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.

I

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.

Índice de figuras

IX

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 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 EnviroSenseIoT 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

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.

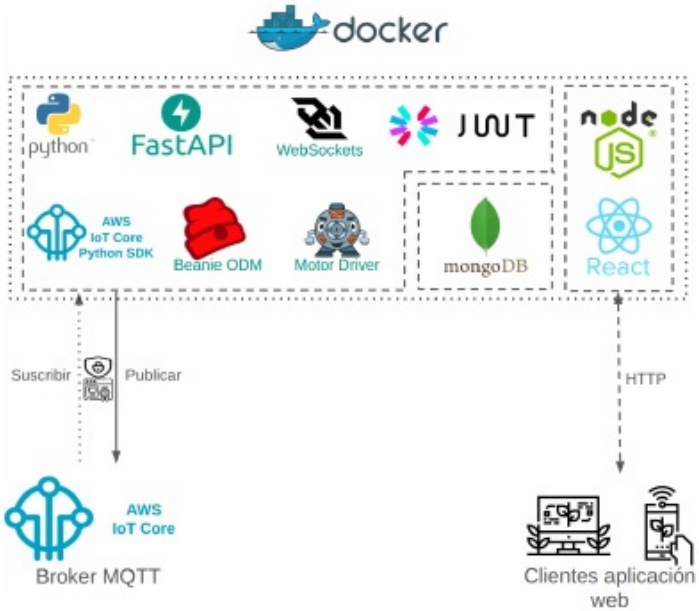

FIGURA 3.3. Arquitectura del servidor del sistema IoT.

18 Capítulo 3. Diseño e implementación

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.

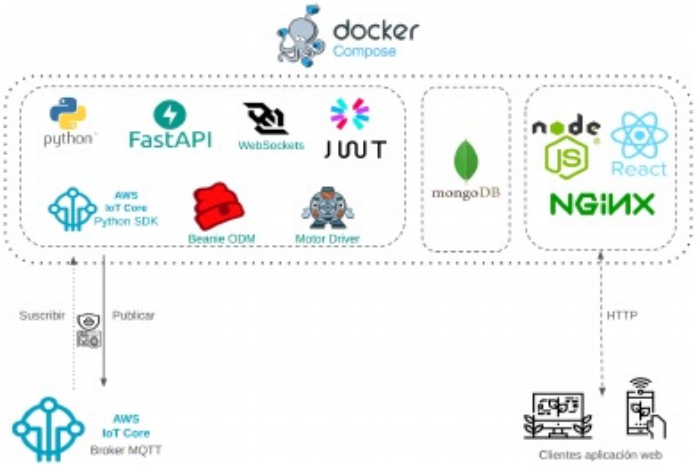

FIGURA 3.3. Arquitectura del servidor del sistema IoT.

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

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	/mqtt/test	Test conexión cliente MQTT.
POST	/mqtt/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

[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	/mqtt/test	Test conexión cliente MQTT.
POST	/mqtt/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.

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 token implementado en el sistema.

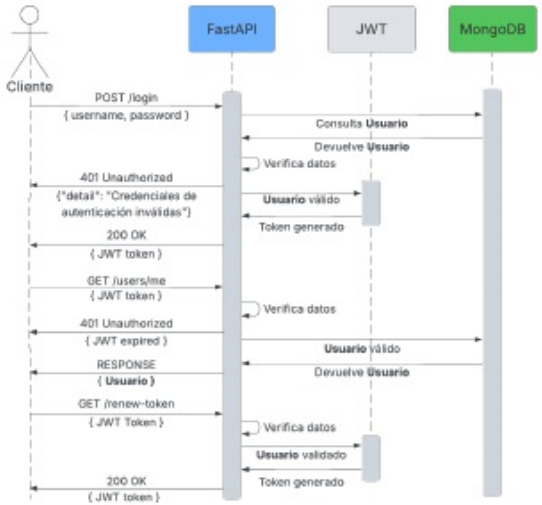

FIGURA 3.4. Esquema de autenticación y autorización.

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

3.4.2. 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 token implementado en el sistema.

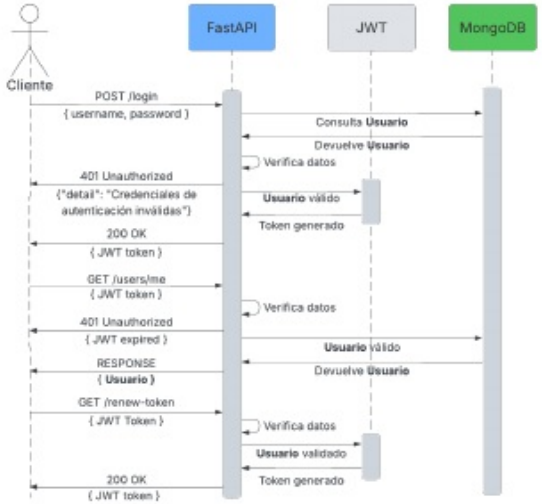

FIGURA 3.4. Esquema de autenticación y autorización.

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

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`.

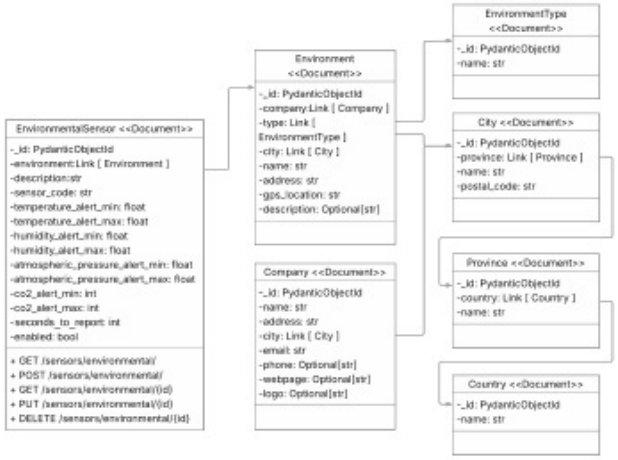

FIGURA 3.5. Ejemplo de diagrama de clase de la colección `EnvironmentalSensor`.

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.

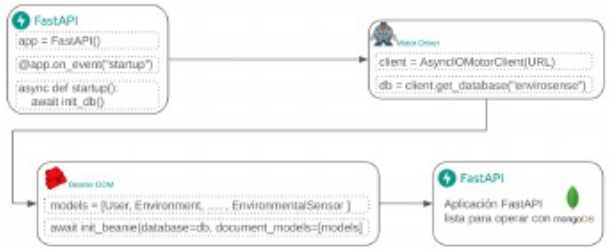

FIGURA 3.6. Pasos para la conexión de FastAPI y MongoDB.

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`.

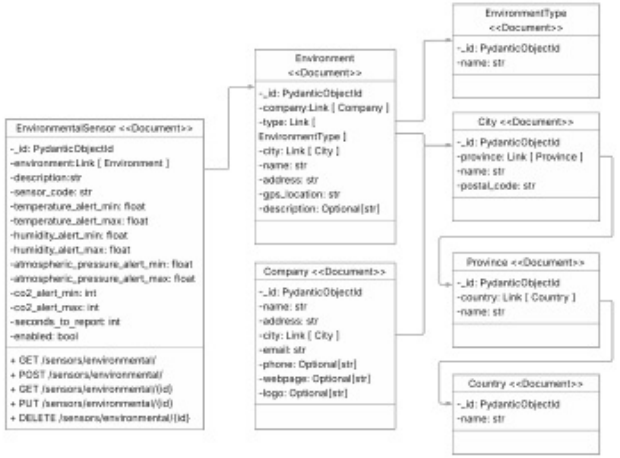

FIGURA 3.5. Diagrama de clase de la colección `EnvironmentalSensor`.

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.

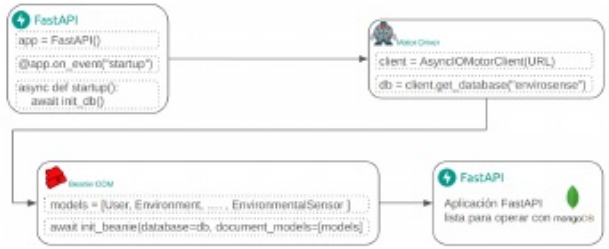

FIGURA 3.6. Pasos para la conexión de FastAPI y MongoDB.