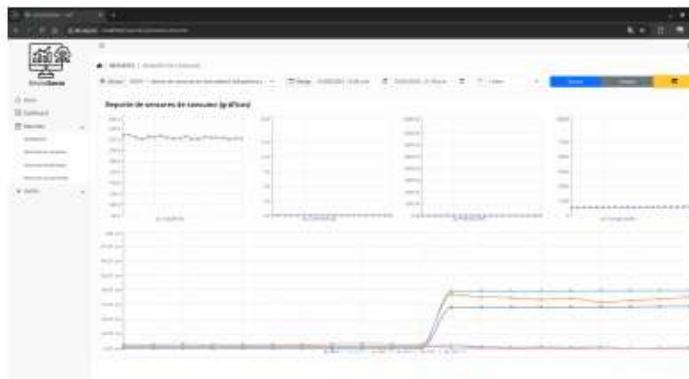


FIGURA 3.16. Pantalla de

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

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.



FIGURA 3.16. Pantalla de visualización de gráficos de consumos registrados.

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

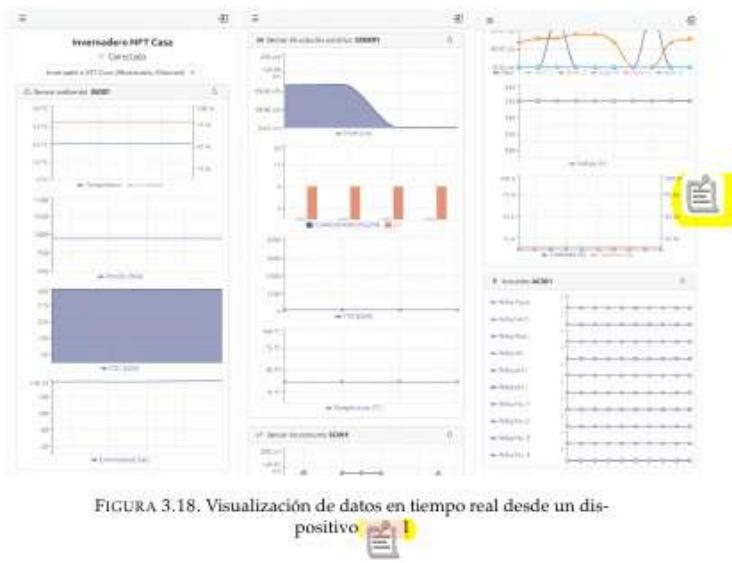


FIGURA 3.17. Pantalla de visualización tabular de actuadores.

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.

3.6. Desarrollo del firmware de los dispositivos IoT

31



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 embebido 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 **bibliotecas**: importación de módulos para sensores, Wi-Fi, MQTT, gestión del tiempo y otras funcionalidades.

3.6. Desarrollo del firmware de los dispositivos IoT

31

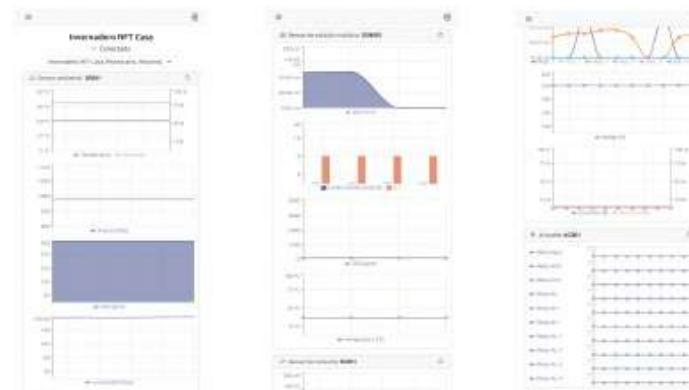


FIGURA 3.18. Visualización de datos en tiempo real desde un dispositivo móvil.

3.6. Desarrollo del firmware de los dispositivos IoT

Esta sección presenta el desarrollo de los distintos firmwares implementados en MicroPython para los diferentes tipos de nodos del sistema. Se describe la arquitectura general del software embebido 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 **bibliotecas**: 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).

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.

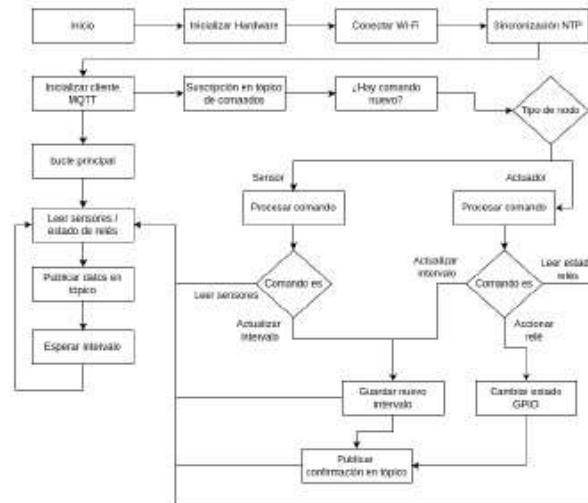


FIGURA 3.19. Flujo de ejecución del código en los s.

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.

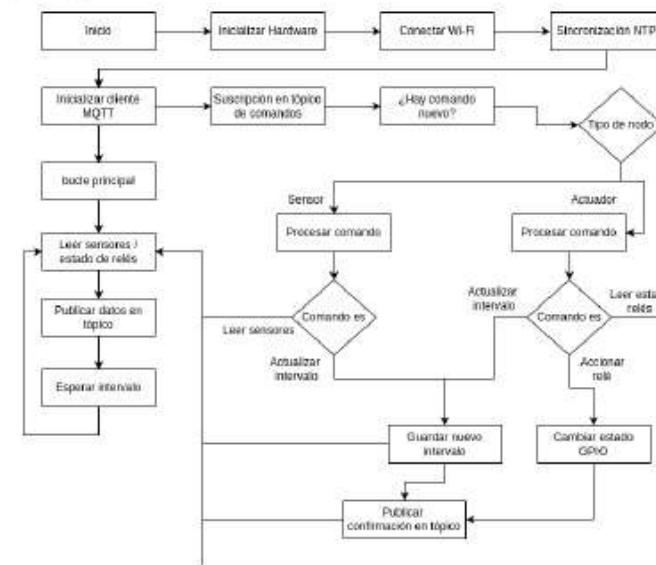


FIGURA 3.19. Flujo de ejecución del código en los nodos.

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  rias utilizadas para gestionar la comunicación inalámbrica:

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 garantizó la persistencia de la configuración incluso después de reinicios.

A continuación, se detallan los archivos de configuración utilizados en 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 (15 segundos). 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  rias 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.

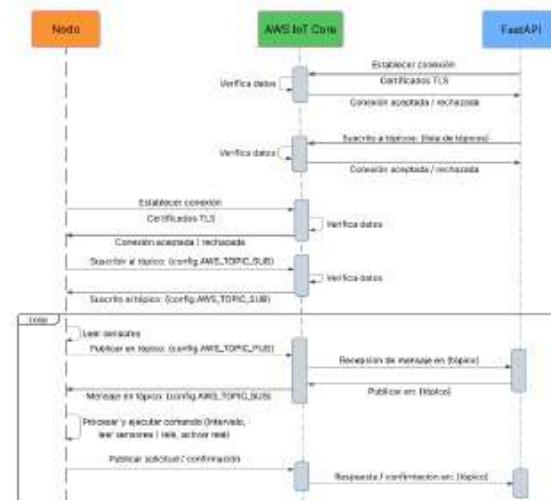


FIGURA 3.20. Flujo de comunicación entre nodos y broker

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.

Wi-Fi

Para gestionar la conexión Wi-Fi, se empleó la biblioteca Wifi Manager [56]. 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 [57], 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.

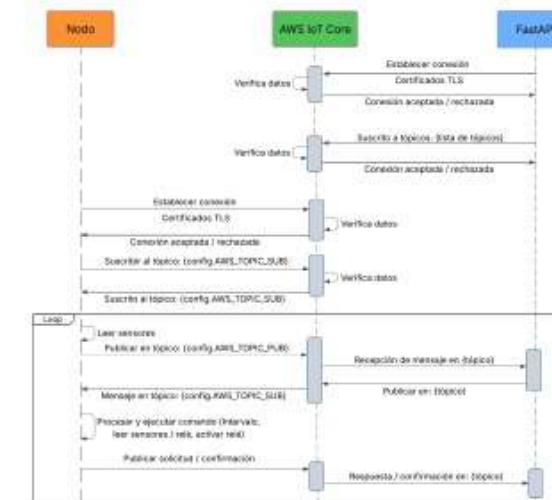


FIGURA 3.20. Flujo de comunicación entre nodos y broker MQTT.

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.

TABLA 3.2. Resumen de nodos: funciones y firmware.

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

TABLA 3.2. Resumen de nodos: funciones y firmware.

Nodo	Funciones principales	Descripción del firmware
Sensor ambiental	Medición de temperatura, humedad y presión (BME280).	Comunicación I ² C para BME280.
	Medición del nivel de iluminación (BH1750).	UART para MH-Z19C.
	Medición de concentración de CO ₂ (MH-Z19C).	Medición inicial de CO ₂ .
	Conexión Wi-Fi. Conexión broker MQTT.	Envío de datos en JSON vía MQTT.
	Configuración de intervalo.	Sincronización NTP.
	Indicador LED.	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.
		Envío de datos en formato JSON vía MQTT.
		Configuración de intervalo.
		Indicador LED.
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.
		Envío de datos en JSON vía MQTT.
		Configuración de intervalo.
Actuador		Indicador LED.
		Control de relés para dispositivos (bombas, aireadores, luces).
		Ejecución simultánea de comandos.
		Conexión Wi-Fi. Conexión broker MQTT.
		Ejecución de temporizadores con Timers ESP32.
		Manejo de estados de relés en JSON.
Indicador LED.		Indicador LED.
		Sincronización NTP.

Herramientas de despliegue y monitoreo

El despliegue del firmware en los nodos se realizó mediante el uso de PyMakr. Esta herramienta permitió:

- Transferir los archivos del firmware (como main.py, config.py y los certificados TLS) al sistema de archivos interno del ESP32.
- Establecer una conexión serial para monitorear logs en tiempo real durante la depuración.

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 maquina 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.

Su integración con el entorno de desarrollo resultó fundamental para facilitar tanto la implementación como la depuración del firmware en los nodos.

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.

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:

- Imagen de maquina 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 [58]: para asociar nombre de dominio a la IP pública dinámica.

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 [59] y servir la aplicación web desarrollada en React.

- 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  `root.cert`).

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

- 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 nombre de dominio gratuito a la IP pública de la instancia EC2.

Se configuró un cliente de actualización, 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 [60]. 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.crt`).
- Clave privada del cliente (archivo `client.key`).
- Certificado root de AWS (archivo `root.crt`).

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

3.7. Despliegue del sistema

39

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 .env 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.

3.7. Despliegue del sistema

39

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 .env 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.

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.

Documentación del proceso de despliegue

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 backend, frontend y el firmware de los nodos IoT, se encuentra disponible en el repositorio público de GitHub [60]. El proyecto se distribuye bajo la licencia MIT.

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 conexionado 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 de 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,  para los microcontroladores.
- Servidor MQTT:
 - El servicio se implementó en AWS IoT Core.
-  idor web:

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 y realizar el conexionado correspondiente.

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 de 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 [61] 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,  para los microcontroladores.
- Servidor MQTT:
 - El servicio se implementó en AWS IoT Core.

- El backend, frontend y la base de datos se desplegaron en una máquina virtual EC2 de AWS, a través del entorno de laboratorio provisto por AWS Academy Learner Lab.

La figura 4.1 muestra el banco de pruebas utilizado para la validación del sistema, donde se observan los sensores, actuadores, microcontroladores y la alimentación eléctrica.



FIGURA 4.1. Banco de pruebas del sistema EnviroSenseIoT.

4.2. Criterios de evaluación

Una vez construido el banco de pruebas, se definieron criterios específicos para evaluar el desempeño del sistema en términos de funcionalidad, comunicación e integración. Las pruebas se enfocaron en verificar el correcto funcionamiento de cada componente de forma individual (sensores, actuadores, backend y frontend) y también en su comportamiento integrado como sistema completo.

Se realizaron ensayos para comprobar la lectura de los sensores, la activación de actuadores, la comunicación entre microcontroladores y el servidor web mediante el protocolo MQTT, así como la correcta gestión de datos en el backend y la usabilidad de la interfaz web. Estas validaciones permitieron identificar el cumplimiento de los objetivos funcionales establecidos para cada módulo.

La tabla 4.1 resume los criterios aplicados durante el proceso de validación en el entorno de prueba.

- Servidor web:

- El backend, frontend y la base de datos se desplegaron en una máquina virtual EC2 de AWS, a través del entorno de laboratorio provisto por AWS Academy Learner Lab.

La figura 4.1 muestra el banco de pruebas utilizado para la validación del sistema, donde se observan los sensores, actuadores, microcontroladores y la alimentación eléctrica.



FIGURA 4.1. Banco de pruebas del sistema EnviroSenseIoT.

4.2. Criterios de evaluación

Una vez construido el banco de pruebas, se definieron criterios específicos para evaluar el desempeño del sistema en términos de funcionalidad, comunicación e integración. Las pruebas se enfocaron en verificar el correcto funcionamiento de cada componente de forma individual (sensores, actuadores, backend y frontend) y también en su comportamiento integrado como sistema completo.

Se realizaron ensayos para comprobar la lectura de los sensores, la activación de actuadores, la comunicación entre microcontroladores y el servidor web mediante el protocolo MQTT, así como la correcta gestión de datos en el backend y la usabilidad de la interfaz web. Estas validaciones permitieron identificar el cumplimiento de los objetivos funcionales establecidos para cada módulo.

La tabla 4.1 resume los criterios aplicados durante el proceso de validación en el entorno de prueba.

4.3. Pruebas de backend

43

TABLA 4.1. Criterios de evaluación del banco de pruebas.

Criterio	Componente	Descripción
Verificación de endpoints y almacenamiento	Backend	Comprobación del funcionamiento de los endpoints y del registro de información en la base de datos.
Usabilidad de la interfaz web	Frontend	Evaluación de la facilidad de uso y el acceso a funcionalidades.
Verificación de mediciones	Microcontroladores	Verificación de los valores reportados por los sensores.
Activación de actuadores	Microcontroladores	Validación de la activación de los módulos de relés ante eventos enviados.
Verificación de comunicación MQTT	Comunicación MQTT	Verificación de la comunicación entre microcontroladores y servidor web a través del intercambio de mensajes en los tópicos.
Integración de módulos	Todo el sistema	Verificación del funcionamiento conjunto de sensores, actuadores y servidor web.

4.3. Pruebas de backend

El backend del sistema cuenta con un total de endpoints, a través de los cuales se realiza la interacción con los distintos módulos y servicios. Para su validación, se utilizó la herramienta Postman, que permitió realizar pruebas funcionales destinadas a verificar que cada endpoint respondiera correctamente a las solicitudes y devolviera los datos esperados conforme a la lógica definida.

Además, FastAPI incorpora una interfaz Swagger [60], que permite documentar, explorar y probar los endpoints desde el navegador. La figura 4.2 muestra esta interfaz, donde los endpoints se organizan por categoría y pueden consultarse y evaluarse en tiempo real.

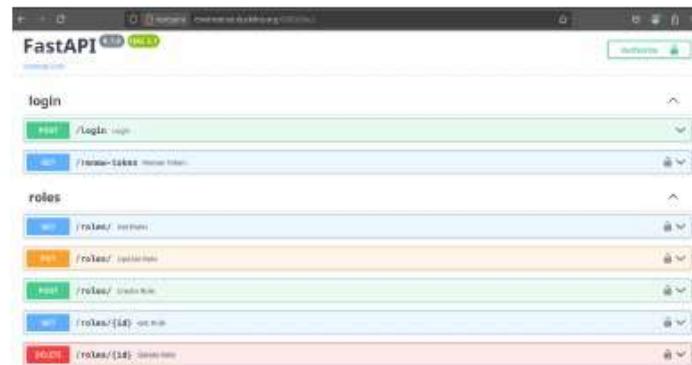


FIGURA 4.2. Interfaz de Swagger.

4.3. Pruebas de backend

43

TABLA 4.1. Criterios de evaluación del banco de pruebas.

Criterio	Componente	Descripción
Verificación de endpoints y almacenamiento	Backend	Comprobación del funcionamiento de los endpoints y del registro de información en la base de datos.
Usabilidad de la interfaz web	Frontend	Evaluación de la facilidad de uso y el acceso a funcionalidades.
Verificación de mediciones	Microcontroladores	Verificación de los valores reportados por los sensores.
Activación de actuadores	Microcontroladores	Validación de la activación de los módulos de relés ante eventos enviados.
Verificación de comunicación MQTT	Comunicación MQTT	Verificación de la comunicación entre microcontroladores y servidor web a través del intercambio de mensajes en los tópicos.
Integración de módulos	Todo el sistema	Verificación del funcionamiento conjunto de sensores, actuadores y servidor web.

4.3. Pruebas de backend

El backend del sistema cuenta con un total de 111 endpoints, a través de los cuales se realiza la interacción con los distintos módulos y servicios. Para su validación, se utilizó la herramienta Postman, que permitió realizar pruebas funcionales destinadas a verificar que cada endpoint respondiera correctamente a las solicitudes y devolviera los datos esperados conforme a la lógica definida.

Además, FastAPI incorpora una interfaz Swagger [62], que permite documentar, explorar y probar los endpoints desde el navegador. La figura 4.2 muestra esta interfaz, donde los endpoints se organizan por categoría y pueden consultarse y evaluarse en tiempo real.

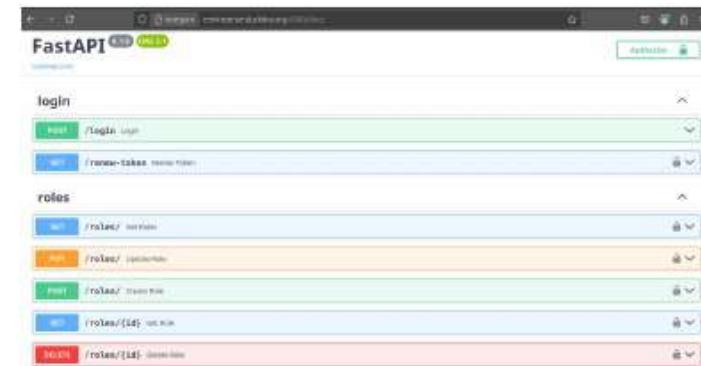


FIGURA 4.2. Interfaz de Swagger.

4.3.1. Pruebas en Postman

En el repositorio de GitHub [58] se encuentra la carpeta `Tests`, que contiene un archivo JSON con la colección completa de pruebas realizadas con Postman. Dicha colección abarca todos los endpoints del backend evaluados, junto con los resultados correspondientes.

Tambien se incluyen un archivo CSV (del inglés *Comma Separated Values*) y un archivo en formato texto, en el que se detallan los endpoints, los métodos HTTP y los resultados obtenidos, para permitir una trazabilidad clara del proceso de validación.

La figura 4.3 presenta un ejemplo de las pruebas realizadas en Postman, donde se detalla la respuesta de los endpoints, el método HTTP utilizado, la URL del endpoint, el código de estado, el tiempo y el tamaño de la respuesta.

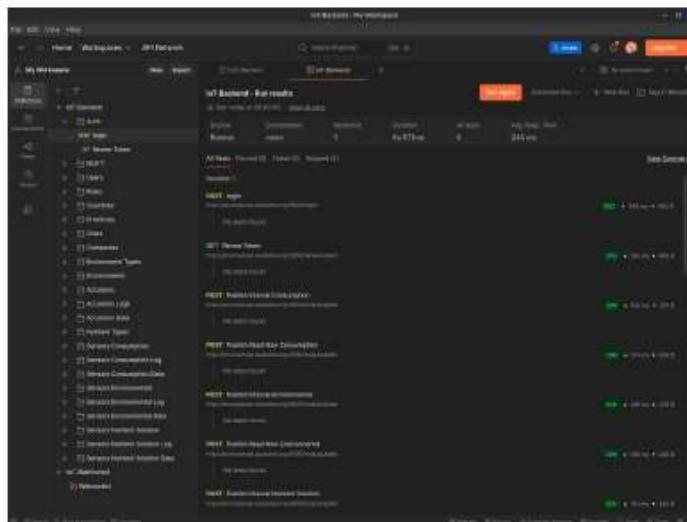


FIGURA 4.3. Pruebas realizadas en Postman.

Evaluación general

Los tiempos de respuesta obtenidos durante las pruebas fueron, en su mayoría, adecuados para un entorno compuesto por servicios desplegados en contenedores Docker dentro de una instancia EC2 de AWS. El promedio se mantuvo cercano a los 200 ms, valor que garantiza una experiencia de usuario satisfactoria y un rendimiento eficiente del sistema.

Las operaciones más exigentes, como las consultas filtradas sobre grandes volúmenes de datos o el cambio de contraseñas, presentaron tiempos de respuesta ligeramente superiores. Sin embargo, se mantuvieron dentro de márgenes aceptables, al considerar la complejidad de estas tareas y las características del entorno de ejecución en la nube.

4.3.1. Pruebas en Postman

En el repositorio de GitHub [60] se encuentra la carpeta `Tests`, que contiene un archivo JSON con la colección completa de pruebas realizadas con Postman. Esta colección incluye la evaluación de la totalidad de los endpoints del backend, junto con sus respectivos resultados.

Tambien se incluyen un archivo CSV (del inglés *Comma Separated Values*) y un archivo en formato texto, en el que se detallan los endpoints, los métodos HTTP y los resultados obtenidos, para permitir una trazabilidad clara del proceso de validación.

La figura 4.3 presenta un ejemplo de las pruebas realizadas en Postman, donde se detalla la respuesta de los endpoints, el método HTTP utilizado, la URL del endpoint, el código de estado, el tiempo y el tamaño de la respuesta.

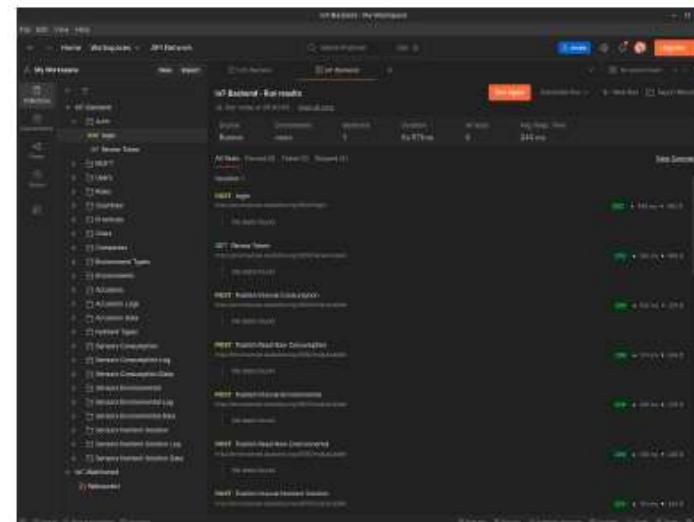


FIGURA 4.3. Pruebas realizadas en Postman.

Evaluación general

Los tiempos de respuesta obtenidos durante las pruebas fueron, en su mayoría, adecuados para un entorno compuesto por servicios desplegados en contenedores Docker dentro de una instancia EC2 de AWS. El promedio se mantuvo cercano a los 200 ms, valor que garantiza una experiencia de usuario satisfactoria [63] y un rendimiento eficiente del sistema.

Las operaciones más exigentes, como las consultas filtradas sobre grandes volúmenes de datos o el cambio de contraseñas, presentaron tiempos de respuesta ligeramente superiores. Sin embargo, se mantuvieron dentro de márgenes aceptables, al considerar la complejidad de estas tareas y las características del entorno de ejecución en la nube.

La [figura 4.2](#) presenta los resultados obtenidos durante las pruebas de desempeño, organizados por categoría funcional. Se detallan los métodos evaluados y los tiempos de respuesta registrados, lo que permite analizar el comportamiento del sistema ante distintos tipos de solicitudes.

TABLA 4.2. Resultados de tiempos de respuesta de las pruebas.

Categoría	Método	Tiempo de Respuesta
Autenticación y usuarios	POST Login	496 ms
	GET Current User	181 ms
	PATCH Update Password	827 ms
Publicación de mensajes en MQQT y logs	POST Publish MQQT	153 – 181 ms
	GET Logs (actuadores y sensores)	173 – 193 ms
Consulta de datos de dispositivos	GET Sensors Data	~ 208 ms
Operaciones de creación, modificación y eliminación	GET Actuators Data	271 ms
	POST Create User / Company	~ 490 ms
	POST Create Sensor / Data	154 – 194 ms
Datos geográficos y configuración	PUT / DELETE	161 – 188 ms
	GET Countries / Provinces / Cities	176 – 187 ms
	POST Create Environment / Actuator / Sensor	179 – 194 ms

El análisis de los tiempos de respuesta arrojó un promedio general de **204,38 ms** sobre un total de **111 solicitudes**. Esta métrica refuerza la conclusión de que el sistema opera de manera eficiente dentro del entorno en la nube.

La tabla [4.3](#) presenta la distribución de los tiempos de respuesta por método HTTP. Además, se observó un comportamiento diferenciado según el tipo de operación solicitada, como se resume a continuación:

TABLA 4.3. Promedios de tiempos de respuesta por método HTTP.

Método HTTP	Tiempo de Respuesta
GET	191.0 ms (46 solicitudes)
POST	199.47 ms (36 solicitudes)
PUT	202.64 ms (14 solicitudes)
DELETE	195.77 ms (13 solicitudes)
PATCH	668.5 ms (2 solicitudes)

Resultados generales de las pruebas

A continuación, se resumen los principales resultados obtenidos durante las pruebas realizadas en Postman:

- Todas las pruebas fueron exitosas, sin registrar errores en el backend ni fallos en las respuestas.

La tabla [4.2](#) presenta los resultados obtenidos durante las pruebas de desempeño, organizados por categoría funcional. Se detallan los métodos evaluados y los tiempos de respuesta registrados, lo que permite analizar el comportamiento del sistema ante distintos tipos de solicitudes.

TABLA 4.2. Resultados de tiempos de respuesta de las pruebas.

Categoría	Método	Tiempo de Respuesta
Autenticación y usuarios	POST Login	496 ms
	GET Current User	181 ms
	PATCH Update Password	827 ms
Publicación de mensajes en MQQT y logs	POST Publish MQQT	153 – 181 ms
	GET Logs (actuadores y sensores)	173 – 193 ms
Consulta de datos de dispositivos	GET Sensors Data	~ 208 ms
Operaciones de creación, modificación y eliminación	GET Actuators Data	271 ms
	POST Create User / Company	~ 490 ms
	POST Create Sensor / Data	154 – 194 ms
Datos geográficos y configuración	PUT / DELETE	161 – 188 ms
	GET Countries / Provinces / Cities	176 – 187 ms
	POST Create Environment / Actuator / Sensor	179 – 194 ms

El análisis de los tiempos de respuesta arrojó un promedio general de **204,38 ms** sobre un total de 111 solicitudes. Esta métrica refuerza la conclusión de que el sistema opera de manera eficiente dentro del entorno en la nube.

La tabla [4.3](#) presenta la distribución de los tiempos de respuesta por método HTTP. Además, se observó un comportamiento diferenciado según el tipo de operación solicitada, como se resume a continuación:

TABLA 4.3. Promedios de tiempos de respuesta por método HTTP.

Método HTTP	Tiempo (ms)	Solicitudes
GET	191,00	46
POST	199,47	36
PUT	202,64	14
DELETE	195,77	13
PATCH	668,50	2

Resultados generales de las pruebas

A continuación, se resumen los principales resultados obtenidos durante las pruebas realizadas en Postman:

- Todas las pruebas fueron exitosas, sin registrar errores en el backend ni fallos en las respuestas.

- Los tiempos de respuesta se mantuvieron estables y apropiados para un sistema desplegado en contenedores sobre infraestructura en la nube.
- Las operaciones clave, como la autenticación, la creación de entidades y las consultas de datos, mostraron tiempos de respuesta ligeramente superiores, pero no afectaron la experiencia del usuario. El rendimiento se mantuvo satisfactorio en el entorno de prueba con servicios en la nube.

4.3.2. Pruebas de validación de WebSocket

Para validar la comunicación en tiempo real mediante WebSocket, se realizaron pruebas enfocadas en verificar la conexión persistente entre múltiples clientes simultáneos y el servidor backend, así como el correcto envío y recepción de mensajes.

Se utilizaron instancias en Postman y navegadores web para establecer conexiones entre el frontend y el backend mediante WebSocket.

Las pruebas confirmaron la estabilidad de la comunicación, la capacidad del backend para distribuir mensajes a múltiples clientes y la recepción inmediata de datos. Estos resultados demostraron la eficiencia y confiabilidad del canal WebSocket implementado.

La figura 4.4 ilustra un ejemplo de la conexión WebSocket entre el frontend y el backend. A la izquierda, se observa la conexión establecida desde el frontend; a la derecha, las realizadas mediante Postman. También se visualiza una terminal de Linux conectada al backend en una instancia de AWS EC2, donde se evidencia la conexión activa de tres clientes simultáneos.

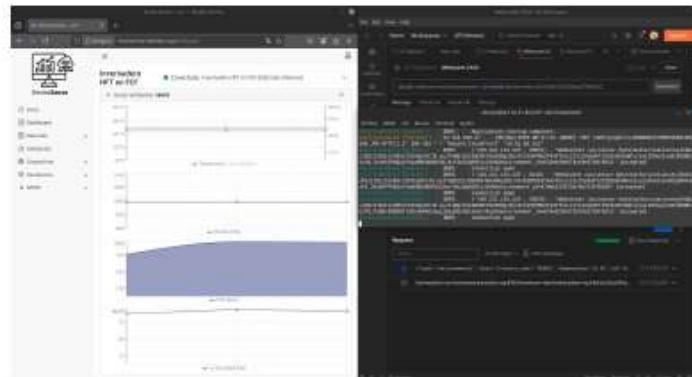


FIGURA 4.4. Pruebas de WebSocket.

4.3.3. Pruebas de validación de almacenamiento en MongoDB

Se llevaron a cabo pruebas para validar el almacenamiento de los datos en MongoDB, con el objetivo de comprobar que las operaciones de escritura realizadas desde el backend, tales como la creación de usuarios, sensores, mediciones y eventos, se registrarán en las colecciones correspondientes.

- Los tiempos de respuesta se mantuvieron estables y apropiados para un sistema desplegado en contenedores sobre infraestructura en la nube.
- Las operaciones clave, como la autenticación, la creación de entidades y las consultas de datos, mostraron tiempos de respuesta ligeramente superiores, pero no afectaron la experiencia del usuario. El rendimiento se mantuvo satisfactorio en el entorno de prueba.

4.3.2. Pruebas de validación de WebSocket

Para validar la comunicación en tiempo real mediante WebSocket, se realizaron pruebas enfocadas en verificar la conexión persistente entre múltiples clientes simultáneos y el servidor backend, así como el correcto envío y recepción de mensajes.

Se utilizaron instancias en Postman y navegadores web para establecer conexiones entre el frontend y el backend mediante WebSocket.

La figura 4.4 ilustra un ejemplo de la conexión WebSocket realizada desde el frontend.



FIGURA 4.4. Pruebas de WebSocket desde el frontend.

4.4. Pruebas de frontend

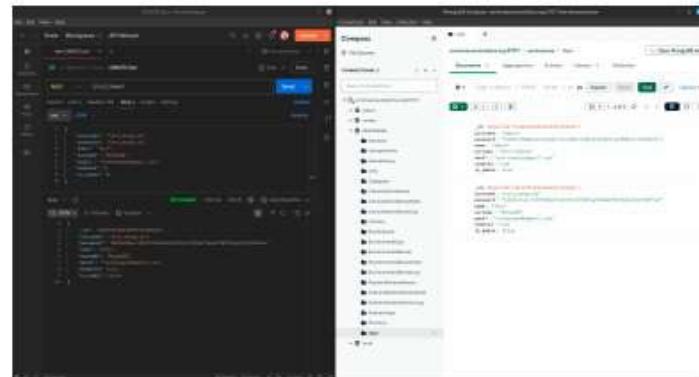
47

Para ello, se realizaron las siguientes acciones:

- Se ejecutaron operaciones desde Postman, para generar nuevas entidades en el sistema.
- A continuación, se accedió directamente a la base de datos MongoDB, a través de la herramienta MongoDB Compass [61], para inspeccionar las colecciones y verificar la existencia y la estructura de los documentos insertados.
- Se validó que las operaciones de actualización y eliminación impactaran correctamente sobre los documentos correspondientes en MongoDB.

Estas validaciones permitieron confirmar que el backend realiza una correcta persistencia de los datos en MongoDB y que no se observaron inconsistencias, ni pérdidas de información durante las operaciones.

La figura 4.5 ilustra un ejemplo del proceso de creación de un nuevo usuario. Se muestra la solicitud enviada desde Postman junto con la respuesta proporcionada por el servidor. Asimismo, en la interfaz de MongoDB Compass se visualiza el nuevo usuario incorporado en la colección User de la base de datos.



RA 4.5. Validación de almacenamiento en MongoDB.

4.4. Pruebas de frontend

Las pruebas realizadas al frontend se centraron en evaluar la seguridad, la usabilidad y la funcionalidad de la interfaz web. Se llevaron a cabo pruebas de autenticación, autorización y acceso a los distintos módulos del sistema. Se validó la correcta visualización de los datos, la interacción con los componentes y la capacidad de respuesta ante diferentes acciones del usuario.

Las validaciones incluyeron:

- Pruebas de autenticación y autorización: se verificó que los usuarios pudieran iniciar sesión correctamente y acceder a las funcionalidades correspondientes a su rol.

4.3. Pruebas de backend

47

La figura 4.5 permite visualizar las conexiones WebSocket realizadas mediante Postman.

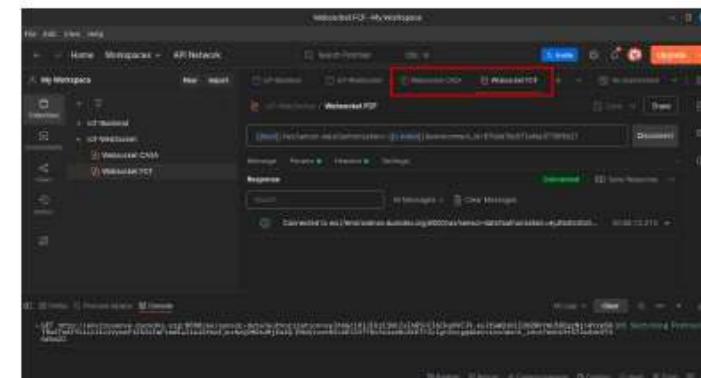


FIGURA 4.5. Pruebas de WebSocket desde Postman.

La figura 4.6 muestra una terminal de Linux conectada a una instancia de AWS EC2, donde se ejecuta el backend. En ella se puede observar la conexión activa de tres clientes WebSocket simultáneos.



FIGURA 4.6. Pruebas de WebSocket desde consola SSH en instancia EC2.

Las pruebas realizadas al canal WebSocket confirmaron la estabilidad de la comunicación, la capacidad del backend para distribuir mensajes de forma simultánea a múltiples clientes y la recepción inmediata de los datos.

4.3.3. Pruebas de validación de almacenamiento en MongoDB

Se llevaron a cabo pruebas para validar el almacenamiento de los datos en MongoDB, con el objetivo de comprobar que las operaciones de escritura realizadas desde el backend, tales como la creación de usuarios, sensores, mediciones y eventos, se registrarán en las colecciones correspondientes.

Para ello, se realizaron las siguientes acciones:

- Pruebas de funcionalidad: se validó que todas las funcionalidades implementadas en el frontend funcionaran correctamente.
- Pruebas de usabilidad: se evaluó la facilidad de uso de la interfaz y su responsividad, para garantizar que los usuarios pudieran navegar sin dificultades, realizar las acciones deseadas y acceder de manera óptima desde diferentes dispositivos y tamaños de pantalla.

4.4.1. Pruebas de autenticación y autorización

Las pruebas de autenticación y autorización se llevaron a cabo para garantizar que los usuarios pudieran iniciar sesión correctamente y acceder a las funcionalidades correspondientes a su rol. Se realizaron pruebas con diferentes tipos de usuarios, como administradores y usuarios estándar, para verificar que cada uno tuviera acceso a las funcionalidades adecuadas y que no pudieran acceder a las funciones restringidas a otros roles.

Pruebas de autenticación

La figura 4.6 muestra un ejemplo de la pantalla de inicio de sesión, donde se ingresan las credenciales de usuario. Para este ejemplo se ingresa un usuario `admin` y una contraseña incorrecta. Se puede observar que el sistema devuelve un mensaje de error en el cual indica que las credenciales son incorrectas.

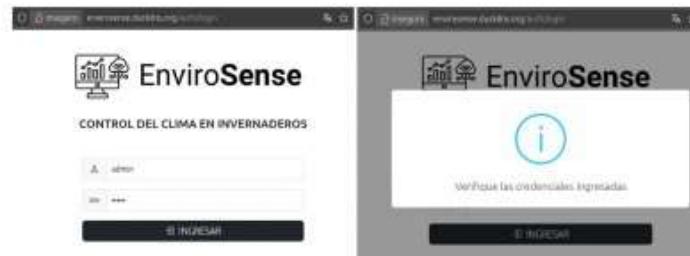


FIGURA 4.6. Pruebas de autenticación.

Pruebas de autenticación y autorización

Una vez que el usuario ingresa las credenciales correctas, se redirige a la pantalla principal del sistema, donde se pueden observar las distintas funcionalidades disponibles según el rol del usuario.

La figura 4.7 ilustra el proceso de inicio de sesión con un usuario `admin` y una contraseña válida. Al ingresar las credenciales correctas, el sistema redirige al usuario a la pantalla principal, donde se visualizan los módulos y funcionalidades disponibles, según el rol asignado. En este caso, el usuario tiene acceso completo a todas las opciones del sistema.

- Se ejecutaron operaciones desde Postman, para generar nuevas entidades en el sistema.
- A continuación, se accedió directamente a la base de datos MongoDB, a través de la herramienta MongoDB Compass, para inspeccionar las colecciones y verificar la existencia y la estructura de los documentos insertados.
- Se validó que las operaciones de actualización y eliminación impactaran correctamente sobre los documentos correspondientes en MongoDB.

La figura 4.7 ilustra un ejemplo del proceso de creación de un nuevo usuario. Se muestra la solicitud enviada desde Postman junto con la respuesta proporcionada por el servidor.

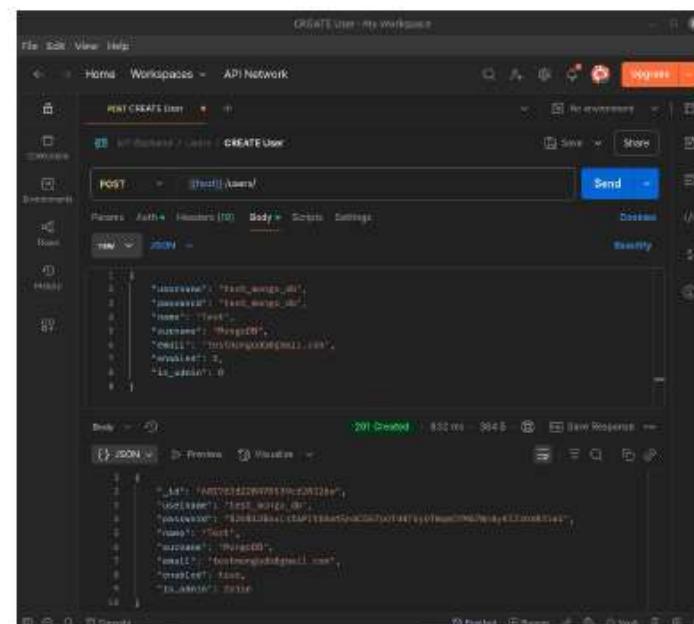


FIGURA 4.7. Creación de usuario desde Postman.

La figura 4.8 muestra la interfaz de MongoDB Compass, donde se visualiza el nuevo usuario incorporado en la colección `User` de la base de datos.

4.4. Pruebas de frontend

49



FIGURA 4.7. Autenticación y autorización de rol administrador.

La figura 4.8 muestra el proceso de inicio de sesión con un usuario estándar martini y una contraseña válida. Al ingresar las credenciales correctas, el sistema redirige al usuario a la pantalla principal, donde se pueden observar las distintas funcionalidades disponibles según el rol del usuario. En este caso, el usuario tiene acceso limitado a las opciones del sistema, lo que garantiza que solo pueda interactuar con las funciones disponibles para su rol.



FIGURA 4.8. Autenticación y autorización de rol usuario.

4.4.2. Pruebas de funcionalidad

Se realizaron pruebas exhaustivas para validar el correcto funcionamiento de todas las funcionalidades del frontend. Se verificó que cada módulo respondiera adecuadamente a las acciones del usuario y que los datos se visualizaran correctamente en la interfaz.

Validación de formularios

Se verificó que los formularios cumplieran con las restricciones definidas, como campos obligatorios, formatos específicos y caracteres mínimos. También se evaluó la correcta visualización de los mensajes de error ante entradas inválidas.

La figura 4.9 muestra un ejemplo del formulario de creación de usuario; en la imagen de la izquierda, el intento de envío con campos incompletos activa mensajes de advertencia; en la imagen de la derecha, al ingresar todos los datos requeridos, el sistema valida correctamente la información y muestra un mensaje de confirmación previo a la creación del nuevo usuario.

4.4. Pruebas de frontend

49

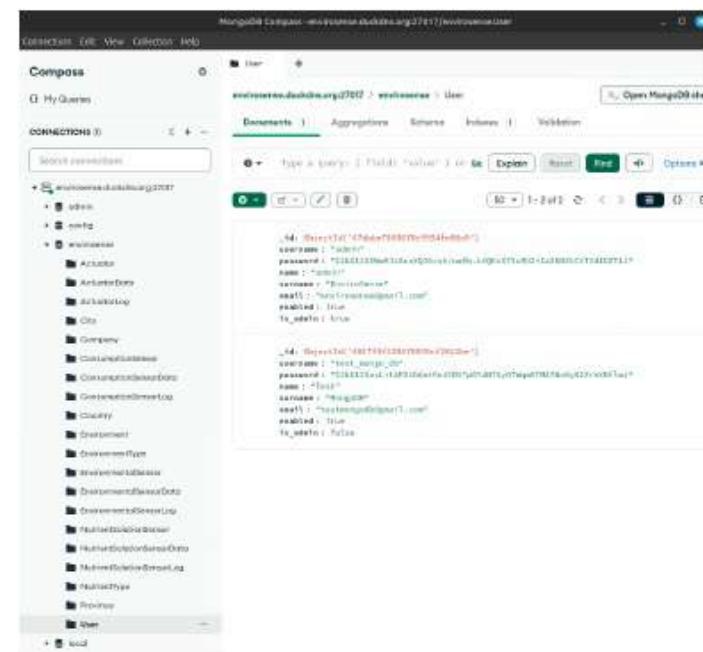


FIGURA 4.8. Validación del usuario creado en MongoDB.

Estas validaciones permitieron confirmar que el backend realiza una correcta persistencia de los datos en MongoDB y que no se observaron inconsistencias, ni pérdidas de información durante las operaciones.

4.4. Pruebas de frontend

Las pruebas realizadas al frontend se centraron en evaluar la seguridad, la usabilidad y la funcionalidad de la interfaz web. Se llevaron a cabo pruebas de autenticación, autorización y acceso a los distintos módulos del sistema. Se validó la correcta visualización de los datos, la interacción con los componentes y la capacidad de respuesta ante diferentes acciones del usuario.

Las validaciones incluyeron:

- Pruebas de autenticación y autorización: se verificó que los usuarios pudieran iniciar sesión correctamente y acceder a las funcionalidades correspondientes a su rol.
- Pruebas de funcionalidad: se validó que todas las funcionalidades implementadas en el frontend funcionaran correctamente.



RA 4.9. Validación de formularios.

La figura 4.10 permite verificar la creación del usuario **prueba** en la base de datos. El registro se visualiza en la tabla de usuarios, donde se reflejan correctamente todos los datos ingresados previamente en el formulario.

ID	NOMBRE DE...	NOMBRE	APELLIDO	CORREO E.I.	ES ADMIN	ESTADO	ACCIONES
Administrator...	admin	admin	Administrador	envirosense...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
00112520247...	martin	Martin	Lacheski	martinlache...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sistema...	prueba	usuario	prueba	correoprue...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

FIGURA 4.10. Verificación de creación de usuario.

Validación de tablas y gráficos

Se verificó la correcta visualización de las tablas, así como el funcionamiento de la paginación y el filtrado. También se evaluaron los gráficos, se comprueba su claridad, precisión y actualización dinámica frente a cambios en los datos.

La figura 4.11 presenta un ejemplo de los reportes del sensor ambiental. En la parte superior se encuentra el formulario para seleccionar sensor, rango de fechas y nivel de agregación; en el centro, la tabla con los datos generados; y abajo, el sistema de paginación para navegar los registros.

- Pruebas de usabilidad: se evaluó la facilidad de uso de la interfaz y su responsividad, para garantizar que los usuarios pudieran navegar sin dificultades, realizar las acciones deseadas y acceder desde diferentes dispositivos y tamaños de pantalla.

4.4.1. Pruebas de autenticación y autorización

Las pruebas de autenticación y autorización se llevaron a cabo para garantizar que los usuarios pudieran iniciar sesión correctamente y acceder a las funcionalidades correspondientes a su rol. Se realizaron pruebas con diferentes tipos de usuarios, como administradores y usuarios estándar, para verificar que cada uno tuviera acceso a las funcionalidades adecuadas y que no pudieran acceder a las funciones restringidas a otros roles.

Pruebas de autenticación

La figura 4.9 muestra un ejemplo de la pantalla de inicio de sesión, donde se ingresan las credenciales de usuario. Para este ejemplo se ingresa un usuario **admin** y una contraseña incorrecta. Se puede observar que el sistema devuelve un mensaje de error en el cual indica que las credenciales son incorrectas.

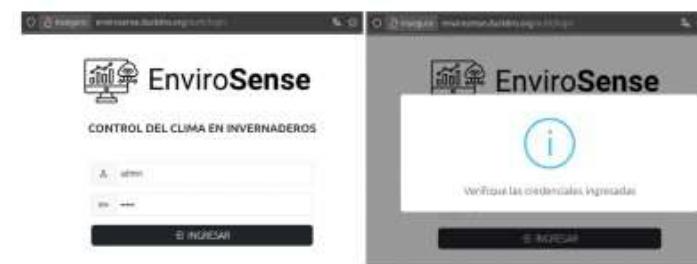


FIGURA 4.9. Pruebas de autenticación.

Pruebas de autenticación y autorización

Una vez que el usuario ingresa las credenciales correctas, se redirige a la pantalla principal del sistema, donde se pueden observar las distintas funcionalidades disponibles según el rol del usuario.

La figura 4.10 ilustra el proceso de inicio de sesión con un usuario **admin** y una contraseña válida. Al ingresar las credenciales correctas, el sistema redirige al usuario a la pantalla principal, donde se visualizan los módulos y funcionalidades disponibles, según el rol asignado. En este caso, el usuario tiene acceso completo a todas las opciones del sistema.

4.4. Pruebas de frontend

51

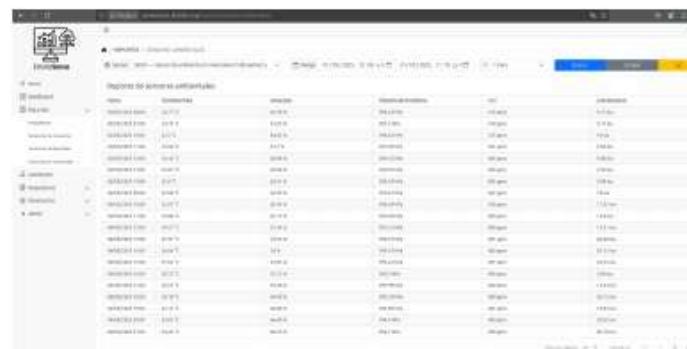


FIGURA 4.11. Pruebas de funcionalidad en tablas.

La figura 4.12 muestra un ejemplo de los gráficos generados por el sistema. Al igual que en la figura 4.11, en la parte superior se observa el formulario que permite seleccionar el sensor, el rango de fechas y el nivel de agregación para generar el gráfico. En la parte inferior se visualizan los gráficos generados, que representan la evolución de los datos a lo largo del tiempo.

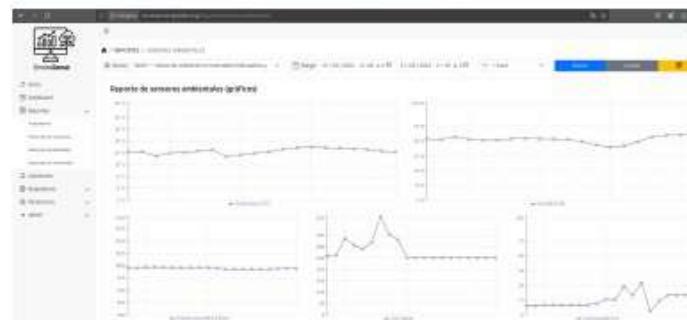


FIGURA 4.12. Pruebas de funcionalidad en gráficos.

Validación de gráficos con WebSocket

Se validó el funcionamiento del dashboard de monitoreo en tiempo real con WebSocket. Se comprobó que los valores sensados se reflejaran en la interfaz sin demoras y que los elementos visuales respondieran de forma fluida a la transmisión continua de datos.

La figura 4.13 muestra el dashboard, que incluye un menú desplegable en la parte superior derecha para seleccionar el ambiente a visualizar. En el centro se presentan los gráficos en tiempo real según el tipo de sensor. En la parte inferior se visualiza el cliente WebSocket en Postman, que confirma la correcta transmisión de datos desde el backend.

4.4. Pruebas de frontend

51



FIGURA 4.10. Autenticación y autorización de rol administrador.

La figura 4.11 muestra el proceso de inicio de sesión con un usuario estándar `martin` y una contraseña válida. Al ingresar las credenciales correctas, el sistema redirige al usuario a la pantalla principal, donde se pueden observar las distintas funcionalidades disponibles según el rol del usuario. En este caso, el usuario tiene acceso limitado a las opciones del sistema, lo que garantiza que solo pueda interactuar con las funciones disponibles para su rol.



FIGURA 4.11. Autenticación y autorización de rol usuario.

4.4.2. Pruebas de funcionalidad

Se realizaron pruebas exhaustivas para validar el correcto funcionamiento de todas las funcionalidades del frontend. Se verificó que cada módulo respondiera adecuadamente a las acciones del usuario y que los datos se visualizaran correctamente en la interfaz.

Validación de formularios

Se verificó que los formularios cumplieran con las restricciones definidas, como campos obligatorios, formatos específicos y caracteres mínimos. También se evaluó la correcta visualización de los mensajes de error ante entradas inválidas.

La figura 4.12 muestra un ejemplo del formulario de creación de usuario: en la imagen de la izquierda, el intento de envío con campos incompletos activa mensajes de advertencia; en la imagen de la derecha, al ingresar todos los datos requeridos, el sistema valida correctamente la información y muestra un mensaje de confirmación previo a la creación del nuevo usuario.



FIGURA 4.13. Pruebas de funcionalidad en el dashboard.

Pruebas de usabilidad

Se evaluó la facilidad de uso de la interfaz y su adaptación a distintos dispositivos, para asegurar una navegación fluida y acceso completo a las funciones desde pantallas de diferentes tamaños.

La figura 4.14 muestra un ejemplo de la interfaz de ambientes desde un dispositivo móvil. Se verificó que todos los elementos se adaptaron correctamente a la pantalla vertical, sin afectar su funcionalidad.

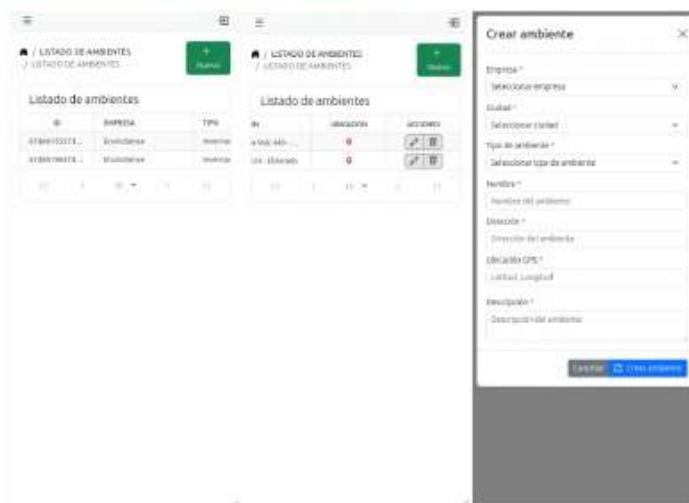


FIGURA 4.14. Interfaz de ambientes en dispositivo móvil.



FIGURA 4.12. Validación de formularios.

La figura 4.13 permite verificar la creación del usuario **prueba** en la base de datos. El registro se visualiza en la tabla de usuarios, donde se reflejan correctamente todos los datos ingresados previamente en el formulario.

ID	NOMBRE DE...	NOMBRE	APELLIDO	CORREO ELE...	ES ADMIN	ESTADO	ACCIONES
610000000001...	admin	admin	Lucheski	enviromet...@...	SI	Activo	
601111222247...	martin	Martin	Lucheski	matheles...@...	NO	Inactivo	
601111222248...	prueba	usuario	anata	correoque...@...	NO	Activo	

FIGURA 4.13. Verificación de creación de usuario.

Validación de tablas y gráficos

Se verificó la correcta visualización de las tablas, así como el funcionamiento de la paginación y el filtrado. También se evaluaron los gráficos, se comprobó su claridad, precisión y actualización dinámica frente a cambios en los datos.

La figura 4.14 presenta un ejemplo de los reportes del sensor ambiental. En la parte superior se encuentra el formulario para seleccionar el sensor, el rango de fechas y el nivel de agregación; en el centro, la tabla con los datos generados; y abajo, el sistema de paginación para navegar los registros.

4.5. Pruebas de componentes

53

La figura 4.15 presenta un listado visualizado en formato apaisado. También en este caso los componentes de la interfaz se ajustaron de forma adecuada, y se conservó la usabilidad.

The screenshot shows a mobile application interface titled "LISTADO DE AMBIENTES". Below it is a table titled "Listado de ambientes" with columns: ID, EMPRESA, TIPO, NOMBRE, and DIRECCIÓN. Two rows are visible:

ID	EMPRESA	TIPO	NOMBRE	DIRECCIÓN
679eb153373...	EnviroSense	Invernadero hidropón...	Invernadero hidropón...	Guillermo Vázquez 440 - ...
679eb1fe573...	EnviroSense	Invernadero hidropón...	Invernadero RFT en FCF	Bertoni 124 - El Dorado

At the bottom, there are pagination controls: "Filas por página: 15", "1-2 de 2", and icons for previous and next pages.

The screenshot shows the same mobile application interface as figure 4.15, but in portrait mode. The table structure remains the same, displaying the two environment entries from above.

FIGURA 4.15. Listado en formato apaisado desde dispositivo móvil.

4.5. Pruebas de componentes

Las pruebas se realizaron con el objetivo de validar el funcionamiento de cada componente y su integración en el entorno de prueba. Se llevaron a cabo verificaciones individuales sobre sensores y actuadores, para evaluar la precisión de las mediciones y la correcta respuesta ante comandos. Estas pruebas permitieron confirmar que cada módulo cumplió con los requisitos establecidos y operó de manera adecuada junto al resto del sistema.

Para llevarlas a cabo, los microcontroladores se conectaron a la computadora mediante un cable USB, lo que posibilitó monitorear la salida por comunicación serial y comprobar el comportamiento de cada módulo en forma directa.

4.5.1. Pruebas de sensores

Prueba de sensor ambiental

Se evaluaron los sensores ambientales BMP280, BH1750 y MH-Z19C, para medir temperatura ambiente, humedad relativa, presión atmosférica, nivel de iluminación y concentración de CO_2 en el entorno de prueba.

La figura 4.16 muestra un ejemplo de las mediciones registradas por el conjunto de sensores ambientales.

4.4. Pruebas de frontend

53

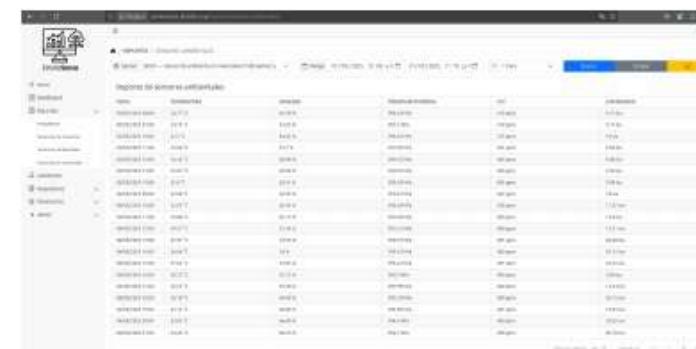


FIGURA 4.14. Pruebas de funcionalidad en tablas.

La figura 4.15 muestra un ejemplo de los gráficos generados por el sistema. Al igual que en la figura 4.14, en la parte superior se observa el formulario que permite seleccionar el sensor, el rango de fechas y el nivel de agregación para generar el gráfico. En la parte inferior se visualizan los gráficos generados, que representan la evolución de los datos a lo largo del tiempo.

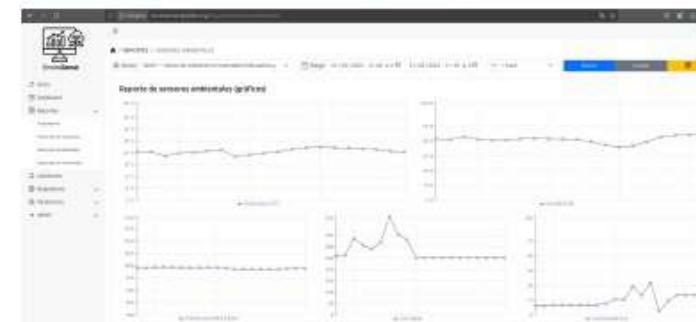


FIGURA 4.15. Pruebas de funcionalidad en gráficos.

Validación de gráficos con WebSocket

Se validó el funcionamiento del dashboard de monitoreo en tiempo real con WebSocket. Se comprobó que los valores sensados se reflejaran en la interfaz sin demoras y que los elementos visuales respondieran de forma fluida a la transmisión continua de datos.

La figura 4.16 muestra el dashboard, que incluye un menú desplegable en la parte superior derecha para seleccionar el ambiente a visualizar. En el centro se presentan los gráficos en tiempo real según el tipo de sensor. En la parte inferior se visualiza el cliente WebSocket en Postman, que confirma la correcta transmisión de datos desde el backend.



FIGURA 4.16. Pruebas de sensor ambiental.

Prueba de sensor de solución nutritiva

Se probaron los sensores que permiten medir el nivel de la solución, temperatura, pH, CE y TDS de la solución nutritiva. Se utilizaron los sensores DS18B20, PH-4502C, sensor de CE, sensor TDS y sensor ultrasónico HC-SR04.

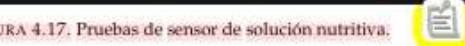
Para validar su funcionamiento, se realizaron dos pruebas:

- Solución con agua tibia y sal: se colocaron los sensores en un recipiente con agua tibia y sal, con el fin de evaluar las respuestas de pH, CE y TDS ante una solución más concentrada.
- Aqua potable natural: se repitió el procedimiento en un recipiente con agua potable natural, para comparar los resultados frente a una solución de menor concentración.

La figura 4.17 presenta los resultados obtenidos en estas pruebas.



FIGURA 4.17. Pruebas de sensor de solución nutritiva.



Prueba de sensor de consumos

Se verificó el funcionamiento de los sensores destinados al monitoreo de consumos eléctricos y del nivel en los depósitos de nutrientes. Para ello, se utilizaron los módulos PZEM-004T y sensores ultrasónicos HC-SR04.

La figura 4.18 muestra un ejemplo de los datos registrados durante las pruebas.



FIGURA 4.18. Pruebas de sensor de consumos.



FIGURA 4.16. Pruebas de funcionalidad en el dashboard.

Pruebas de usabilidad

Se evaluó la facilidad de uso de la interfaz y su adaptación a distintos dispositivos, para asegurar una navegación fluida y acceso completo a las funciones desde pantallas de diferentes tamaños.

La figura 4.17 muestra un ejemplo de la interfaz de ambientes desde un dispositivo móvil. Se verificó que todos los elementos se adaptaron correctamente a la pantalla vertical, sin afectar su funcionalidad.

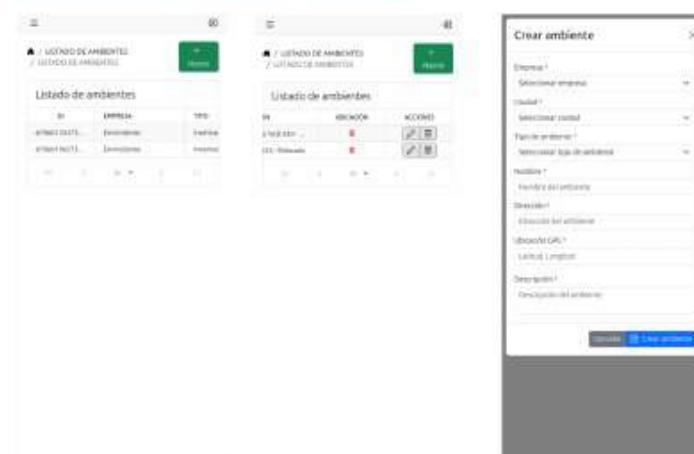


FIGURA 4.17. Interfaz de ambientes en dispositivo móvil.

La figura 4.18 presenta un listado visualizado en formato apaisado. También en este caso los componentes de la interfaz se ajustaron de forma adecuada, y se

4.6. Pruebas de comunicación

55

4.5.2. Pruebas de actuadores

Se realizaron pruebas de los módulos de relés, que permiten activar y desactivar dispositivos eléctricos. Se verificó su correcto funcionamiento al activar y desactivar los relés mediante comandos enviados.

La figura 4.19 muestra un ejemplo de la prueba realizada con el módulo de relé.



FIGURA 4.19. Pruebas de actuadores.

La figura 4.20 muestra un ejemplo de la prueba realizada con el módulo de relé. Se verifica el encendido y apagado de los relés mediante comandos enviados desde el microcontrolador.



FIGURA 4.20. Pruebas de encendido de relés.

4.6. Pruebas de comunicación

Las pruebas de comunicación tuvieron como objetivo validar el correcto funcionamiento de los canales de comunicación del sistema. Se verificaron los mensajes entre los microcontroladores y el servidor backend mediante el protocolo MQTT, para comprobar que los mensajes enviados fueron recibidos sin errores ni pérdidas.

Además, se evaluó la comunicación entre el backend y el frontend. Se confirmó el procesamiento adecuado de las solicitudes de la interfaz de usuario por el backend y la correcta presentación de las respuestas en el frontend.

Estas pruebas garantizaron la integridad del flujo de datos en toda la arquitectura del sistema, tanto en el envío como en la recepción de información.

4.6.1. Pruebas en sensor ambiental

La figura 4.21 muestra una prueba en la que se envió un mensaje MQTT al sensor ambiental para modificar el intervalo de envío de datos a 45 segundos.

4.5. Pruebas de componentes

55

conservó la usabilidad.

Listado de ambientes				
ID	EMPRESA	TIPO	NOMBRE	DIRECCIÓN
67deb153373...	EnviroSense	Invernadero hidropón...	Invernadero hidropón...	Guillermo Vázquez 430 - ...
67deb1f6573...	EnviroSense	Invernadero hidropón...	Invernadero NFT en PCF	Betox 124 - Edorado
Más por página: 15 ▾ 1-2 de 2				

Listado de ambientes				
TIPO	NOMBRE	DIRECCIÓN	UBICACIÓN	ACCIONES
Invernadero hidropón...	Invernadero hidropón...	Guillermo Vázquez 440 -	
Invernadero hidropón...	Invernadero NFT en PCF	Betox 124 - Edorado	...	
Más por página: 15 ▾ 1-2 de 2				

FIGURA 4.18. Listado en formato apaisado desde dispositivo móvil.

4.5. Pruebas de componentes

Las pruebas se realizaron con el objetivo de validar el funcionamiento de cada componente y su integración en el entorno de prueba. Se llevaron a cabo verificaciones individuales sobre sensores y actuadores, para evaluar la precisión de las mediciones y la correcta respuesta ante comandos. Estas pruebas permitieron confirmar que cada módulo cumplió con los requisitos establecidos y operó de manera adecuada junto al resto del sistema.

Para llevarlas a cabo, los microcontroladores se conectaron a la computadora mediante un cable USB, lo que posibilitó monitorear la salida por comunicación serial y comprobar el comportamiento de cada módulo en forma directa.

4.5.1. Pruebas de sensores

Prueba de sensor ambiental

Se evaluaron los sensores ambientales BMP280, BH1750 y MH-Z19C, para medir temperatura ambiente, humedad relativa, presión atmosférica, nivel de iluminación y concentración de CO_2 en el entorno de prueba.

La figura 4.19 muestra un ejemplo de las mediciones registradas por el conjunto de sensores ambientales.



```
POST /api/sensor/temperature HTTP/1.1
Host: 192.168.1.100:5000
Content-Type: application/json
Accept: application/json

{
    "sensor": "DS18B20_00000000000000000000000000000000",
    "value": 28.0
}
```

FIGURA 4.21. Pruebas de comunicación con sensor ambiental.

En la figura 4.22 se observa una prueba en la que se solicitó al sensor enviar sus datos al backend. El dispositivo procesó la solicitud y respondió con los valores capturados, los cuales fueron recibidos correctamente y almacenados en la base de datos.



```
GET /api/sensor/temperature HTTP/1.1
Host: 192.168.1.100:5000
Content-Type: application/json
Accept: application/json
```

FIGURA 4.22. Pruebas de comunicación con sensor ambiental.

4.6.2. Pruebas en sensor de consumos

La figura 4.23 presenta una prueba en la que se envió un mensaje MQTT al sensor de consumos para establecer un nuevo intervalo de transmisión de datos de 45 segundos.



```
mosquitto_sub -t "consumo/intervalo"
mosquitto_pub -t "consumo/intervalo" -m "45"
```

FIGURA 4.23. Pruebas de comunicación con sensor de consumos.

La figura 4.24 ilustra una prueba en la que se solicitó al sensor de consumos transmitir sus mediciones al backend. El sensor respondió con los datos requeridos, que fueron correctamente registrados en la base de datos.



```
POST /api/consumo/medida HTTP/1.1
Host: 192.168.1.100:5000
Content-Type: application/json
Accept: application/json

{
    "consumo": "PZEM-004T_00000000000000000000000000000000",
    "medida": "Consumo electrico",
    "valor": 100
}
```

FIGURA 4.24. Pruebas de comunicación con sensor de consumos.



```
SELECT * FROM consumo ORDER BY id DESC
```

FIGURA 4.19. Mediciones de sensor ambiental.

Prueba de sensor de solución nutritiva

Se probaron los sensores que permiten medir el nivel de la solución, temperatura, pH, CE y TDS de la solución nutritiva. Se utilizaron los sensores DS18B20, PH-4502C, sensor de CE, sensor TDS y sensor ultrasónico HC-SR04.

Para validar su funcionamiento, se realizaron dos pruebas:

- Solución con agua tibia y sal: se colocaron los sensores en un recipiente con agua tibia y sal, con el fin de evaluar las respuestas de pH, CE y TDS ante una solución más concentrada.
- Agua potable natural: se repitió el procedimiento en un recipiente con agua potable natural, para comparar los resultados frente a una solución de menor concentración.

La figura 4.20 presenta los resultados obtenidos en estas pruebas.



```
SELECT * FROM nutriente ORDER BY id DESC
```

FIGURA 4.20. Mediciones de sensor de solución nutritiva.

Prueba de sensor de consumos

Se verificó el funcionamiento de los sensores destinados al monitoreo de consumos eléctricos y del nivel en los depósitos de nutrientes. Para ello, se utilizaron los módulos PZEM-004T y sensores ultrasónicos HC-SR04.

La figura 4.21 muestra un ejemplo de los datos registrados durante las pruebas.



```
SELECT * FROM consumo ORDER BY id DESC
```

FIGURA 4.21. Mediciones de sensor de consumos.

4.6. Pruebas de comunicación

57

4.6.3. Pruebas en sensor de solución nutritiva

La figura 4.25 presenta una prueba en la que se envió un mensaje MQTT al sensor de solución nutritiva para fijar un nuevo intervalo de transmisión de datos en 45 segundos.



FIGURA 4.25. Pruebas de comunicación con sensor de solución nutritiva.

La figura 4.26 evidencia una prueba en la que se solicitó al sensor de solución nutritiva la transmisión de los valores de nivel, temperatura, pH, CE y TDS al backend. El sensor respondió con los datos requeridos, los cuales se registraron correctamente en la base de datos.



FIGURA 4.26. Pruebas de comunicación con sensor de solución nutritiva.

4.6.4. Pruebas de comunicación en el actuador

La figura 4.27 muestra una prueba en la que se envió un mensaje MQTT al actuador para establecer un nuevo intervalo de transmisión de datos de 45 segundos.



FIGURA 4.27. Pruebas de comunicación con actuador.

La figura 4.28 ilustra una prueba en la que se solicitó al actuador que transmitiera el estado de los relés al backend. El actuador respondió con los valores de cada relé, que fueron correctamente registrados en la base de datos.

4.6. Pruebas de comunicación

57

4.5.2. Pruebas de actuadores

Se realizaron pruebas de los módulos de relés, que permiten activar y desactivar dispositivos eléctricos. Se verificó su correcto funcionamiento al activar y desactivar los relés mediante comandos enviados.

La figura 4.22 muestra un ejemplo de la prueba realizada con el módulo de relé.



FIGURA 4.22. Estados de actuadores.

La figura 4.23 muestra un ejemplo de la prueba realizada con el módulo de relé. Se verifica el encendido y apagado de los relés mediante comandos enviados desde el microcontrolador.



FIGURA 4.23. Pruebas de encendido de relés.

4.6. Pruebas de comunicación

Las pruebas de comunicación tuvieron como objetivo validar el correcto funcionamiento de los canales de comunicación del sistema. Se verificaron los mensajes entre los microcontroladores y el servidor backend mediante el protocolo MQTT, para comprobar que los mensajes enviados fueron recibidos sin errores ni pérdidas.

Además, se evaluó la comunicación entre el backend y el frontend. Se confirmó el procesamiento adecuado de las solicitudes de la interfaz de usuario por el backend y la correcta presentación de las respuestas en el frontend.

Estas pruebas garantizaron la integridad del flujo de datos en toda la arquitectura del sistema, tanto en el envío como en la recepción de información.

4.6.1. Pruebas en sensor ambiental

La figura 4.24 muestra una prueba en la que se envió un mensaje MQTT al sensor ambiental para modificar el intervalo de envío de datos a 45 segundos.

```

$ mosquitto_sub -t "Relay/001/Status"
[...]
{
    "Topic": "Relay/001/Status",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'status': 'on'}"
}
[...]
$ mosquitto_pub -t "Relay/001/Control" -m '{"status": "off"}'
[...]
{
    "Topic": "Relay/001/Control",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'status': 'off'}"
}
[...]

```

FIGURA 4.28. Pruebas de comunicación con actuador.

La figura 4.29 permite visualizar una prueba en la que se envió un mensaje MQTT al actuador para activar el relé correspondiente a la bomba de agua durante cinco segundos. Al recibir el comando, el actuador ejecutó la acción y envió un mensaje con el estado de los relés. Finalizado el tiempo, transmitió un nuevo mensaje con el estado actualizado, y la confirmación de que la operación ha sido completada correctamente.

```

$ mosquitto_sub -t "Relay/001/Status"
[...]
{
    "Topic": "Relay/001/Status",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'status': 'on'}"
}
[...]
$ mosquitto_pub -t "Relay/001/Control" -m '{"status": "off"}'
[...]
{
    "Topic": "Relay/001/Control",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'status': 'off'}"
}
[...]

```

FIGURA 4.29. Pruebas de comunicación con actuador.

4.7. Prueba integral del sistema

Esta prueba tuvo como objetivo validar el funcionamiento completo del sistema, para asegurar la correcta interacción entre todos los componentes: frontend, backend y dispositivos conectados. Se buscó verificar la secuencia de comunicación entre las distintas partes sin interrupciones ni errores.

La prueba comenzó en el frontend, donde se seleccionó un actuador y se accedió a su detalle. A continuación, se activó el relé correspondiente a la bomba de agua por un período de 30 segundos.

La figura 4.30 muestra la interfaz del frontend al momento de ejecutar la orden de activación del relé.



FIGURA 4.30. Envío de comando de activación al microcontrolador.

```

$ mosquitto_sub -t "Relay/001/Status"
[...]
{
    "Topic": "Relay/001/Status",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'status': 'on'}"
}
[...]
$ mosquitto_pub -t "Relay/001/Control" -m '{"status": "off"}'
[...]
{
    "Topic": "Relay/001/Control",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'status': 'off'}"
}
[...]

```

FIGURA 4.24. Pruebas de comunicación en sensor ambiental.

En la figura 4.25 se observa una prueba en la que se solicitó al sensor enviar sus datos al backend. El dispositivo procesó la solicitud y respondió con los valores capturados, los cuales fueron recibidos correctamente y almacenados en la base de datos.

```

$ mosquitto_sub -t "Sensor/001/Status"
[...]
{
    "Topic": "Sensor/001/Status",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'status': 'on'}"
}
[...]
$ mosquitto_pub -t "Sensor/001/Control" -m '{"status": "off"}'
[...]
{
    "Topic": "Sensor/001/Control",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'status': 'off'}"
}
[...]

```

FIGURA 4.25. Solicitud de datos en sensor ambiental.

4.6.2. Pruebas en sensor de consumos

La figura 4.26 presenta una prueba en la que se envió un mensaje MQTT al sensor de consumos para establecer un nuevo intervalo de transmisión de datos de 45 segundos.

```

$ mosquitto_sub -t "Consumo/001/Status"
[...]
{
    "Topic": "Consumo/001/Status",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'status': 'on'}"
}
[...]
$ mosquitto_pub -t "Consumo/001/Control" -m '{"interval": 45}'
[...]
{
    "Topic": "Consumo/001/Control",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'interval': 45}"
}
[...]

```

FIGURA 4.26. Pruebas de comunicación en sensor de consumos.

La figura 4.27 ilustra una prueba en la que se solicitó al sensor de consumos transmitir sus mediciones al backend. El sensor respondió con los datos requeridos, que fueron correctamente registrados en la base de datos.

```

$ mosquitto_sub -t "Consumo/001/Status"
[...]
{
    "Topic": "Consumo/001/Status",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'status': 'on'}"
}
[...]
$ mosquitto_pub -t "Consumo/001/Control" -m '{"interval": 45}'
[...]
{
    "Topic": "Consumo/001/Control",
    "QoS": 1,
    "Retain": false,
    "Payload": "{'interval': 45}"
}
[...]

```

FIGURA 4.27. Solicitud de datos en sensor de consumos.

4.7. Prueba integral del sistema

59

Al hacer clic en la interfaz, el frontend envió una petición HTTP tipo POST al backend. La figura 4.31 muestra la salida por consola del frontend, donde se visualiza el envío de dicha solicitud.



FIGURA 4.31. Salida por consola del frontend.

El backend recibió esta solicitud, la procesó y publicó un mensaje MQTT en el tópico correspondiente. Luego respondió al frontend con un estado 200 OK, para indicar que la acción fue ejecutada correctamente.

En la figura 4.32 se observa la salida por consola del backend, el registro de la recepción de la solicitud, el envío del mensaje MQTT y la respuesta al frontend.



FIGURA 4.32. Salida por consola del backend.

El microcontrolador, suscripto al tópico correspondiente, recibió el mensaje, con el comando, el canal correspondiente y el tiempo de activación. Procesó la información y activó el relé de la bomba de agua.

Luego, envió una respuesta al backend con el estado actualizado de los relés. Una vez transcurrido el tiempo programado, volvió a enviar un nuevo mensaje para indicar que la operación fue completada con éxito.

La figura 4.33 muestra la consola del microcontrolador, donde se registra la recepción del mensaje, la activación del relé y la respuesta enviada.



FIGURA 4.33. Salida por consola del microcontrolador.

Posteriormente, el backend recibió el mensaje del microcontrolador y lo almacenó en la base de datos.

La figura 4.34 muestra la salida por consola del backend, donde se observa la recepción del mensaje del microcontrolador y el almacenamiento en la base de datos.

4.6. Pruebas de comunicación

59

4.6.3. Pruebas en sensor de solución nutritiva

La figura 4.28 presenta una prueba en la que se envió un mensaje MQTT al sensor de solución nutritiva para fijar un nuevo intervalo de transmisión de datos en 45 segundos.



FIGURA 4.28. Pruebas de comunicación en sensor de solución nutritiva.

La figura 4.29 evidencia una prueba en la que se solicitó al sensor de solución nutritiva la transmisión de los valores de nivel, temperatura, pH, CE y TDS al backend. El sensor respondió con los datos requeridos, los cuales se registraron correctamente en la base de datos.



FIGURA 4.29. Solicitud de datos en sensor de solución nutritiva.

4.6.4. Pruebas de comunicación en el actuador

La figura 4.30 muestra una prueba en la que se envió un mensaje MQTT al actuador para establecer un nuevo intervalo de transmisión de datos de 45 segundos.



FIGURA 4.30. Pruebas de comunicación en actuador.

La figura 4.31 ilustra una prueba en la que se solicitó al actuador que transmitiera el estado de los relés al backend. El actuador respondió con los valores de cada relé, que fueron correctamente registrados en la base de datos.

```
 1 //include module dependencies
 2 
 3 var express = require('express');
 4 var router = express.Router();
 5 var User = require('../models/user');
 6 var middleware = require('../middleware');
 7 
 8 //Route Handler - GET /users
 9 router.get('/', function(req, res) {
10   User.find({}, function(err, users) {
11     if (err) {
12       console.log(err);
13       res.status(500).send("There was a problem finding users");
14     } else {
15       res.json(users);
16     }
17   });
18 }
19 
20 //Route Handler - GET /users/:id
21 router.get('/:id', function(req, res) {
22   User.findById(req.params.id, function(err, user) {
23     if (err) {
24       console.log(err);
25       res.status(500).send("There was a problem finding user");
26     } else {
27       res.json(user);
28     }
29   });
30 }
31 
32 //Route Handler - POST /users
33 router.post('/', middleware.isLoggedIn, function(req, res) {
34   var newUser = {
35     username: req.body.username,
36     password: req.body.password,
37     email: req.body.email
38   };
39 
40   User.register(newUser, req.body.password, function(err, user) {
41     if (err) {
42       console.log(err);
43       res.status(500).send("There was a problem creating user");
44     } else {
45       //Log user in and redirect to dashboard
46       //Create cookie with user id and expire it after 300 days
47       var cookieExpire = Date.now() + 3000000000;
48       res.cookie('userId', user._id);
49       res.cookie('username', user.username);
50       res.cookie('password', user.password);
51       res.cookie('email', user.email);
52       res.cookie('expDate', cookieExpire);
53       req.logIn(user, function(err) {
54         if (err) {
55           console.log(err);
56           res.status(500).send("There was a problem logging in user");
57         } else {
58           res.redirect('/users/show/' + user._id);
59         }
60       });
61     }
62   });
63 }
64 
65 //Route Handler - PUT /users/:id
66 router.put('/:id', middleware.isLoggedIn, function(req, res) {
67   User.findByIdAndUpdate(req.params.id, {
68     $set: {
69       username: req.body.username,
70       password: req.body.password,
71       email: req.body.email
72     }
73   }, function(err, user) {
74     if (err) {
75       console.log(err);
76       res.status(500).send("There was a problem updating user");
77     } else {
78       res.json(user);
79     }
80   });
81 }
82 
83 //Route Handler - DELETE /users/:id
84 router.delete('/:id', middleware.isLoggedIn, function(req, res) {
85   User.findByIdAndDelete(req.params.id, function(err, user) {
86     if (err) {
87       console.log(err);
88       res.status(500).send("There was a problem deleting user");
89     } else {
90       res.json(user);
91     }
92   });
93 }
94 
```

FIGURA 4.34. Salida por consola del backend

Para validar el almacenamiento, se utilizó MongoDB Compass. En la figura 4.35 se muestra la colección `ActuatorData`, donde se evidencia el registro del estado de los relés y la marca temporal de la operación, para reflejar el encendido y apagado de la bomba de agua.



FIGURA 4.35. Verificación de persistencia en MongoDB Compass

Finalmente, el backend, envía un mensaje a todos los clientes conectados mediante WebSocket, para notificar la actualización del estado de los relés. El cliente recibe el mensaje y actualiza la interfaz, para reflejar el nuevo estado de los relés.

La figura 4.36 muestra el dashboard del frontend, donde se visualiza el estado actualizado de los relés, junto con la hora del sistema como referencia.

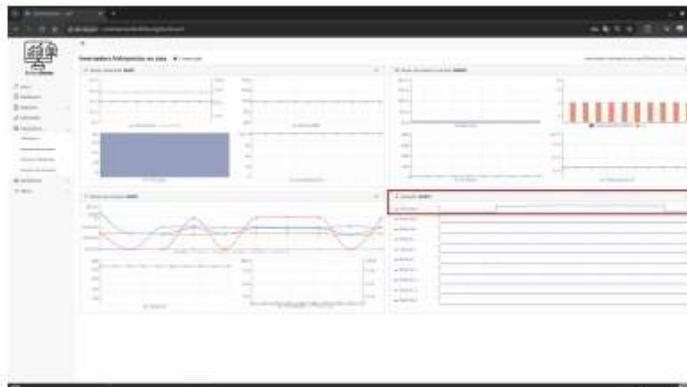


FIGURA 4.36. Actualización del estado de los relés en el frontend

FIGURA 4.31. Solicitud de datos en actuador

La figura 4.32 permite visualizar una prueba en la que se envió un mensaje MQTT al actuador para activar el relé correspondiente a la bomba de agua durante cinco segundos. Al recibir el comando, el actuador ejecutó la acción y envió un mensaje con el estado de los relés. Finalizado el tiempo, transmitió un nuevo mensaje con el estado actualizado, y la confirmación de que la operación ha sido completada correctamente.

FIGURA 4.32. Prueba de envío de comando en actuadores

4.7. Prueba integral del sistema

El objetivo de esta prueba fue validar el funcionamiento completo del sistema y asegurar la correcta interacción entre el frontend, el backend y los dispositivos conectados. Se verificó que la secuencia de comunicación entre los distintos componentes se realizara sin interrupciones ni errores.

La prueba se inició desde el frontend, donde se seleccionó un actuador, se accedió a su detalle y se activó el relé correspondiente a la bomba de agua durante 30 segundos. La figura 4.33 muestra la interfaz del frontend al momento de ejecutar esta orden.



FIGURA 4.33: Envío de comando de activación al microcontrolador.

Capítulo 5

Conclusiones

Este capítulo presenta los resultados obtenidos sobre el trabajo realizado. Además, se presentan mejoras como posibles trabajos futuros.

5.1. Conclusiones generales

Este trabajo alcanzó con éxito el objetivo de diseñar e implementar un prototipo funcional para el monitoreo y control remoto de condiciones climáticas en invernaderos, mediante tecnologías IoT y una arquitectura centrada en el usuario. Se integraron sensores y actuadores capaces de gestionar variables ambientales, parámetros de la solución nutritiva, consumos de energía y nutrientes, y el control de dispositivos a distancia, todo respaldado por una infraestructura en la nube, segura y escalable.

El sistema desarrollado permite:

- Registrar y consultar variables ambientales y parámetros del invernadero.
- Controlar actuadores desde una interfaz remota.
- Acceder a datos en tiempo real e históricos a través de una aplicación web responsive.

Cada componente desarrollado, desde los sensores y actuadores hasta el servidor y la aplicación web, cumplió con los requerimientos técnicos establecidos. La incorporación del protocolo MQTT con cifrado TLS, junto con el uso de servicios en la nube como AWS IoT Core y EC2, garantizó una comunicación segura, eficiente y escalable.

La aplicación web, diseñada con un enfoque centrado en la experiencia del usuario, resultó accesible, intuitiva y eficaz para la supervisión y el control del sistema. El sistema permite asignar distintos roles a los usuarios: el perfil administrador accede a todas las funciones, mientras que el usuario estándar tiene acceso exclusivo a la visualización del monitoreo en tiempo real y a los reportes. Esta diferenciación contribuye a una interacción segura y adaptada a distintos perfiles.

La interfaz gráfica, desarrollada con tecnologías modernas como React y Bootstrap, facilitó la interacción con el sistema, y su funcionalidad favoreció la toma de decisiones informadas y el aprovechamiento de los datos tanto en entornos académicos como productivos.

Las pruebas realizadas en un entorno controlado confirmaron la estabilidad, eficacia y adaptabilidad del sistema, y evidencian su potencial para mejorar la

4.7. Prueba integral del sistema

Al hacer clic en la interfaz, el frontend envió una petición HTTP tipo POST al backend. La figura 4.34 muestra la salida por consola del frontend, donde se visualiza el envío de dicha solicitud.



```
POST /api/reporting/555-00-00000-1000 HTTP/1.1
Content-Type: application/json
Accept: */*
Host: 127.0.0.1:8080
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win 10 X64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4285.120 Safari/537.36
Content-Length: 100
Content-Type: application/json; charset=UTF-8
{
    "id": "555-00-00000-1000",
    "temp": 25.5,
    "humid": 60.0,
    "light": 1000.0,
    "water": 100.0,
    "co2": 1000.0
}
```

FIGURA 4.34. Salida por consola del frontend.

El backend recibió esta solicitud, la procesó y publicó un mensaje MQTT en el tópico correspondiente. Luego respondió al frontend con un estado 200 OK, para indicar que la acción fue ejecutada correctamente.

En la figura 4.35 se observa la salida por consola del backend, el registro de la recepción de la solicitud, el envío del mensaje MQTT y la respuesta al frontend.



```
[2021-07-19 10:59:39] [INFO] [Frontend] Received message: {"id": "555-00-00000-1000", "temp": 25.5, "humid": 60.0, "light": 1000.0, "water": 100.0, "co2": 1000.0}
[2021-07-19 10:59:39] [INFO] [Backend] Publishing message: {"id": "555-00-00000-1000", "temp": 25.5, "humid": 60.0, "light": 1000.0, "water": 100.0, "co2": 1000.0} to topic: "iot/555-00-00000-1000"
[2021-07-19 10:59:39] [INFO] [Frontend] Response received: {"id": "555-00-00000-1000", "temp": 25.5, "humid": 60.0, "light": 1000.0, "water": 100.0, "co2": 1000.0}
```

FIGURA 4.35. Salida por consola del backend.

El microcontrolador, suscripto al tópico correspondiente, recibió el mensaje, con el comando, el canal correspondiente y el tiempo de activación. Procesó la información y activó el relé de la bomba de agua.

Luego, envió una respuesta al backend con el estado actualizado de los relés. Una vez transcurrido el tiempo programado, volvió a enviar un nuevo mensaje para indicar que la operación fue completada con éxito.

La figura 4.36 muestra la consola del microcontrolador, donde se registra la recepción del mensaje, la activación del relé y la respuesta enviada.



```
[2021-07-19 10:59:39] [INFO] [Frontend] Received message: {"id": "555-00-00000-1000", "temp": 25.5, "humid": 60.0, "light": 1000.0, "water": 100.0, "co2": 1000.0}
[2021-07-19 10:59:39] [INFO] [Microcontroller] Publishing message: {"id": "555-00-00000-1000", "temp": 25.5, "humid": 60.0, "light": 1000.0, "water": 100.0, "co2": 1000.0} to topic: "iot/555-00-00000-1000"
[2021-07-19 10:59:39] [INFO] [Frontend] Response received: {"id": "555-00-00000-1000", "temp": 25.5, "humid": 60.0, "light": 1000.0, "water": 100.0, "co2": 1000.0}
```

FIGURA 4.36. Salida por consola del microcontrolador.

Posteriormente, el backend recibió el mensaje del microcontrolador y lo almacenó en la base de datos.

La figura 4.37 muestra la salida por consola del backend, donde se observa la recepción del mensaje del microcontrolador y el almacenamiento en la base de datos.

eficiencia operativa, la sostenibilidad y la autonomía en la gestión de cultivos bajo condiciones controladas.

El desarrollo del prototipo permitió aplicar conocimientos en sistemas embebidos, comunicación segura, desarrollo web, gestión de bases de datos y despliegue de servicios en la nube. Esta experiencia constituye una base sólida para futuras investigaciones y desarrollos tecnológicos en el ámbito de la agricultura de precisión y el monitoreo ambiental.

5.2. Próximos pasos

A continuación, se proponen líneas de trabajo que permitirán continuar con el desarrollo y perfeccionar el sistema prototípico:

- Optimización del firmware: reescribir el firmware en lenguaje C con el entorno de Espressif, para mejorar el rendimiento, la eficiencia y la estabilidad del sistema.
- Actualizaciones OTA: incorporar un sistema de actualizaciones inalámbricas que permita aplicar mejoras sin intervención física.
- Automatización e inteligencia artificial: desarrollar un sistema de control automático basado en reglas y aplicar técnicas de inteligencia artificial para optimizar el uso de recursos y mejorar la toma de decisiones.
- Control local: implementar un sistema autónomo que funcione sin conexión a Internet, con capacidad de decisión in situ.
- Validación en entornos reales: ejecutar pruebas piloto en invernaderos productivos para evaluar el desempeño del sistema en condiciones operativas reales.
- Interfaz móvil: desarrollar una aplicación para dispositivos móviles que complemente la plataforma web y facilite el acceso en campo.
- Expansión funcional: ampliar el sistema con sensores adicionales y evaluar su aplicabilidad en otros contextos, como la gestión de recursos hídricos u otros cultivos.

```
Relé 1 encendido
Relé 2 apagado
Relé 3 apagado
Relé 4 apagado
Relé 5 apagado
Relé 6 apagado
Relé 7 apagado
Relé 8 apagado
Relé 9 apagado
Relé 10 apagado
Relé 11 apagado
Relé 12 apagado
Relé 13 apagado
Relé 14 apagado
Relé 15 apagado
Relé 16 apagado
Relé 17 apagado
Relé 18 apagado
Relé 19 apagado
Relé 20 apagado
Relé 21 apagado
Relé 22 apagado
Relé 23 apagado
Relé 24 apagado
Relé 25 apagado
Relé 26 apagado
Relé 27 apagado
Relé 28 apagado
Relé 29 apagado
Relé 30 apagado
Relé 31 apagado
Relé 32 apagado
Relé 33 apagado
Relé 34 apagado
Relé 35 apagado
Relé 36 apagado
Relé 37 apagado
Relé 38 apagado
Relé 39 apagado
Relé 40 apagado
Relé 41 apagado
Relé 42 apagado
Relé 43 apagado
Relé 44 apagado
Relé 45 apagado
Relé 46 apagado
Relé 47 apagado
Relé 48 apagado
Relé 49 apagado
Relé 50 apagado
Relé 51 apagado
Relé 52 apagado
Relé 53 apagado
Relé 54 apagado
Relé 55 apagado
Relé 56 apagado
Relé 57 apagado
Relé 58 apagado
Relé 59 apagado
Relé 60 apagado
Relé 61 apagado
Relé 62 apagado
Relé 63 apagado
Relé 64 apagado
Relé 65 apagado
Relé 66 apagado
Relé 67 apagado
Relé 68 apagado
Relé 69 apagado
Relé 70 apagado
Relé 71 apagado
Relé 72 apagado
Relé 73 apagado
Relé 74 apagado
Relé 75 apagado
Relé 76 apagado
Relé 77 apagado
Relé 78 apagado
Relé 79 apagado
Relé 80 apagado
Relé 81 apagado
Relé 82 apagado
Relé 83 apagado
Relé 84 apagado
Relé 85 apagado
Relé 86 apagado
Relé 87 apagado
Relé 88 apagado
Relé 89 apagado
Relé 90 apagado
Relé 91 apagado
Relé 92 apagado
Relé 93 apagado
Relé 94 apagado
Relé 95 apagado
Relé 96 apagado
Relé 97 apagado
Relé 98 apagado
Relé 99 apagado
Relé 100 apagado
```

FIGURA 4.37. Salida por consola del backend.

Para validar el almacenamiento, se utilizó MongoDB Compass. En la figura 4.38 se muestra la colección `ActuatorData`, donde se evidencia el registro del estado de los relés y la marca temporal de la operación, para reflejar el encendido y apagado de la bomba de agua.



FIGURA 4.38. Verificación de persistencia en MongoDB Compass.

Finalmente, el backend, envía un mensaje a todos los clientes conectados mediante WebSocket, para notificar la actualización del estado de los relés. El cliente recibe el mensaje y actualiza la interfaz, para reflejar el nuevo estado de los relés.

La figura 4.39 muestra el dashboard del frontend, donde se visualiza el estado actualizado de los relés, junto con la hora del sistema como referencia.

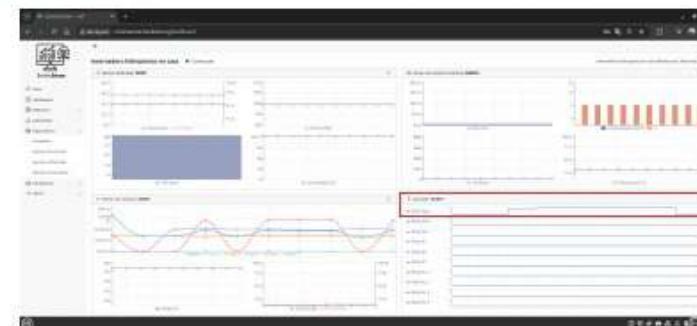


FIGURA 4.39. Actualización del estado de los relés en el frontend.

Apéndice A

Modelo de datos implementado en el trabajo

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

Capítulo 5

Conclusiones

Este capítulo presenta los resultados obtenidos sobre el trabajo realizado. Además, se presentan mejoras como posibles trabajos futuros.

5.1. Conclusiones generales

Este trabajo logró con éxito el objetivo de diseñar e implementar un prototipo funcional para el monitoreo y control remoto de condiciones climáticas en invernaderos, mediante tecnologías IoT y una arquitectura centrada en el usuario.

Se integraron sensores y actuadores capaces de relevar variables ambientales, parámetros de la solución nutritiva, consumos de energía y nutrientes, así como de controlar dispositivos a distancia, todo respaldado por una infraestructura en la nube, segura y escalable.

El sistema desarrollado permite:

- Registrar y consultar variables ambientales y parámetros del invernadero.
- Controlar actuadores desde una interfaz remota.
- Acceder a datos en tiempo real e históricos a través de una aplicación web responsive.

Cada componente desarrollado, desde los sensores y actuadores hasta el servidor y la aplicación web, cumplió con los requerimientos establecidos. La incorporación del protocolo MQTT con cifrado TLS, junto con el uso de servicios en la nube como AWS IoT Core y EC2, garantizó una comunicación segura, eficiente y escalable.

La aplicación web, diseñada con un enfoque centrado en la experiencia del usuario, resultó accesible, intuitiva y eficaz para la supervisión y el control del sistema. El sistema permite asignar distintos roles a los usuarios: el perfil administrador accede a todas las funciones, mientras que el usuario estándar tiene acceso exclusivo a la visualización del monitoreo en tiempo real y a los reportes. Esta diferenciación contribuye a una interacción segura y adaptada a distintos perfiles.

La interfaz gráfica, desarrollada con tecnologías modernas como React y Bootstrap, facilitó la interacción con el sistema, y su funcionalidad favoreció la toma de decisiones informadas y el aprovechamiento de los datos tanto en entornos académicos como productivos.

Las pruebas realizadas en un entorno controlado confirmaron la estabilidad, eficacia y adaptabilidad del sistema, y evidencian su potencial para mejorar la

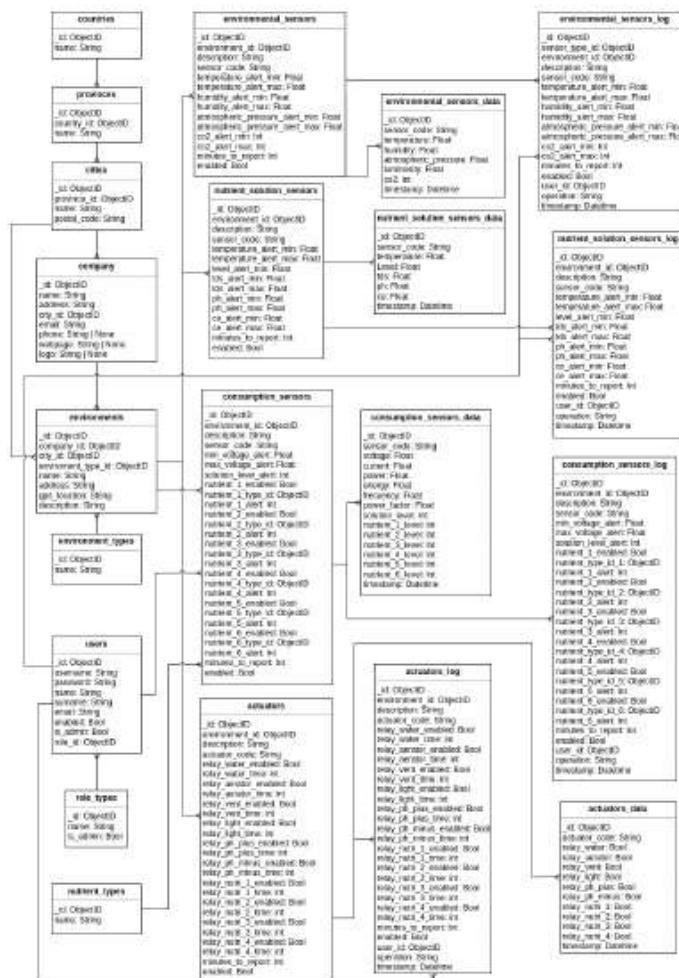


FIGURA A.1. Modelo de datos implementado en el trabajo.

eficiencia operativa, la sostenibilidad y la autonomía en la gestión de cultivos bajo condiciones controladas.

El desarrollo del prototipo permitió aplicar conocimientos en sistemas embebidos, comunicación segura, desarrollo web, gestión de bases de datos y despliegue de servicios en la nube. Esta experiencia constituye una base sólida para futuras investigaciones y desarrollos tecnológicos en el ámbito de la agricultura de precisión y el monitoreo ambiental.

5.2. Próximos pasos

A continuación, se proponen líneas de trabajo que permitirán continuar con el desarrollo y perfeccionar el sistema prototipado:

- Optimización del firmware: reescribir el firmware en lenguaje C con el entorno de Espressif, para mejorar el rendimiento, la eficiencia y la estabilidad del sistema.
- Migración a LoRaWAN: Reemplazar la conexión Wi-Fi por una red LoRaWAN para ampliar la cobertura en entornos remotos, reducir el consumo energético y escalar el número de nodos conectados.
- Actualizaciones OTA: incorporar un sistema de actualizaciones inalámbricas que permita aplicar mejoras sin intervención física.
- Automatización e inteligencia artificial: desarrollar un sistema de control automático basado en reglas y aplicar técnicas de inteligencia artificial para optimizar el uso de recursos y mejorar la toma de decisiones.
- Control local: implementar un sistema autónomo que funcione sin conexión a Internet, con capacidad de decisión in situ.
- Validación en entornos reales: ejecutar pruebas piloto en invernaderos productivos para evaluar el desempeño del sistema en condiciones operativas reales.
- Gestión de roles y permisos: implementar la creación de roles y la asignación de permisos específicos a cada usuario.
- Aplicación móvil: desarrollar una app para dispositivos móviles que complementa la aplicación web y optimice la experiencia del usuario.
- Expansión funcional: ampliar el sistema con sensores adicionales y evaluar su aplicabilidad en otros contextos, como la gestión de recursos hídricos u otros cultivos.

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	/mqqt/test	Test conexión cliente MQTT
POST	/mqqt/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

Apéndice A

Modelo de datos implementado en el trabajo

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

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

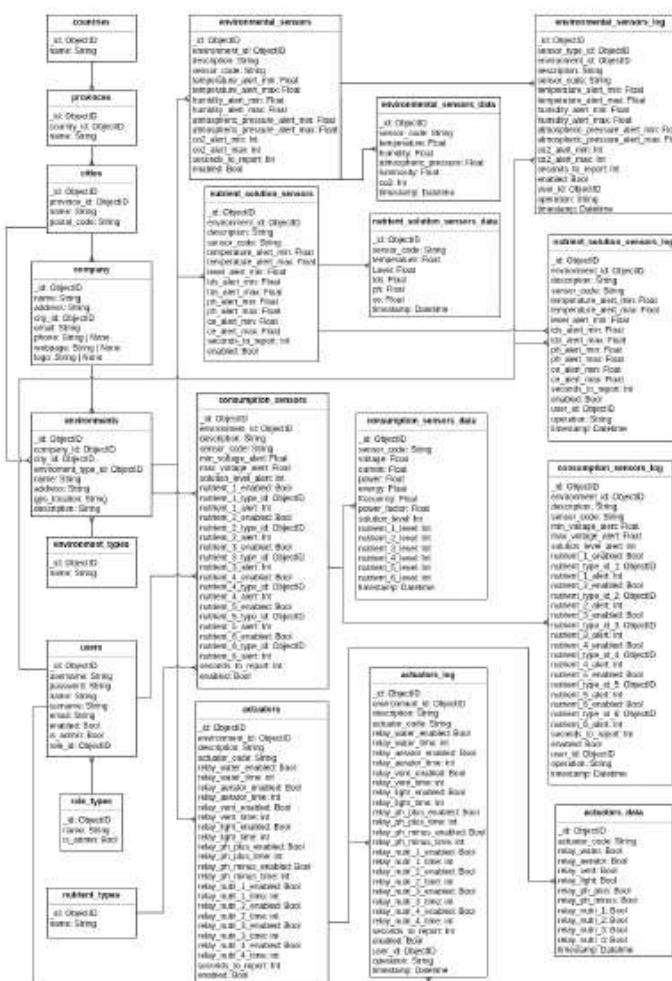


FIGURA A.1. Modelo de datos implementado en el trabajo.

Apéndice B. Resumen de endpoints de la API

67

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
 GET	/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
 GET	/sensors/nutrients/solution/data/	Crear dato histórico
GET	/sensors/nutrients/solution/data/{id}	Obtener un dato histórico

67

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. Endpoints esenciales: autenticación, gestión de usuarios y configuración del sistema.

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.

TABLA B.2. Endpoints operativos: control de ambientes, actuadores y monitoreo de consumos.

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.

Apéndice C

Despliegue del sistema en instancia EC2

C.1. Introducción

El presente anexo documenta el procedimiento realizado para la instalación, configuración y puesta en funcionamiento del entorno de servidor utilizado para alojar la solución EnviroSenseIoT. Esta solución se desplegó sobre una instancia virtual en Amazon EC2, se utilizaron servicios de infraestructura en la nube para garantizar la disponibilidad, escalabilidad y acceso remoto seguro al sistema.

Se detallan paso a paso las acciones ejecutadas para:

- Crear una instancia EC2 en Amazon Web Services (AWS) con Ubuntu 24.04 con sus respectivas claves de acceso.
- Configurar los puertos de red para permitir el acceso a los servicios necesarios.
- Acceder a la instancia EC2 mediante una conexión SSH segura.
- Instalar y configurar Docker como entorno de ejecución para contenedores.
- Incorporar Docker Compose para la orquestación de servicios.
- Configurar un cliente de actualización dinámica de DNS (DuckDNS) que permita asociar un nombre de dominio estático a la IP dinámica de la instancia.
- Descargar el repositorio del proyecto, crear archivos de configuración necesarios y establecer los certificados de seguridad requeridos para el funcionamiento del backend.
- Configurar la ejecución del sistema como servicio para garantizar su disponibilidad y facilitar su gestión.

C.2. Creación de la instancia EC2

Para crear la instancia EC2, se siguieron los pasos detallados a continuación:

1. Ingresar a la consola de AWS y seleccionar EC2.
2. Hacer clic en "Lanzar una instancia".
3. Elegir la imagen de Ubuntu Server 24.04 LTS (HVM), SSD Volume Type.

TABLA B.3. Endpoints de monitoreo ambiental y de nutrientes.

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.

4. Elegir el tipo de instancia: t2.micro.
5. Configurar los detalles de la instancia:
 - Número de instancias: 1
 - Configuración de red: VPC predeterminada.
 - Subred: subred predeterminada.
 - Asignar IP pública: sí.
6. Configurar almacenamiento:
 - Tamaño y tipo de disco: 8 GB gp3.
7. Crear un nuevo grupo de seguridad con las siguientes reglas:
 - a) SSH, TCP/22, Origen: 0.0.0.0/0.
 - b) HTTP, TCP/80, Origen: 0.0.0.0/0.
 - c) HTTPS, TCP/443, Origen: 0.0.0.0/0.
 - d) WebSocket, TCP/8000, Origen: 0.0.0.0/0.
 - e) MongoDB, TCP/27017, Origen: 0.0.0.0/0.
8. Generar una nueva clave de acceso:
 - Nombre de la clave: envirosense-app-key.
 - Formato de clave: PEM.
 - Descargar la clave privada ([envirosense-app-key.pem](#)) y resguardarla en un lugar seguro.

La figura C.1 muestra el resumen de la instancia creada.

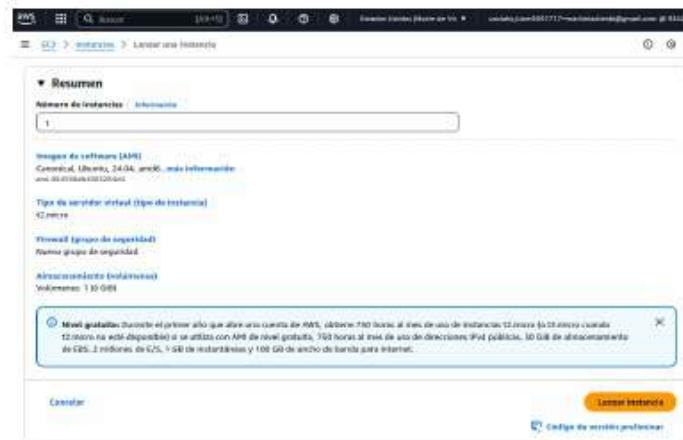


FIGURA C.1. Resumen de la instancia EC2 creada.

C.3. Instalación y configuración del servidor EC2

Una vez creada la instancia EC2, se procedió a su configuración e instalación de los servicios necesarios para el funcionamiento del sistema EnviroSenseIoT. Para ello, se siguieron los siguientes pasos:

C.3.1. Conectarse a la instancia EC2

Para establecer una conexión con una instancia EC2 mediante SSH, primero se debe abrir una terminal y ubicar la clave privada `envirosense-app-key.pem`, que será utilizada para la autenticación. Es fundamental restringir los permisos del archivo para que solo el propietario tenga acceso de lectura, de modo que el sistema SSH lo acepte como válido. Esto se logra al ejecutar el siguiente comando:

```
chmod 400 envirosense-app-key.pem
```

Una vez configurados los permisos, se puede establecer la conexión al utilizar el DNS público de la instancia. Por ejemplo:

```
ssh -i "key.pem" ubuntu@ec2-X.compute-1.amazonaws.com
```

La figura C.2 muestra la conexión SSH a la instancia EC2.

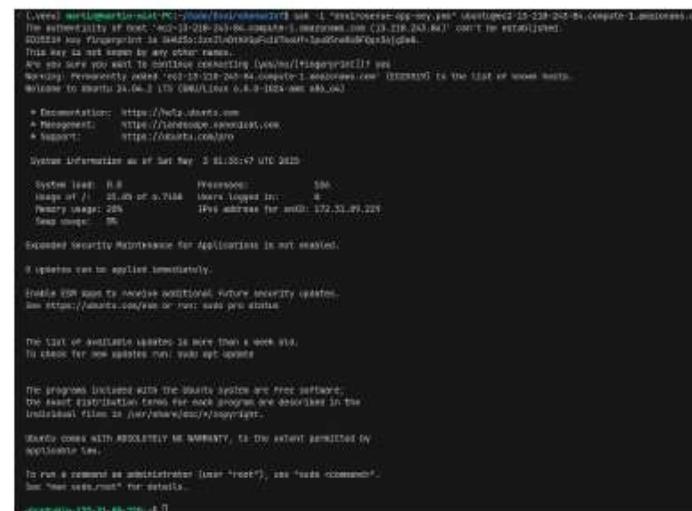


FIGURA C.2. Conexión SSH a la instancia EC2.

Una vez que se ingresó a la instancia EC2, se actualizaron los paquetes del sistema con los comandos:

```
sudo apt update
sudo apt upgrade
```

Apéndice C

Despliegue del sistema en instancia EC2

C.1. Introducción

El presente anexo documenta el procedimiento realizado para la instalación, configuración y puesta en funcionamiento del entorno de servidor utilizado para alojar la solución EnviroSenseIoT. Esta solución se desplegó sobre una instancia virtual en Amazon EC2, se utilizaron servicios de infraestructura en la nube para garantizar la disponibilidad, escalabilidad y acceso remoto seguro al sistema.

Se detallan paso a paso las acciones ejecutadas para:

- Crear una instancia EC2 en Amazon Web Services (AWS) con Ubuntu 24.04 con sus respectivas claves de acceso.
- Configurar los puertos de red para permitir el acceso a los servicios necesarios.
- Acceder a la instancia EC2 mediante una conexión SSH segura.
- Instalar y configurar Docker como entorno de ejecución para contenedores.
- Incorporar Docker Compose para la orquestación de servicios.
- Configurar un cliente de actualización dinámica de DNS (DuckDNS) que permita asociar un nombre de dominio estático a la IP dinámica de la instancia.
- Descargar el repositorio del proyecto, crear archivos de configuración necesarios y establecer los certificados de seguridad requeridos para el funcionamiento del backend.
- Configurar la ejecución del sistema como servicio para garantizar su disponibilidad y facilitar su gestión.

C.2. Creación de la instancia EC2

Para crear la instancia EC2, se siguieron los pasos detallados a continuación:

1. Ingresar a la consola de AWS y seleccionar EC2.
2. Hacer clic en "Lanzar una instancia".
3. Elegir la imagen de Ubuntu Server 24.04 LTS (HVM), SSD Volume Type.

C.3.2. Instalación de Docker

Para instalar Docker en Ubuntu, se siguió el tutorial oficial disponible en <https://docs.docker.com/engine/install/ubuntu/>. A continuación, se detallan los pasos realizados:

En primer lugar, se actualizó el sistema y se instalaron las dependencias necesarias:

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg
  -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Luego, se agregó el repositorio de Docker a las fuentes de APT y se actualizó nuevamente el sistema:

```
echo \'
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt
  /keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release &&
  echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

A continuación, se procedió a instalar Docker y sus componentes esenciales, y se verificó la instalación mediante los siguientes comandos:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
sudo docker run hello-world
```

Para permitir el uso de Docker sin la necesidad de utilizar sudo, se agregó el usuario al grupo docker con los siguientes comandos:

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
docker run hello-world
```

Por último, se configuraron los servicios de Docker y containerd para que se inicien automáticamente al arrancar el sistema, mediante los siguientes comandos:

```
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
```

C.3.3. Instalación de Docker Compose

Para instalar Docker Compose, se siguieron los siguientes pasos: Se descargó la versión más reciente de Docker Compose desde el repositorio oficial de GitHub. En este caso, se utilizó la versión 2.17.2, que es la última versión estable al momento de la instalación. Se utilizó el siguiente comando para descargar el binario de Docker Compose y guardararlo en la ruta /usr/local/bin/docker-compose:

4. Elegir el tipo de instancia: t2.micro.

5. Configurar los detalles de la instancia:

- Número de instancias: 1
- Configuración de red: VPC predeterminada.
- Subred: subred predeterminada.
- Asignar IP pública: si.

6. Configurar almacenamiento:

- Tamaño y tipo de disco: 8 GB gp3.

7. Crear un nuevo grupo de seguridad con las siguientes reglas:

- a) SSH, TCP/22, Origen: 0.0.0.0/0.
- b) HTTP, TCP/80, Origen: 0.0.0.0/0.
- c) HTTPS, TCP/443, Origen: 0.0.0.0/0.
- d) WebSocket, TCP/8000, Origen: 0.0.0.0/0.
- e) MongoDB, TCP/27017, Origen: 0.0.0.0/0.

8. Generar una nueva clave de acceso:

- Nombre de la clave: envirosense-app-key.
- Formato de clave: PEM.
- Descargar la clave privada ([envirosense-app-key.pem](#)) y resguardarla en un lugar seguro.

La figura C.1 muestra el resumen de la instancia creada.

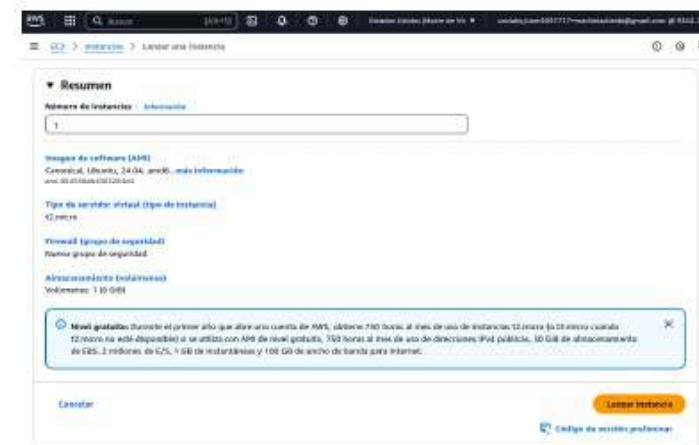


FIGURA C.1. Resumen de la instancia EC2 creada.

C.3. Instalación y configuración del servidor EC2

73

```
sudo curl -sSL
https://github.com/docker/compose/releases/download/
v2.17.2/docker-compose-linux-x86_64
-a /usr/local/bin/docker-compose
```

Luego, se asignaron permisos de ejecución al binario de Docker Compose para que pueda ser ejecutado con el siguiente comando:

```
sudo chmod +x /usr/local/bin/docker-compose
```

A continuación, se verificó que la instalación fuera exitosa mediante:

```
docker-compose --version
```

Finalmente, se creó un enlace simbólico para facilitar su ejecución desde cualquier ruta del sistema:

```
sudo ln -s /usr/local/bin/docker-compose
/usr/bin/docker-compose
```

C.3.4. Instalación del cliente DuckDNS

Para comenzar con la instalación del cliente DuckDNS en la instancia EC2, se instalaron los paquetes necesarios mediante los siguientes comandos:

```
sudo apt-get update && sudo apt-get install cloud-utils -y
sudo apt install amazon-ec2-utils
```

A continuación, se siguieron las instrucciones proporcionadas por el sitio oficial de DuckDNS, específicamente en la sección para instancias EC2.

Luego, se creó un archivo llamado `duck.sh` en el directorio raíz del servidor. Este script se encarga de verificar si la dirección IP pública ha cambiado y, en caso afirmativo, actualizar el registro correspondiente en DuckDNS. El contenido del archivo es el siguiente:

```
#!/bin/bash
current_ip=""
while true; do
latest_ip=$(curl -Ss https://checkip.amazonaws.com | grep -oE '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' | cut -d " " -f 2)
[ $current_ip != "$latest_ip" ] && latest_ip=$(curl -s ifconfig.me)
echo "Current IP: $current_ip | Latest IP: $latest_ip"
if [ "$current_ip" == "$latest_ip" ]; then
echo "IP no ha cambiado."
else
echo "IP ha cambiado. Se actualiza DuckDNS..."
current_ip="$latest_ip"
response=$(curl -s -k "https://www.duckdns.org/update?
domains=envirosense&token=TOKEN&ip=$latest_ip")
echo "DuckDNS response: $response" >> ~/duckdns/duck.log
fi
sleep 5m
done
```

Este script se ejecuta de forma continua, verifica cada cinco minutos si se produjo un cambio en la IP pública de la instancia.

C.3. Instalación y configuración del servidor EC2

73

C.3. Instalación y configuración del servidor EC2

Una vez creada la instancia EC2, se procedió a su configuración e instalación de los servicios necesarios para el funcionamiento del sistema EnviroSenseIoT. Para ello, se siguieron los siguientes pasos:

C.3.1. Conectarse a la instancia EC2

Para establecer una conexión con una instancia EC2 mediante SSH, primero se debe abrir una terminal y ubicar la clave privada `envirosense-app-key.pem`, que será utilizada para la autenticación. Es fundamental restringir los permisos del archivo para que solo el propietario tenga acceso de lectura, de modo que el sistema SSH lo acepte como válido. Esto se logra al ejecutar el siguiente comando:

```
chmod 400 envirosense-app-key.pem
```

Una vez configurados los permisos, se puede establecer la conexión al utilizar el DNS público de la instancia. Por ejemplo:

```
ssh -i "key.pem" ubuntu@ec2-X.compute-1.amazonaws.com
```

La figura C.2 muestra la conexión SSH a la instancia EC2.

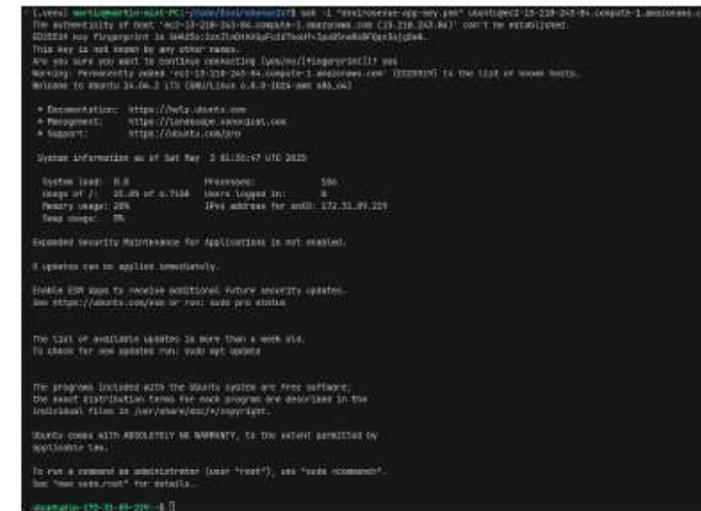


FIGURA C.2. Conexión SSH a la instancia EC2.

Una vez que se ingresó a la instancia EC2, se actualizaron los paquetes del sistema con los comandos:

```
sudo apt update
sudo apt upgrade
```

C.3.5. Descarga y configuración del repositorio EnviroSense

Para poner en marcha el sistema, primero se clonó el repositorio del código fuente desde GitHub y se accedió al directorio correspondiente con los siguientes comandos:

```
git clone
https://github.com/martinlacheski/EnviroSenseIoT.git
cd EnviroSenseIoT
```

A continuación, se ingresó al directorio EnviroSenseIoT y se creó el archivo .env, donde se definieron las variables de entorno necesarias para el correcto funcionamiento del sistema. Para ello, se utilizó el siguiente comando:

```
nano .env
```

En este archivo se incluyó la configuración vinculada a la conexión con la base de datos MongoDB, claves JWT para autenticación, y parámetros específicos tanto del backend como del frontend.

El contenido del archivo .env tiene los siguientes parámetros:

```
# Conexión MongoDB
MONGO_HOST=host # Dirección IP o nombre de host de MongoDB
MONGO_USER=user # Nombre de usuario para la conexión
MONGO_PASSWORD=password # Contraseña para la conexión
MONGO_DB=database # Nombre de la base de datos

# Conexión Backend
BACKEND_HOST=0.0.0.0 # Permitir conexión desde cualquier IP
BACKEND_CORS_ORIGINS=origins # Orígenes permitidos

# Configuración de JWT
# Generar una clave secreta con el comando:
# openssl rand -hex 32
BACKEND_SECRET_KEY=clave # Clave secreta generada
BACKEND_ALGORITHM="HS256" # Algoritmo de firma
BACKEND_ACCESS_TOKEN_EXPIRE_MINUTES=120 # Expiración token

# Configuración de Frontend
VITE_PWD_SIGNUP_ENABLED=true
VITE_BACKEND_API_URL=url # URL del backend
VITE_BACKEND_SOCKET_URL=url # URL del socket
```

Luego, se accedió al directorio backend del proyecto y se creó una carpeta llamada certificates, destinada a almacenar los certificados necesarios para las conexiones seguras:

```
cd backend
mkdir certificates
```

Dentro de esta carpeta, se generaron los archivos client.crt, client.key y root.crt, que contienen los certificados y la clave privada requeridos por AWS IoT Core para establecer comunicaciones seguras vía MQTT con TLS.

C.3.2. Instalación de Docker

Para instalar Docker en Ubuntu, se siguió el tutorial oficial disponible en <https://docs.docker.com/engine/install/ubuntu/>. A continuación, se detallan los pasos realizados:

En primer lugar, se actualizó el sistema y se instalaron las dependencias necesarias:

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg
> /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Luego, se agregó el repositorio de Docker a las fuentes de APT y se actualizó nuevamente el sistema:

```
echo N
"deb [arch=$dpkg --print-architecture] signed-by=/etc/apt
/keyrings/docker.asc] N
https://download.docker.com/linux/ubuntu N
$(> /etc/os-release &&
echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}" stable" | N
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

A continuación, se procedió a instalar Docker y sus componentes esenciales, y se verificó la instalación mediante los siguientes comandos:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
sudo docker run hello-world
```

Para permitir el uso de Docker sin la necesidad de utilizar sudo, se agregó el usuario al grupo docker con los siguientes comandos:

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
docker run hello-world
```

Por último, se configuraron los servicios de Docker y containerd para que se inicien automáticamente al arrancar el sistema, mediante los siguientes comandos:

```
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
```

C.3.3. Instalación de Docker Compose

Para instalar Docker Compose, se siguieron los siguientes pasos: Se descargó la versión más reciente de Docker Compose desde el repositorio oficial de GitHub. En este caso, se utilizó la versión 2.17.2, que es la última versión estable al momento de la instalación. Se utilizó el siguiente comando para descargar el binario de Docker Compose y guardararlo en la ruta /usr/local/bin/docker-compose:

C.3. Instalación y configuración del servidor EC2

75

C.3.6. Configuración del sistema como servicio

Para garantizar que la aplicación se inicie automáticamente con el sistema, se creó un servicio a través de `systemd`. En primer lugar, se generó el archivo de configuración del servicio con el siguiente comando:

```
sudo nano /etc/systemd/system/envirosense-iot.service
```

Dentro del archivo, se definió el siguiente contenido:

```
[Unit]
Description=EnviroSenseIoT Docker Compose
After=docker.service
Requires=docker.service

[Service]
Type=oneshot
RemainAfterExit=yes
WorkingDirectory=/home/ubuntu/EnviroSenseIoT
ExecStart=/usr/bin/docker-compose up --build -d
ExecStop=/usr/bin/docker-compose down
TimeoutStartSec=0

[Install]
WantedBy=multi-user.target
```

Este servicio está diseñado para levantar y detener los contenedores Docker definidos en el archivo `docker-compose.yml`, ubicado en el directorio del sistema.

Luego se recargó la configuración de `systemd` con el siguiente comando:

```
sudo systemctl daemon-reload
```

Luego, se habilitó el servicio para que se ejecute automáticamente al iniciar el sistema:

```
sudo systemctl enable envirosense-iot.service
```

También se realizó una prueba de ejecución manual del servicio para validar su funcionamiento:

```
sudo systemctl start envirosense-iot.service
```

Para comprobar que la aplicación se ejecutaba correctamente, se consultó el estado del servicio y se verificó la actividad de los contenedores:

```
sudo systemctl status envirosense-iot.service
docker ps
```

Finalmente, se realizó un reinicio del sistema para asegurarse de que el servicio se active correctamente de forma automática:

```
sudo reboot now
```

Después del reinicio, se validó que los contenedores estuvieran en ejecución y que el servicio estuviera activo:

```
docker ps
sudo systemctl status envirosense-iot.service
```

C.3. Instalación y configuración del servidor EC2

75

```
sudo curl -SL
https://github.com/docker/compose/releases/download/
v2.17.2/docker-compose-linux-x86_64
> /usr/local/bin/docker-compose
```

Luego, se asignaron permisos de ejecución al binario de Docker Compose para que pueda ser ejecutado con el siguiente comando:

```
sudo chmod +x /usr/local/bin/docker-compose
```

A continuación, se verificó que la instalación fuera exitosa mediante:

```
docker-compose --version
```

Finalmente, se creó un enlace simbólico para facilitar su ejecución desde cualquier ruta del sistema:

```
sudo ln -s /usr/local/bin/docker-compose
/usr/bin/docker-compose
```

C.3.4. Instalación del cliente DuckDNS

Para comenzar con la instalación del cliente DuckDNS en la instancia EC2, se instalaron los paquetes necesarios mediante los siguientes comandos:

```
sudo apt-get update && sudo apt-get install cloud-utils -y
sudo apt install amazon-ec2-utils
```

A continuación, se siguieron las instrucciones proporcionadas por el sitio oficial de DuckDNS, específicamente en la sección para instancias EC2.

Luego, se creó un archivo llamado `duck.sh` en el directorio raíz del servidor. Este script se encarga de verificar si la dirección IP pública ha cambiado y, en caso afirmativo, actualizar el registro correspondiente en DuckDNS. El contenido del archivo es el siguiente:

```
#!/bin/bash
current_ip=""
while true; do
latest_ip=$(curl -S ec2-metadata --public-ipv4 | cut -d " " -f 2)
l = z "$latest_ip" ] && latest_ip=$(curl -S ifconfig.me)
echo "Current IP: $current_ip | Latest IP: $latest_ip"
if [ "$current_ip" == "$latest_ip" ]; then
echo "IP no ha cambiado."
else
echo "IP ha cambiado. Se actualiza DuckDNS..."
current_ip="$latest_ip"
response=$(curl -s -k "https://www.duckdns.org/update?
domains=envirosense&token=TOKEN&ip=$latest_ip")
echo "DuckDNS response: $response" >> ~/duckdns/duck.log
fi
sleep 5m
done
```

Este script se ejecuta de forma continua, verifica cada cinco minutos si se produjo un cambio en la IP pública de la instancia.

La figura C.3 muestra el acceso a la aplicación web EnviroSenseIoT a través del navegador web, con el nombre de dominio `envirosense.duckdns.org` asociado a la IP pública de la instancia EC2.



FIGURA C.3. Acceso a la aplicación web EnviroSenseIoT.

Finalmente, se comprobó el correcto funcionamiento de la aplicación y la comunicación mediante WebSocket, lo que permitió visualizar los datos en tiempo real. La figura C.4 ilustra cómo se presentan estos datos en la interfaz web desplegada en la instancia EC2.

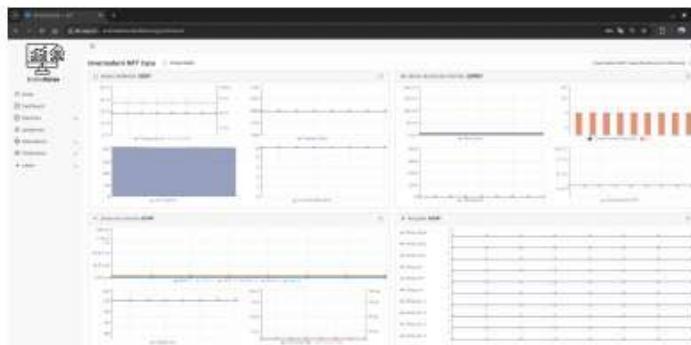


FIGURA C.4. Visualización de datos en tiempo real.

C.3.5. Descarga y configuración del repositorio EnviroSense

Para poner en marcha el sistema, primero se clonó el repositorio del código fuente desde GitHub y se accedió al directorio correspondiente con los siguientes comandos:

```
git clone
https://github.com/martinlacheski/EnviroSenseIoT.git
cd EnviroSenseIoT
```

A continuación, se ingresó al directorio EnviroSenseIoT y se creó el archivo `.env`, donde se definieron las variables de entorno necesarias para el correcto funcionamiento del sistema. Para ello, se utilizó el siguiente comando:

```
nano .env
```

En este archivo se incluyó la configuración vinculada a la conexión con la base de datos MongoDB, claves JWT para autenticación, y parámetros específicos tanto del backend como del frontend.

El contenido del archivo `.env` tiene los siguientes parámetros:

```
# Conexión MongoDB
MONGO_HOST=host # Dirección IP o nombre de host de MongoDB
MONGO_USER=user # Nombre de usuario para la conexión
MONGO_PASSWORD=password # Contraseña para la conexión
MONGO_DB=database # Nombre de la base de datos
```

```
# Conexión Backend
BACKEND_HOST=0.0.0.0 # Permitir conexión desde cualquier IP
BACKEND_CORS_ORIGINS=origins # Orígenes permitidos
```

```
# Configuración de JWT
# Generar una clave secreta con el comando:
# openssl rand -hex 32
BACKEND_SECRET_KEY=clave # Clave secreta generada
BACKEND_ALGORITHM="HS256" # Algoritmo de firma
BACKEND_ACCESS_TOKEN_EXPIRE_MINUTES=120 # Expiración token
```

```
# Configuración de Frontend
VITE_PWD_SIGNUP_ENABLED=true
VITE_BACKEND_API_URL=url # URL del backend
VITE_BACKEND_SOCKET_URL=url # URL del socket
```

Luego, se accedió al directorio `backend` del proyecto y se creó una carpeta llamada `certificates`, destinada a almacenar los certificados necesarios para las conexiones seguras:

```
cd backend
mkdir certificates
```

Dentro de esta carpeta, se generaron los archivos `client.crt`, `client.key` y `root.crt`, que contienen los certificados y la clave privada requeridos por AWS IoT Core para establecer comunicaciones seguras vía MQTT con TLS.

Bibliografía

- [1] Global Agricultural Productivity (GAP). *2016 Global Agricultural Productivity Report*. Inf. t c. Documento en linea. 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-Aguilar e Ireneo Lorenzo L pez-Cruz. «La eficiencia en el uso del agua en la agricultura controlada». En: *Tecnolog a y ciencias del agua* 5.2 (2014). Documento en linea, p ags. 177-183. URL: http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S2007-24222014000200012&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 Edici n. *Misiones: la hidroponia, cada vez m s presente*. Primera Edici n. 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 h bitos de Consumo». En: *Ecolog a Austral* 28.3 (2018), p ags. 572-580. URL: https://www.scielo.org.ar/scielo.php?script=sci_arttext&pid=S1667-782X201800400011&lng=es&tlang=es (visitado 20-03-2025).
- [6] Hidroponia 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, p ag. 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. ¿Qu  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. *Protocolos 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. t c. IETF, doi: 10.17487/RFC6455. URL: <https://tools.ietf.org/html/rfc6455>.

C.3. Instalaci n y configuraci n del servidor EC2

C.3.6. Configuraci n del sistema como servicio

Para garantizar que la aplicaci n se inicie automaticamente con el sistema, se cre  un servicio a trav s de `systemd`. En primer lugar, se gener  el archivo de configuraci n del servicio con el siguiente comando:

```
sudo nano /etc/systemd/system/envirosense-iot.service
```

Dentro del archivo, se defini  el siguiente contenido:

```
[Unit]
Description=EnviroSenseIoT Docker Compose
After=docker.service
Requires=docker.service

[Service]
Type=oneshot
RemainAfterExit=yes
WorkingDirectory=/home/ubuntu/EnviroSenseIoT
ExecStart=/usr/bin/docker-compose up --build -d
ExecStop=/usr/bin/docker-compose down
TimeoutStartSec=0

[Install]
WantedBy=multi-user.target
```

Este servicio est  dise ado para levantar y detener los contenedores Docker definidos en el archivo `docker-compose.yml`, ubicado en el directorio del sistema.

Luego se recarg  la configuraci n de `systemd` con el siguiente comando:

```
sudo systemctl daemon-reload
```

Luego, se habilit  el servicio para que se ejecute automaticamente al iniciar el sistema:

```
sudo systemctl enable envirosense-iot.service
```

Tambi n se realiz  una prueba de ejecuci n manual del servicio para validar su funcionamiento:

```
sudo systemctl start envirosense-iot.service
```

Para comprobar que la aplicaci n se ejecutaba correctamente, se consult  el estado del servicio y se verific  la actividad de los contenedores:

```
sudo systemctl status envirosense-iot.service
docker ps
```

Finalmente, se realiz  un reinicio del sistema para asegurarse de que el servicio se active correctamente de forma autom tica:

```
sudo reboot now
```

Despu s del reinicio, se valid  que los contenedores estuvieran en ejecuci n y que el servicio estuviera activo:

```
docker ps
sudo systemctl status envirosense-iot.service
```

- [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/_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 relé 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).

La figura C.3 muestra el acceso a la aplicación web EnviroSenseIoT a través del navegador web, con el nombre de dominio `envirosense.duckdns.org` asociado a la IP pública de la instancia EC2.



FIGURA C.3. Acceso a la aplicación web EnviroSenseIoT.

Finalmente, se comprobó el correcto funcionamiento de la aplicación y la comunicación mediante WebSocket, lo que permitió visualizar los datos en tiempo real. La figura C.4 ilustra cómo se presentan estos datos en la interfaz web desplegada en la instancia EC2.

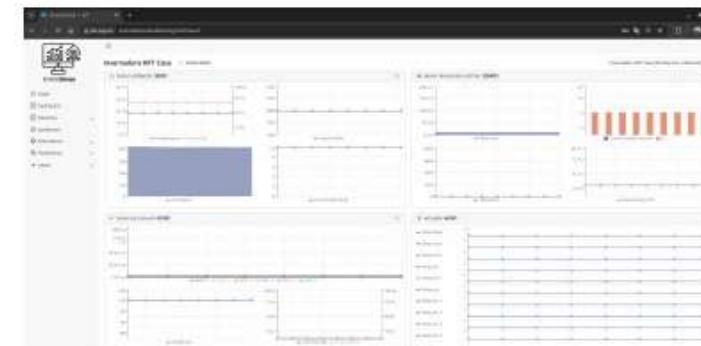


FIGURA C.4. Visualización de datos en tiempo real.

- [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] *Docker Compose Documentation*. URL: <https://docs.docker.com/compose/> (visitado 02-05-2025).
- [43] Beanie: *Asynchronous Python ODM for MongoDB*. URL: <https://beanie-odm.dev/> (visitado 01-04-2025).
- [44] Inc. MongoDB. Motor: *Asynchronous Python driver for MongoDB*. URL: <https://motor.readthedocs.io/en/stable/index.html> (visitado 01-04-2025).
- [45] 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).
- [46] FastAPI WebSockets Reference. URL: <https://fastapi.tiangolo.com/reference/websockets/> (visitado 02-04-2025).
- [47] React-Bootstrap: *Bootstrap components built with React*. URL: <https://react-bootstrap.netlify.app/> (visitado 27-04-2025).
- [48] Bootstrap Icons: *Free, high quality SVG icon library*. URL: <https://icons.getbootstrap.com/> (visitado 28-04-2025).
- [49] Axios: *Promise based HTTP client for the browser and node.js*. URL: <https://axios-http.com/> (visitado 28-04-2025).
- [50] Redux Toolkit: *The official, opinionated, batteries-included toolset for efficient Redux development*. URL: <https://redux-toolkit.js.org/> (visitado 28-04-2025).
- [51] React Router: *Declarative Routing for React.js*. URL: <https://reactrouter.com/> (visitado 28-04-2025).
- [52] Recharts: *A composable charting library built on React components*. URL: <https://recharts.org/en-US> (visitado 28-04-2025).
- [53] React Data Table Component: *Interactive and accessible data tables for React*. Versión 7.6.2 utilizada en el proyecto. URL: <https://reactdatatable.com/> (visitado 28-04-2025).
- [54] Igor Ferreira. *micropython-wifi_manager*: WiFi connection handler for MicroPython. URL: https://github.com/ferreira-igor/micropython-wifi_manager (visitado 28-04-2025).
- [55] MicroPython Community. *micropython-lib*: Standard library modules for MicroPython. URL: <https://github.com/micropython/micropython-lib> (visitado 28-04-2025).
- [56] DuckDNS: *Free Dynamic DNS hosted on AWS*. URL: <https://www.duckdns.org/> (visitado 02-05-2025).
- [57] NGINX: *High Performance Load Balancer, Web Server, & Reverse Proxy*. URL: <https://nginx.org/> (visitado 02-05-2025).
- [58] Martin Lacheski. *EnviroSenseloT*. URL: <https://github.com/martinlacheski/EnviroSenseloT> (visitado 03-05-2025).

Bibliografía

- [1] Global Agricultural Productivity (GAP). *2016 Global Agricultural Productivity Report*. Inf. téc. URL: [Enlace](#). Global Agricultural Productivity, 2016. (Visitado 20-03-2025).
- [2] Raquel Salazar-Moreno, Abraham Rojano-Aguilar e Irineo Lorenzo López-Cruz. «La eficiencia en el uso del agua en la agricultura controlada». En: *Tecnología y ciencias del agua* 5.2 (2014). URL: [Enlace](#), págs. 177-183. (Visitado 20-03-2025).
- [3] Misiones Online. *Horticultura en Misiones*. URL: [Enlace](#). (Visitado 20-03-2025).
- [4] Primera Edición. *Misiones: la hidroponía, cada vez más presente*. URL: [Enlace](#). Primera Edición. (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). URL: [Enlace](#), págs. 572-580. (Visitado 20-03-2025).
- [6] Hidroponia FIL. URL: [Enlace](#). (Visitado 20-03-2025).
- [7] Hidrosense. URL: [Enlace](#). (Visitado 20-03-2025).
- [8] iPonia. URL: [Enlace](#). (Visitado 20-03-2025).
- [9] Growcast. URL: [Enlace](#). (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). URL: [Enlace](#), pág. 01009. ISSN: 2271-2097.
- [11] OASIS Open. *Foundational IoT Messaging Protocol MQTT Becomes International OASIS Standard*. URL: [Enlace](#). OASIS Open. 2014. (Visitado 25-03-2025).
- [12] Amazon Web Services. *¿Qué es MQTT?* URL: [Enlace](#). (Visitado 25-03-2025).
- [13] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Internet Requests for Comments, RFC. URL: [Enlace](#).
- [14] IBM Corporation. *Protocolos TCP/IP*. URL: [Enlace](#). International Business Machines (IBM). (Visitado 25-03-2025).
- [15] I. Fette y A. Melnikov. *The WebSocket Protocol*. Inf. téc. URL: [Enlace](#). IETF, DOI: 10.17487/RFC6455.
- [16] Amazon Web Services. *Transport Security in AWS IoT Core*. URL: [Enlace](#). (Visitado 25-03-2025).
- [17] Espressif Systems (Shanghai) Co., Ltd. *ESP32-WROOM-32 Datasheet*. URL: [Enlace](#). (Visitado 26-03-2025).
- [18] Bosch Sensortec. *BME280 - Combined humidity, pressure and temperature sensor*. URL: [Enlace](#). (Visitado 26-03-2025).
- [19] ROHM Semiconductor. *BH1750 - Ambient Light Sensor (ALS)* - Datasheet. URL: [Enlace](#). (Visitado 26-03-2025).
- [20] Zhengzhou Winsen Electronics Technology Co., Ltd. *MH-Z19C NDIR CO₂ Sensor - Terminal Type Manual*. URL: [Enlace](#). (Visitado 26-03-2025).
- [21] DFRobot. *Technical Data of the PH-4502C Liquid pH Sensor*. URL: [Enlace](#). (Visitado 26-03-2025).

- [59] Mercado Libre. *Placa Base Mother Screw Shield NodeMCU ESP32 WROOM 38 Pines*. URL: https://articulo.mercadolibre.com.ar/MLA-1422223210-placa-base-mother-screw-shield-nodemcu-esp32-wroom-38-pines-_JM (visitado 04-05-2025).
- [60] Swagger. *Swagger: API Documentation & Design Tools for Teams*. SmartBear Software. URL: <https://swagger.io/> (visitado 04-05-2025).
- [61] Inc. MongoDB. *MongoDB Compass: GUI for MongoDB*. URL: <https://www.mongodb.com/products/tools/compass> (visitado 04-05-2025).

- [22] METTLER TOLEDO. *Sensores de Conductividad*. URL: [Enlace](#). METTLER TOLEDO International Inc. (Visitado 26-03-2025).
- [23] EC Buying. *Technical data of the EC sensor*. URL: [Enlace](#). (Visitado 26-03-2025).
- [24] Hach Company. *Sólidos totales y disueltos*. URL: [Enlace](#). (Visitado 26-03-2025).
- [25] DFRobot. *Technical data TDS sensor*. URL: [Enlace](#). (Visitado 26-03-2025).
- [26] Dallas Semiconductor. *Technical data Sensor DS18B20*. URL: [Enlace](#). (Visitado 26-03-2025).
- [27] Sparkfun Electronics. *Technical data of the HC-SR04 ultrasonic sensor*. URL: [Enlace](#). (Visitado 26-03-2025).
- [28] Unit Electronics. *Technical data of the PZEM-004T electrical energy sensor*. URL: [Enlace](#). (Visitado 26-03-2025).
- [29] Songle Relay. *Technical data of the relay*. URL: [Enlace](#). (Visitado 26-03-2025).
- [30] Damien P. George et al. *MicroPython*. URL: [Enlace](#). (Visitado 26-03-2025).
- [31] CTA Electronics. *MicroPython - Recursos y Guías*. URL: [Enlace](#). (Visitado 26-03-2025).
- [32] Tiangolo. *FastAPI*. URL: [Enlace](#). (Visitado 26-03-2025).
- [33] MongoDB, Inc. *MongoDB Documentation*. URL: [Enlace](#). (Visitado 26-03-2025).
- [34] Meta (formerly Facebook) and contributors. *React: Biblioteca de JavaScript para interfaces de usuario*. URL: [Enlace](#). (Visitado 26-03-2025).
- [35] Docker, Inc. *Docker: Desarrollo acelerado de aplicaciones en contenedores*. URL: [Enlace](#). (Visitado 26-03-2025).
- [36] Amazon.com, Inc. *AWS IoT Core*. URL: [Enlace](#). (Visitado 26-03-2025).
- [37] Amazon.com, Inc. *EC2, Nube de cómputo elástica de Amazon*. URL: [Enlace](#). (Visitado 26-03-2025).
- [38] Microsoft Corp. *Visual Studio Code*. URL: [Enlace](#). (Visitado 26-03-2025).
- [39] Pycom. *Pymakr: Plugin for syncing scripts to Pycom devices*. URL: [Enlace](#). (Visitado 26-03-2025).
- [40] Inc. MongoDB. *MongoDB Compass: GUI for MongoDB*. URL: [Enlace](#). (Visitado 04-05-2025).
- [41] Postman, Inc. *Postman API Platform*. URL: [Enlace](#). (Visitado 26-03-2025).
- [42] Git. *Sistema de control de versiones Git*. URL: [Enlace](#). (Visitado 26-03-2025).
- [43] GitHub, Inc. *GitHub*. URL: [Enlace](#). (Visitado 26-03-2025).
- [44] Docker Compose Documentation. URL: [Enlace](#). (Visitado 02-05-2025).
- [45] Beanie: *Asynchronous ODM for Python and MongoDB*. URL: [Enlace](#). (Visitado 01-04-2025).
- [46] Inc. MongoDB. *Motor: Asynchronous Python driver for MongoDB*. URL: [Enlace](#). (Visitado 01-04-2025).
- [47] Amazon Web Services. *AWS IoT SDKs - Guía del desarrollador*. URL: [Enlace](#). (Visitado 02-04-2025).
- [48] FastAPI WebSockets Reference. URL: [Enlace](#). (Visitado 02-04-2025).
- [49] React-Bootstrap. *Bootstrap components built with React*. URL: [Enlace](#). (Visitado 27-04-2025).
- [50] Bootstrap Icons. *Free, high quality SVG icon library*. URL: [Enlace](#). (Visitado 28-04-2025).
- [51] Axios: *Promise based HTTP client for the browser and node.js*. URL: [Enlace](#). (Visitado 28-04-2025).
- [52] Redux Toolkit: *The official, opinionated, batteries-included toolset for efficient Redux development*. URL: [Enlace](#). (Visitado 28-04-2025).
- [53] React Router: *Declarative Routing for React*. URL: [Enlace](#). (Visitado 28-04-2025).
- [54] Recharts: *A composable charting library built on React components*. URL: [Enlace](#). (Visitado 28-04-2025).