Extensión del módulo de gestión de datos de la plataforma Smart Campus UIS para el soporte de diversos protocolos y almacenamiento de datos

Juan Camilo Lozada Garavito y Jerson Julian Cañon Castillo

Trabajo de Grado para Optar al Título de Ingenieros de Sistemas

Director

Gabriel Rodrigo Pedraza Ferreira

PhD. Ciencias de la Computación

Codirector

Daniel Felipe Jaimes Blanco

Ingeniero de Sistemas

Universidad Industrial de Santander

Facultad de Ingenierías Físico Mecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2025

Dedicatoria

Dedico este proyecto a mi familia, por ser el motor de mi motivación, por el apoyo constante que me brindaron para alcanzar este sueño, por su amor sincero y la fortaleza que representan para mí. Espero seguir creciendo profesionalmente y poder contribuir con mi conocimiento al bienestar de los demás.

Jerson Julian Cañon Castillo

Agradecimientos

Le agradezco a Dios, primeramente, por permitirme llegar hasta aquí, por llenarme de fuerzas cada día y darme la motivación para alcanzar cada uno de mis sueños. Agradezco a mis hermanos, quienes han sido mi principal inspiración y un gran apoyo en cada paso de este camino.

A mis padres, gracias por estar siempre aconsejándome y ayudándome a cumplir los planes que me proponía, con amor y sabiduría.

Agradezco a mi compañero Juan, una persona que aportó significativamente en mi vida universitaria, con quien pude crecer académicamente gracias a su conocimiento, dedicación y compromiso en cada reto que enfrentamos juntos a lo largo de la carrera.

Le agradezco a nuestro director Gabriel por compartir su conocimiento, por ser una guía fundamental en el proceso de aprendizaje y ayudarnos a alcanzar los objetivos de este proyecto.

También agradezco a nuestro codirector Daniel por su asesoramiento y su disposición para ayudarnos en la realización de este proyecto.

Finalmente, doy mis más sinceros agradecimientos a cada una de las personas que hicieron parte de mi vida universitaria y que, con su presencia, me permitieron crecer tanto intelectual como personalmente.

Jerson Julian Cañon Castillo

Dedicatoria

Dedico este proyecto a mi mama, mi motor de vida, por el apoyo incondicional y cariño diario que ayudaron a formar al hombre que soy ahora. El ejemplo vivo de que las metas se logran con sacrificio, esfuerzo, actitud y mucho amor. Por qué gran parte de este logro es por todo su esfuerzo diario que me permitió crecer como persona y profesional. Te amo madre.

Juan Camilo Lozada Garavito

Agradecimientos

De nuevo quiero agradecer a la persona que me dio la vida y me permitió estar aquí terminando una meta más en mi vida, mi madre. La piedra angular de mi vida y la mayor motivación para culminar esta carrera.

Quiero agradecer a mis abuelos por estar siempre presentes, por todos esos momentos que hemos pasado juntos y en los que he aprendido un montón de valores que me han ayudado culminar este proyecto.

Agradezco también a mis tías, que siempre han sido incondicionales y han estado conmigo para lo que necesite, son ejemplo de personas guerreras, fuertes y trabajadoras. Y han sido muy importantes a lo largo de toda mi carrera.

Agradezco a Sofi, mi hermanita, que en su corta edad me recuerda lo bonito que es compartir y amar a todos por igual. Ha sido una gran motivación para seguir adelante.

Agradezco a Mari, lo más bonito que me dio la universidad, su amor, compañía y fidelidad son únicas. Gracias a su apoyo diario todo lo logrado hasta el día de hoy ha sido un poco más fácil.

Y por último quiero agradecer a mi compañero Julián por estar siempre apoyándome y aconsejándome en todos los retos enfrentados durante la carrera. Al profe Gabriel por la confianza depositada en el desarrollo del proyecto, y transmitirnos siempre con buena actitud sus conocimientos que nos permitieron crecer como profesionales. Y nuestro codirector Daniel por sus consejos y guía que permitieron tener un mejor desarrollo de este proyecto.

Juan Camilo Lozada Garavito

Tabla de Contenido

Pág.

Lista de Tablas

Lista de Figuras

Lista de Apéndices

Glosario

AMQP (Advanced Message Queuing Protocol): Protocolo de mensajería avanzado basado en colas. Facilita la comunicación confiable, ordenada y flexible entre sistemas mediante un modelo de publish/suscribe. Es la base de funcionamiento de brokers como RabbitMQ.

API (Application Programming Interface): Conjunto de herramientas y protocolos que permiten la interacción entre componentes de software. Las APIs facilitan la integración de servicios, como las interacciones entre el backend y otros módulos de la plataforma.

Arquitectura de cebolla (Onion Architecture): Patrón de diseño modular en el que el núcleo (las entidades del dominio y reglas de negocio) se encuentra rodeado por capas que representan funcionalidades de aplicación, interfaces e infraestructura. La dependencia fluye desde el exterior hacia el interior, permitiendo una alta mantenibilidad y desacoplamiento de tecnologías externas.

Arquitectura hexagonal: Enfoque de diseño que separa el núcleo de la aplicación de los detalles externos mediante el uso de puertos (interfaces) y adaptadores. Permite que los componentes del negocio se prueben de forma aislada, facilitando la adaptabilidad a diferentes dispositivos o sistemas de entrada/salida.

Arquitecturas modulares: Conjunto de arquitecturas de software que tienen como objetivo dividir un sistema en módulos independientes que puedan desarrollarse, probarse, desplegarse y mantenerse de manera autónoma. Estas arquitecturas facilitan la escalabilidad y la evolución del sistema ante cambios tecnológicos y de requerimientos.

CoAP (Constrained Application Protocol): Protocolo utilizado en IoT para establecer comunicación entre sensores y actuadores. Está orientado a dispositivos con recursos limitados y redes restringidas.

Gateway: Se conoce como un dispositivo que actúa como puente entre diferentes redes o protocolos. En el contexto de este proyecto, el gateway es utilizado para recibir los mensajes de los sensores y dirigirlos al módulo de gestión de datos.

HTTP (Hypertext Transfer Protocol): Protocolo fundamental en el web utilizado para establecer conexiones entre clientes y servidores, permitiendo el intercambio de información mediante solicitudes y respuestas.

MQTT (Message Queuing Telemetry Transport): Protocolo de mensajería ligero y eficiente, utilizado en entornos IoT con recursos limitados. En este proyecto se utilizó MQTT a través del bróker mensajería EMQX para asegurar una comunicación rápida y confiable entre dispositivos y backend.

RDBMS (Relational Database Management System): Son **s**istemas que permiten la gestión de bases de datos relacionales. La base de datos relacionales utiliza esquemas estructurados de tablas con diferentes campos de información y relaciones entre las tablas.

Reencolado: Técnica mediante la cual se coloca un mensaje obtenido previamente en una nueva cola para su procesamiento posterior.

TSDB (Time Series Database): Es un tipo de base de datos optimizada para almacenar y consultar datos que varían en atreves del tiempo. Sus índices están basados en marcas de tiempo, lo que permite consultas eficientes sobre grandes volúmenes de datos en diferentes rangos de tiempo. InfluxDB es un ejemplo representativo.

Resumen

Título: Extensión del módulo de gestión de datos de la plataforma Smart Campus UIS para el soporte de diversos protocolos y almacenamiento de datos *

Autor: Juan Camilo Lozada Garavito y Jerson Julian Cañon Castillo **

Palabras clave: IoT, Smart Camus UIS, protocolos, InfuxDB, MongoDB, MinIO, arquitectura modular, MQTT, AMQP, RTSP, HTTP.

Descripción: El presente proyecto propone la extensión y mejora del módulo de gestión de datos de la plataforma *Smart Campus UIS*, con el objetivo de la recepción, almacenamiento y redistribución de datos provenientes de dispositivos IoT que incorporan un conjunto de sensores. La iniciativa surge ante las limitaciones que tenía el módulo existente, basado únicamente en el protocolo de comunicación MQTT y una base de datos documental, lo dificultaba tanto el manejo de datos de gran tamaño (videos e imágenes) como la organización y análisis eficiente de datos temporales. Para abordar esta problemática, se definieron los requisitos funcionales y no funcionales, adoptando un enfoque basado en una arquitectura modular, —específicamente la arquitectura Onion—, que separa la lógica de negocio de las dependencias tecnológicas.

La solución diseñada e implementada permite el soporte de múltiples protocolos de comunicación (AMQP, MQTT, RTSP Y HTTP) y emplea diversas bases de datos especializadas: InfluxDB para datos de series temporales, MongoDB para datos estructurados y semiestructurados, y MinIO para el almacenamiento de objetos. Además, el módulo fue diseñado con un enfoque de flexibilidad y escalabilidad, permitiendo la integración futura de nuevos protocolos, alternativas de almacenamiento y mecanismos de redistribución de datos según las necesidades del entorno. El proyecto abarca fases de análisis de requerimientos, diseño de la arquitectura, implementación, pruebas y validación funcional, reflejando mejoras en la eficiencia de almacenamiento, la transmisión y consulta de datos.

En definitiva, este trabajo constituye un avance significativo en la evolución y escalabilidad de la plataforma *Smart Campus UIS*, ofreciendo una solución robusta y flexible para futuros entornos IoT.

---

* Trabajo de grado

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Gabriel Rodrigo Pedraza Ferreira. Codirector: Daniel Felipe Jaimes Blanco

Abstract

Title: Extension of the Data Management Module of the Smart Campus UIS Platform for the Support of Various Protocols and Data Storage*

Author(s): Juan Camilo Lozada Garavito y Jerson Julian Cañon Castillo **

Key Words: IoT, Smart Campus UIS, protocols, InfluxDB, MongoDB, MinIO, modular architecture, MQTT, AMQP, RTSP, HTTP.

Description: This project proposes the extension and enhancement of the data management module of the *Smart Campus UIS* platform, with the objective of enabling the reception, storage, and redistribution of data from IoT devices equipped with a set of sensors. The initiative arises from the limitations of the existing module, which relied solely on the MQTT communication protocol and a document-oriented database, making it difficult to handle large data volumes (such as videos and images) as well as to efficiently organize and analyze time-series data. To address these challenges, both functional and non-functional requirements were defined, adopting a modular architecture approach—specifically, the Onion Architecture—which separates business logic from technological dependencies.

The designed and implemented solution supports multiple communication protocols (AMQP, MQTT, RTSP, and HTTP) and leverages a variety of specialized databases: InfluxDB for time-series data, MongoDB for structured and semi-structured data, and MinIO for object storage. Furthermore, the module was designed with a focus on flexibility and scalability, enabling future integration of new protocols, storage alternatives, and data redistribution mechanisms according to evolving needs. The project encompasses requirement analysis, architectural design, implementation, testing, and functional validation, demonstrating improvements in data storage, transmission, and query efficiency.

Ultimately, this work represents a significant advancement in the evolution and scalability of the Smart Campus UIS platform, offering a robust and flexible solution for future IoT environments.

---

* Bachelor's Thesis

** Faculty of Physicomechanical Engineering. School of Systems and Computer Engineering. Supervisor: Gabriel Rodrigo Pedraza Ferreira. Co-Supervisor: Daniel Felipe Jaimes Blanco

Introducción

La evolución tecnológica ha impulsado el crecimiento de las plataformas de Internet de las Cosas (IoT), que permiten la conexión y comunicación de dispositivos para la recolección y análisis de datos en tiempo real. Las universidades, como centros de innovación y aprendizaje, no son ajenas a esta tendencia. En este contexto, la Universidad Industrial de Santander (UIS) ha emprendido el ambicioso proyecto Smart Campus UIS (Jiménez, Cárcamo y Pedraza, 2020), una iniciativa que busca transformar el campus universitario en un ecosistema inteligente y conectado.

El proyecto *Smart Campus UIS* se basa en la implementación de una infraestructura de Internet de las Cosas (IoT) diseñada para capturar, procesar y analizar datos en tiempo real provenientes de diversos sensores distribuidos por el campus. Esta infraestructura tiene como objetivo mejorar diversos aspectos de la vida académica y optimizar el funcionamiento general del campus universitario.

Sin embargo, a medida que el proyecto evoluciona, se identificó limitaciones en el módulo que se tenía de gestión de datos. Estas restricciones estaban relacionadas principalmente con la capacidad de manejar diversos tipos de datos no estructurados y temporales, la compatibilidad con múltiples protocolos de comunicación (AMQP, RSTP Y HTTP), y la flexibilidad en el almacenamiento —utilizando bases de datos de series temporales como InfluxDB, documentales como MongoDB y almacenamiento de objetos como MinIO— y en la consulta de la información recopilada. Además, se evidenció la

necesidad de mejorar la redistribución eficiente de la información, dependiendo del tipo de dato: temporal, documental u orientado a objetos.

El presente trabajo de grado se enfocó en abordar estas limitaciones mediante el rediseño y extensión del módulo de gestión de datos de la plataforma Smart Campus UIS. Para cumplir con este objetivo, se desarrolló una solución que permita el soporte de protocolos de comunicación como MQTT y AMQP para datos en texto plano provenientes de los gateways, y RTSP para la captura de video en tiempo real. Además, se implementaron diferentes modelos de almacenamiento de datos: MinIO, una base de datos compatible con S3, para guardar datos no estructurados; InfluxDB para datos en series temporales; y MongoDB para almacenar el historial de los mensajes. Con esto cambios, se mejoró la capacidad de la plataforma para manejar diversos tipos de información, como videos e imágenes, se optimizó el rendimiento general del sistema y se facilitó su escalabilidad futura mediante una nueva arquitectura.

Este proyecto represento un paso significativo en la evolución de la plataforma Smart Campus UIS, añadiendo una mayor capacidad de extensión a futuro de manera sostenible y mantenible para la integración de nuevos protocolos, la extensión de la API sacar provecho de los datos almacenados y lograr un mejor análisis de estos. Además, implementación en entornos reales contribuyendo a la creación de un entorno universitario más inteligente, eficiente y adaptable a las necesidades cambiantes de la comunidad académica.

1. Planteamiento y justificación del problema

La plataforma *Smart Campus UIS* contaba con un microservicio de datos que capturaba información a través de un gateway utilizando el protocolo MQTT. Aunque MQTT es eficiente para enviar mensajes cortos y frecuentes, se exploraron otros protocolos más adecuados para la transmisión de formatos de datos más pesados, como videos e imágenes. Esto permite que la plataforma continúe utilizando MQTT para su propósito original, al tiempo que se implementaron soluciones adicionales para soportar otros tipos de datos.

Por otro lado, la plataforma almacenaba datos en una base de datos documental, gestionada con MongoDB. Se busco una alternativa que permita consultar los datos con flexibilidad en rangos de tiempo, así como una base de datos adecuada para almacenar videos e imágenes. Esto garantiza la flexibilidad necesaria para que cada tipo de dato se almacene en su base de datos correspondiente, lo que optimiza el almacenamiento y mejora el rendimiento. Al tener consultas eficientes y escalabilidad independiente, se facilita el manejo de datos heterogéneos y se permite un control más granular sobre la seguridad y los permisos. Además, el uso de herramientas especializadas para cada tipo de dato agiliza el desarrollo, resultando en un sistema más robusto y eficiente.

2. Objetivos

2.1 Objetivo general

Rediseñar el módulo de gestión de datos de la plataforma *Smart Campus UIS* para soportar múltiples protocolos de comunicación y diferentes modelos de almacenamiento de datos, además de permitir su extensibilidad.

2.2 Objetivos específicos

- Determinar los requerimientos funcionales y no funcionales para la extensión del módulo de gestión de datos de la plataforma *Smart Campus UIS.*
- Diseñar una solución a partir de los requerimientos funcionales y no funcionales establecidos.
- Implementar la solución, abarcando el backend del módulo de gestión de datos, la base de datos temporal y el almacenamiento de objetos.
- Validar la solución implementada mediante pruebas que verifiquen su funcionalidad y calidad.

3. Marco de referencia

3.1 Marco conceptual

3.1.1 Internet de las cosas (IoT)

El Internet de las cosas (IoT) representa un cambio de paradigma en la informática, transformando los objetos cotidianos en dispositivos inteligentes capaces de detectar, procesar y comunicar datos. Se enfoca en conectar dispositivos físicos a Internet para recopilar e intercambiar datos, lo que permite el monitoreo remoto y tareas de automatización básicas. Esto abarca una amplia gama de aplicaciones, desde dispositivos domésticos inteligentes como termostatos y bombillas hasta sensores industriales y equipos de atención médica. (Quincozes et al, 2024)

Pero para lograr esta interoperabilidad los desarrolladores de IoT deben considerar la interconexión de muchos dispositivos heterogéneos, el análisis de cantidades masivas de datos, la definición de eventos y respuestas relevantes, y la definición de flujos de datos y acciones. La complejidad y heterogeneidad intrínsecas del mundo de IoT y la falta de estandarización relacionada con el proceso de desarrollo de aplicaciones de IoT han sido reconocidas como problemas relevantes en esta área. (Corradini et al, 2023)

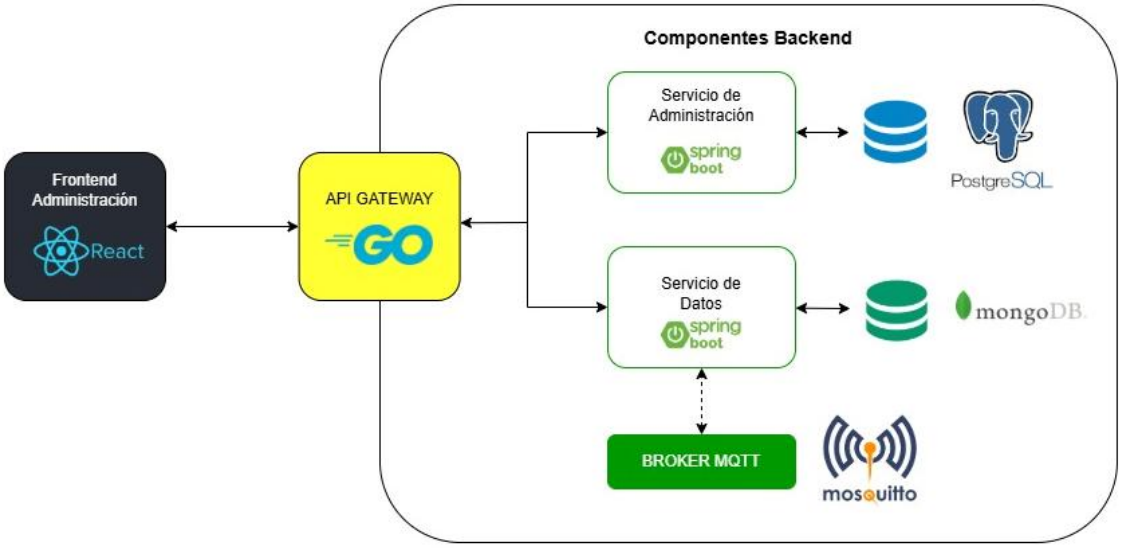
3.1.2 Smart Campus UIS

El proyecto *Smart Campus UIS* busca desarrollar una infraestructura de internet de las cosas (IoT) para manejar datos en tiempo real en el campus universitario. Sensores ubicados en diferentes lugares recogen información para mejorar varios aspectos de la vida académica y el

funcionamiento del campus.

Figura 1

Arquitectura de Smart Campus antes de la extensión



3.1.3 Protocolos y tecnologías IoT

Entre los protocolos de capa de aplicación de IoT populares se encuentran el transporte de telemetría de cola de mensajes (MQTT), el Protocolo de aplicación restringida (CoAP), el Protocolo de cola de mensajes avanzado (AMQP), el Protocolo de presencia y mensajería extensible (XMPP), el protocolo de mensajería orientada a texto simple (STOMP), el Protocolo de transferencia de estado representacional (RESTful), Protocolo de transferencia de hipertexto (HTTP), el Protocolo simple de acceso a objetos (SOAP) y WebSocket. Sin embargo, el proceso de elegir un protocolo de mensajería adecuado y eficaz representa un desafío que puede llegar a ser muy abrumador para las organizaciones, ya que depende de las características específicas y los requisitos de mensajería que necesite el sistema IoT. (Tran et la, 2024)

Algunas aplicaciones que hacen uso de los diferentes protocolos nombrados y ayudan a la creación de aplicaciones IoT son:

- Bevywise es una plataforma que despliega aplicaciones IoT industriales y comerciales que permiten recolectar, analizar y visualizar datos históricos en tiempo real. Brinda una API REST para facilitar la construcción de aplicaciones móviles para los clientes. (Tran et la, 2024)
- ThingsBoard es una plataforma de código abierto que permite la recopilación, procesamiento, visualización y gestión de dispositivos de datos. Los dispositivos adoptan protocolos de capa de aplicación IoT como MQTT, CoAP y HTTP para proporcionar conexión. La plataforma admite implementaciones tanto en la nube como locales, así como también permite la construcción de clústeres para máxima escalabilidad, tolerancia a fallas y rendimiento mediante la implementación de la arquitectura de microservicios. (Tran et la, 2024)
- IoTgo es una plataforma de código abierto que permite a cualquier persona implementar su propio servicio en la nube. Según los autores, la plataforma fue diseñada para ser abierta, gratuita, sencilla y fácil de usar. La plataforma admite interfaces móviles, web y de escritorio. Además, proporciona una API que, al integrarse en las interfaces, permite a los clientes controlar los dispositivos. La plataforma admite los protocolos HTTP y WebSocket. (Tran et la, 2024)

3.1.4 Protocolo de transmisión en tiempo real (RTSP)

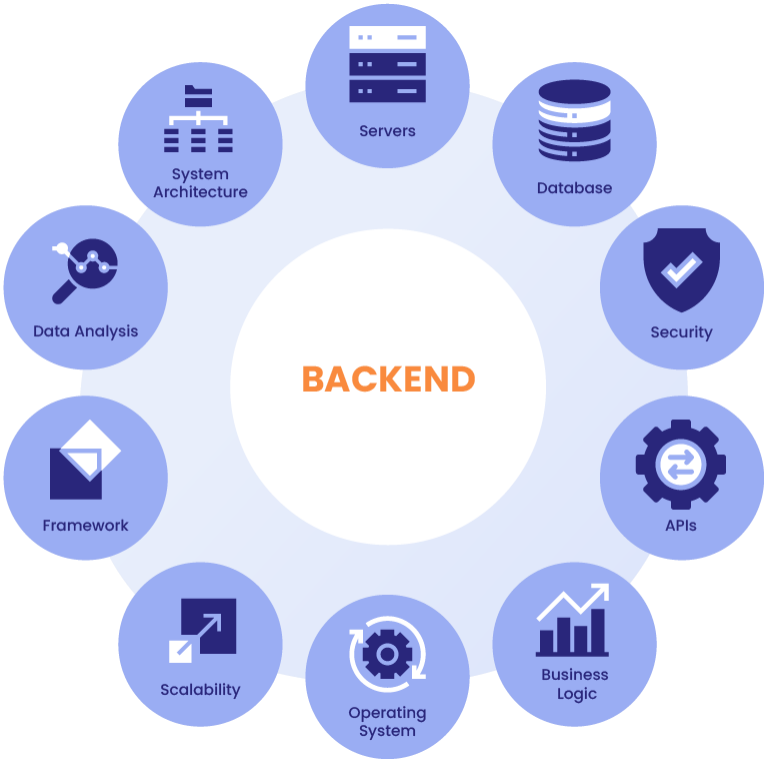
El Protocolo de Transmisión en Tiempo Real (RTSP) se utiliza para transferir datos multimedia en tiempo real, incluyendo audio y video, entre un servidor y un cliente. Es un protocolo de transmisión, lo que significa que busca facilitar escenarios en los que los datos multimedia se transfieren y renderizan simultáneamente; es decir, se muestra el video y se reproduce el audio. RTSP utiliza una conexión de Protocolo de Control de Transmisión (TCP) para controlar la sesión de transmisión multimedia, aunque también es posible utilizar UDP para este propósito. La entidad que envía la solicitud RTSP que inicia la sesión se denomina cliente, y la entidad que responde a dicha solicitud se denomina servidor. (Microsoft Learn, 2024)

3.1.5 Backend

Backend es la parte de una solución software que opera del lado del servidor. Su función es conectar las bases de datos, gestionar la autenticación de usuarios, alojar la solución en un servidor, asegurar la protección de los datos y, en general, procesar los datos para que posteriormente puedan ser consumidos. (Otar, 2023)

Figura 2

Elementos del Backend



Adaptado de (Otar, 2023)

3.1.6 Bases de datos documentales

Este tipo de bases de datos pertenece a la categoría de bases de datos NoSQL. Se caracterizan por almacenar y gestionar datos en forma de documentos en formato JSON, los cuales son agrupados mediante colecciones. Permite escalar de manera eficiente, brindando una solución para almacenar grandes volúmenes de datos semiestructurados. (Luis Soler, 2024)

Principales gestores de bases de datos documentales que se usan con mayor frecuencia:

- MongoDB: Permite almacenar grandes volúmenes de datos de forma flexible, escalable y eficiente. Cuenta con herramientas gráficas de gestión y visualización de datos que permiten explorar las bases de datos, crear consultas, analizar esquemas, entre otras funcionalidades. También ofrece herramientas de línea de comandos (Shell y CLI) para administradores y desarrolladores. Además, cuenta con un servicio en la nube que permite desplegar instancias de bases de datos. (Oscar Fernandez, 2021)

- Azure CosmosDB: es un servicio que está basado en la nube. Es multimodelo y posee distribución global, lo que facilita la escalabilidad y brinda baja latencia para aplicaciones que requieren alta disponibilidad, flexibilidad y acceso global. Permite realizar consultas con una sintaxis similar a SQL. (Microsoft, 2025)

- DynamoDB: es un servicio de base de datos proporcionado por Amazon Web Services (AWS) que ofrece alta escalabilidad para aplicaciones modernas. Cuenta con

funcionalidades como respaldos automáticos, recuperación de datos y almacenamiento en caché. Además de admitir modelos documentales, también permite el almacenamiento de datos mediante el modelo clave-valor. (Chritopher Adamson, 2023)

3.1.7 Bases de datos temporales

Las bases de datos de series temporales (TSDB) pertenecen a las bases de datos NoSQL y ofrecen la posibilidad de almacenar múltiples series temporales, de modo que las consultas para recuperar datos de una o varias series temporales para un intervalo de tiempo determinado son especialmente eficientes. (Ye, F. et al., 2020) La indexación en las TSDB se hace por medio de las marcas de tiempo de los datos, por esto su gran rendimiento en las consultas con intervalos de tiempo. (Mario San Emeterio, 2023)

Algunas TSDB populares en la actualidad son:

- InfluxDB es una TSDB, ofrece un conjunto de características para gestionar datos de series temporales, como la agregación y la agrupación. Además de eso, InfluxDB también admite consultas matemáticas para ayudar a analizar los datos. InfluxDB también ofrece un panel de control de monitoreo integrado para visualizar los datos almacenados. InfluxDB utiliza un enfoque basado en push para recopilar datos, lo que significa que la fuente de datos transmite activamente sus datos a InfluxDB. (Simanjuntak & Surantha, 2022)
- Prometheus es una solución de monitorización de código abierto que se utiliza para comprender las percepciones de los datos métricos y enviar las alertas necesarias. Cuenta

con una base de datos local de series temporales en disco que almacena los datos en un formato personalizado en el disco. (Avi, 2020)

- TimescaleDB es una base de datos relacional de código abierto que hace que SQL sea escalable para datos de series temporales. Esta base de datos está construida sobre PostgreSQL. (Avi, 2020)

3.1.8 Almacenamiento de objetos

El almacenamiento de objetos es una arquitectura de almacenamiento de datos informáticos que está diseñada para manejar grandes cantidades de datos no estructurados. Designa los datos como unidades distintas, agrupadas con metadatos y un identificador único que se puede usar para ubicar y acceder a cada unidad de datos. (Google, 2024)

Las unidades, o los objetos, pueden almacenarse de forma local, pero, pero por lo general, se almacenan en la nube, facilitando su acceso desde cualquier lugar. La mayoría de los datos no estructurados son: correos electrónicos, archivos multimedia y de audio, páginas web, datos de sensores y otros tipos de contenido digital que no se adaptan a una base de datos tradicional. (Google, 2024)

En el almacenamiento de objetos los bloques de datos de un archivo se agrupan como un objeto, junto con sus metadatos relevantes y su identificador personalizado y se colocan en un entorno plano conocido como un grupo de almacenamiento. (Google, 2024)

Cuando se quiere acceder a los datos, los sistemas de almacenamiento de objetos utilizan el identificador único y los metadatos para encontrar el objeto de interés. Se puede

acceder a ellos mediante las APIs de RESTful, HTTP Y HTTPS para consultar metadatos de objetos. (Google, 2024)

Los principales servicios de almacenamiento de objetos son:

- **Amazon S3**: ofrece una variedad de clases de almacenamiento entre las cuales puede elegir en función de los requisitos de rendimiento, acceso a los datos, resiliencia y costos de sus cargas de trabajo. Las clases de almacenamiento de S3 se crearon específicamente para brindar el menor costo posible de almacenamiento para los diferentes patrones de acceso. Las clases de almacenamiento de S3 son ideales prácticamente para cualquier caso de uso, incluidos los que cuentan con necesidades de rendimiento demandantes, lagos de datos, requisitos de residencia de datos, patrones de acceso desconocidos o cambiantes, o almacenamiento de archivos. (Amazon, 2024)

- **Google Cloud:** es un servicio de almacenamiento de objetos potente, simple y rentable construido para el escalamiento de Google. Los SDK de Firebase para Cloud Storage agregan la seguridad de Google a las operaciones de carga y descarga de archivos de las apps de Firebase, sin importar la calidad de la red. (Firebase, 2024)

  Se puede utilizar para almacenar imágenes, audio, video y otros tipos de contenido que se caractericen por ser no estructurados. El servidor permite usar la herramienta Firebase Admin SDK para administrar buckets y crear URLs de descarga, además de las APIs de Google Cloud Storage para acceder a los archivos. (Firebase, 2024)

- **Minio:** Minio es un software de almacenamiento de objetos de código abierto que es compatible con la API de Amazon S3. Está diseñado para ser escalable y de alto rendimiento, y puede desplegarse en diferentes entornos, como nubes públicas o privadas, entornos orquestados y bordes. (Kumar, 2020)

3.1.9 Arquitecturas modulares

Uno de los principales objetivos de las arquitecturas modular son dividir responsabilidades por medio de diferentes capas y estas capas con el crecimiento del sistema pueden transformarse en módulos.

Cada una de estas arquitecturas produce sistemas que son:

- **Independientes de los frameworks:** La arquitectura no depende de la existencia de una biblioteca de software con numerosas funcionalidades. Esto permite utilizar dichos frameworks como herramientas, en lugar de tener que limitar el sistema a sus limitaciones. (Robert C, 2012)
- **Comprobables:** Las reglas de negocio se pueden probar sin la interfaz de usuario, la base de datos, el servidor web ni ningún otro elemento externo. (Robert C, 2012)

- **Independientes de la interfaz de usuario:** La interfaz de usuario puede cambiar fácilmente sin modificar el resto del sistema. Por ejemplo, una interfaz web podría sustituirse por una interfaz de consola sin modificar las reglas de negocio. (Robert C, 2012)

- **Independientes de la base de datos:** Puede sustituir Oracle por Mongo. Sus reglas de negocio no están vinculadas a la base de datos. (Robert C, 2012)

- **Independientes de cualquier agente externo:** De hecho, sus reglas de negocio simplemente desconocen por completo el mundo exterior. (Robert C, 2012)

Algunas arquitecturas famosas que cumplen estos requisitos pueden ser:

- **Arquitectura modular:** El patrón de arquitectura hexagonal, también conocido como patrón de puertos y adaptadores, fue propuesto por el Dr. Alistair Cockburn en 2005. Su objetivo es crear arquitecturas de acoplamiento flexible en las que los componentes de las aplicaciones puedan probarse de forma independiente, sin depender de los almacenes de datos ni de las interfaces de usuario. se utiliza para aislar la lógica empresarial (lógica de dominio) del código de infraestructura relacionado, como el código para acceder a una base de datos o el código externo. (AWS, 2020)

- **Arquitectura de cebolla:** Su premisa principal es que controla el acoplamiento. La regla fundamental es que todo el acoplamiento se dirige hacia el centro. En el centro, tenemos el Modelo de Dominio, que representa la combinación de estado y comportamiento que modela la verdad para la organización. Alrededor del Modelo de Dominio se encuentran otras capas con más comportamiento. La primera capa alrededor del Modelo de Dominio es donde normalmente encontramos las interfaces que proporcionan comportamientos de guardado y recuperación de objetos, llamadas interfaces de repositorio. Sin embargo, el

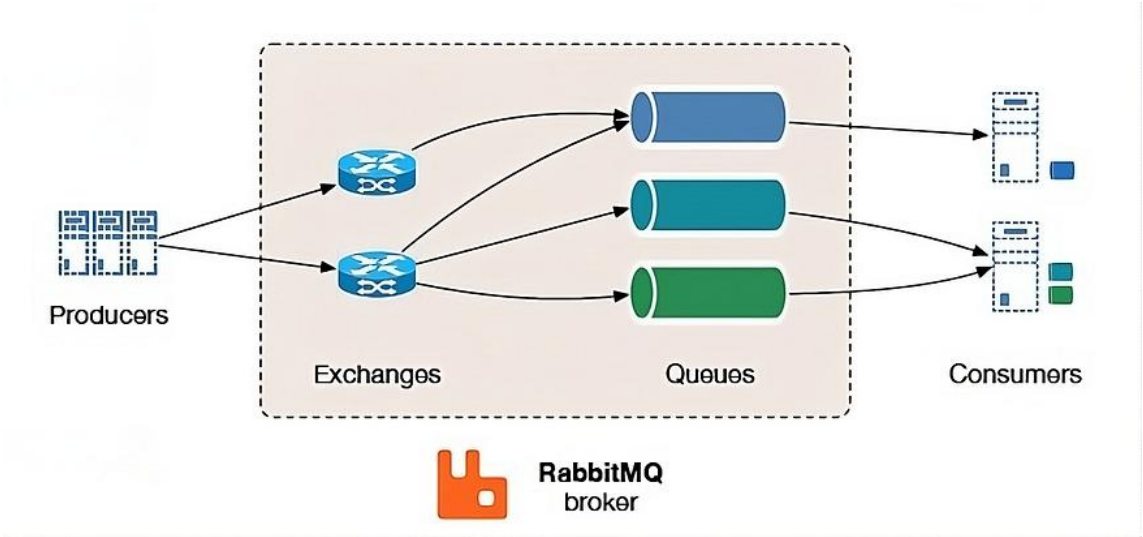
comportamiento de guardado de objetos no se encuentra en el núcleo de la aplicación. La capa exterior está reservada para elementos que cambian con frecuencia. Estos elementos deben aislarse intencionalmente del núcleo de la aplicación. (Jeffrey Palermo, 2018)

3.1.10 Brokers de mensajería

Un bróker de mensajería es un intermediario encargado de gestionar la comunicación entre aplicaciones o sistemas de software que desean intercambiar mensajes entre sí. En lugar de establecer una comunicación directa, actúa como puente entre emisores (publishers) y los receptores (subscribers) de mensajes, utilizando un modelo conocido como publish/subscribe, en el cual los mensajes se almacenan en una cola de solicitudes pendientes y se envían a los componentes suscriptos. (Diego Cortes, 2023)

Figura 3

Arquitectura del bróker RabbitMQ.



Adaptado de (Diego Cortes, 2023)

A continuación, se presentan dos de los brokers de mensajería más utilizados:

- RabbitMQ: Este bróker de mensajería está basado en el protocolo AMQP (Advanced Message Queuing Protocol). Su función principal es recibir, almacenar y distribuir mensajes entre aplicaciones de forma confiable, ordenada y flexible. (Diego Cortes, 2023)

- EMQX: Se trata de un bróker de mensajería de alto rendimiento que implementa el protocolo MQTT (Message Queuing Telemetry Transport). Proporciona alta disponibilidad y escalabilidad horizontal gracias a su capacidad para agruparse en clústeres sin nodo maestro. Además, incorpora un motor de reglas SQL que permite transformar, filtrar y enrutar mensajes en tiempo real. (EMQ Technologies, 2023)

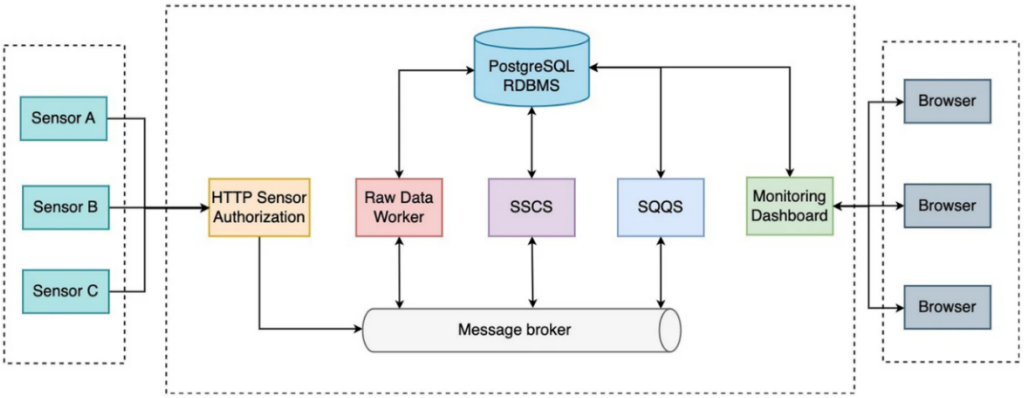
3.2 Estado del arte

3.2.1 Base de datos de series temporales múltiples sobre arquitectura de microservicios para un sistema de monitoreo del sueño basado en IoT

En esta investigación buscaban mejorar el sistema de monitoreo de un estudio anterior, que proponía una arquitectura basada en eventos y microservicios. Además, una base de datos única para todos los servicios y un sistema de administración de bases de datos relacionales (RDBMS). La mejora consiste en múltiples cambios en la arquitectura y tecnología del sistema. Se proponía una base de datos de series temporales y el servidor de Protocolo de telemetría de cola de mensajes (MQTT). (Mario San Emeterio, 2023)

Figura 4

Arquitectura con SQL y HTTP de un caso de estudio de IoT.



Adaptado de (Mario San Emeterio, 2023)

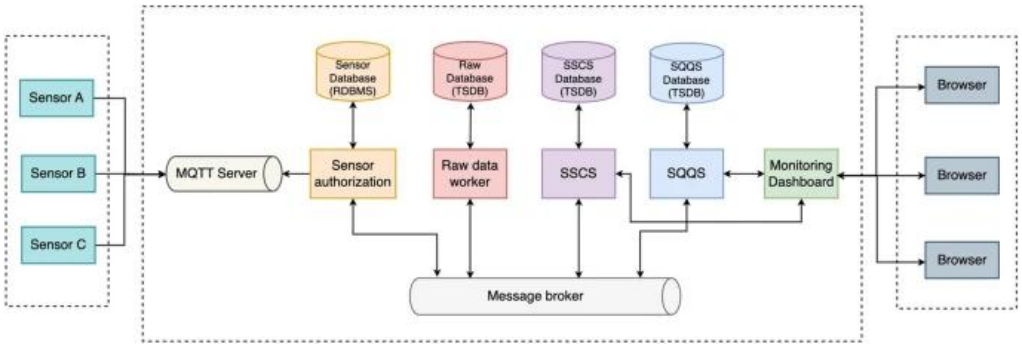
En la figura 4 se puede observar la arquitectura que se deseaba mejorar en el caso de estudio.

En la arquitectura que se propuso en el estudio, se incluye un sensor de autorización que opera a través del protocolo HTTP. Además, cada servicio dentro de esta arquitectura almacena los datos en intervalos de tiempo a través de bases de datos relacionales (RDBMS).

La arquitectura propuesta para mejorar el rendimiento de la anterior se puede evidenciar en la figura 5.

Figura 5

Arquitectura propuesta de un caso de estudio de IoT.



Adaptado de (Mario San Emeterio, 2023)

La propuesta del estudio busco optimizar la arquitectura de microservicios mediante la implementación de un bróker de mensajería basado en el protocolo MQTT para interconectar cada microservicio. Además, para los servicios que requerían gestionar datos en intervalos de tiempo, se utilizaron bases de datos de series temporales (TSDB). Por otro lado, el servicio responsable de la autorización conservo su base de datos relacional (RDBMS) para mantener la consistencia y eficiencia en el manejo de datos transaccionales.

Realizaron diversas pruebas a ambas arquitecturas para evaluar cuál de las dos propuestas ofrecía un mejor rendimiento:

- Evaluación de rendimiento: Utilizaron un sensor simulado con Apache JMeter, el sensor simulado envía 28.800 solicitudes y 57.600 solicitudes. (Mario San Emeterio, 2023)
- Evaluación de lectura de la base de datos: Llenaron la base de datos SSQS con 9600 mensajes y la base de datos SQQS con 1000 mensajes. Simulando diez clientes para enviar 100 solicitudes a cada uno. (Mario San Emeterio, 2023)

- Evaluación de escritura de base de datos: Se aseguraron de que la cantidad de datos en la base de datos sea 0 e iniciaron el consumidor para consumir y almacenar los datos en la base de datos. (Mario San Emeterio, 2023)

El investigador concluyo que, con base en las pruebas realizadas, la arquitectura y las tecnologías propuestas mejoraron significativamente el rendimiento. Su implementación de MQTT aumento el rendimiento del sistema a 2,43 veces más rápido que el sistema anterior. Con la función de escritura por lotes de InfluxDB notaron un aumento en el rendimiento de escritura de aproximadamente 20,95 veces más rápido. Y, Por último, la implementación de múltiples bases de datos para almacenar datos específicos le proporciono al sistema general una mejor escalabilidad, resiliencia e independencia. (Mario San Emeterio, 2023)

3.2.2 AWS e IoT para la monitorización médica remota en tiempo real

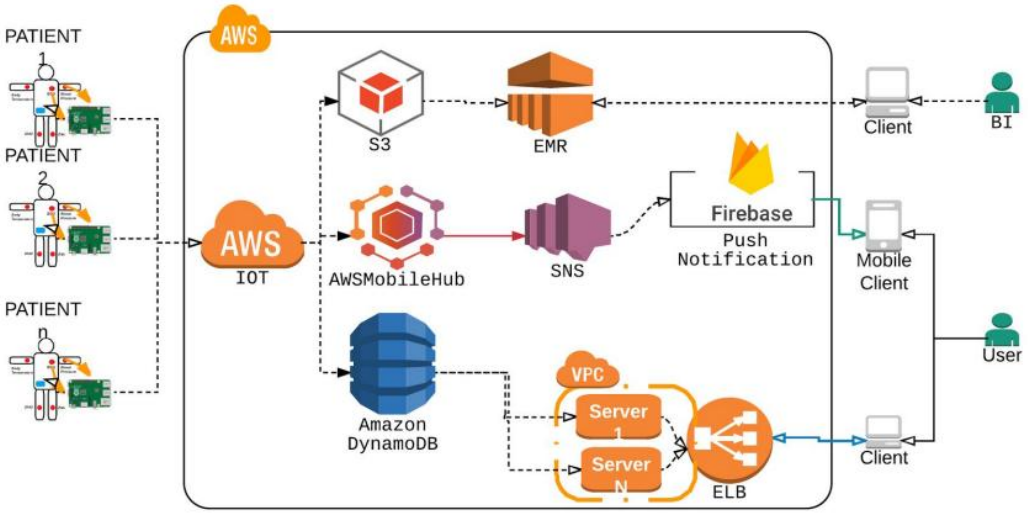
Este estudio propuso una arquitectura nueva para un sistema de monitoreo médico remoto que integraba computación en la nube e Internet de las Cosas (IoT). Aprovecharon los servicios de Amazon Web Services (AWS) para almacenar y analizar en tiempo real los datos provenientes de sensores médicos. Su objetivo era permitir el seguimiento de los parámetros de salud de los pacientes a distancia, facilitando una atención médica más eficiente y oportuna, especialmente para personas mayores con enfermedades crónicas. (Abdelilah Bouslama, 2019)

El sistema tenía sensores para registrar la saturación de oxígeno, la frecuencia cardíaca y la temperatura corporal. Los datos que se recopilaban se enviaban a los servidores en la nube mediante protocolos IoT, donde se analizaban con modelos de aprendizaje automático para

identificar posibles problemas de salud y enviar alertas tempranas a los profesionales médicos. (Abdelilah Bouslama, 2019)

Figura 6

Arquitectura del sistema de monitoreo de salud.



Adaptado de (Abdelilah Bouslama, 2019)

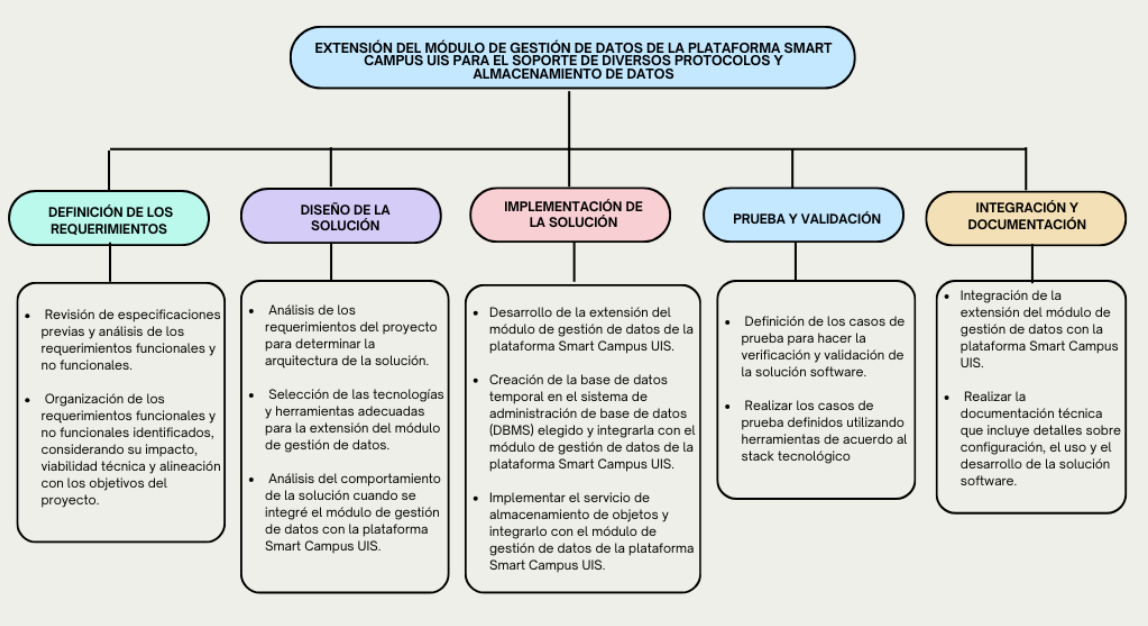
Para el autor del estudio, este enfoque mejoro la gestión de la salud de los pacientes, aligerando la carga de trabajo en los hospitales y aumento la calidad de vida de los pacientes mediante un monitoreo continuo y preciso. (Abdelilah Bouslama, 2019)

4. Metodología

En esta parte se explica el enfoque utilizado para llevar a cabo el proyecto, desde la definición de los requerimientos del sistema hasta la prueba y validación e integración y documentación en la plataforma *Smart Campus UIS*. Cada fase detalla las actividades concretas y los resultados previstos, como se puede ver en la Estructura de desglose del trabajo (EDT) basado en fases, garantizando un proceso organizado y consistente para lograr los objetivos del proyecto.

Figura 7

EDT basado en fases de la solución.



4.1 Definición de los requerimientos

En esta fase se identifican y documentan tanto los requerimientos funcionales como los no funcionales de la extensión del módulo de gestión de datos para el soporte de diversos protocolos y almacenamiento de datos, asegurando que todas las necesidades del proyecto sean comprendidas y estén alineadas con los objetivos de *Smart Campus UIS*

Actividades:

A.1.1. Revisión de especificaciones previas y análisis de los requerimientos funcionales y no funcionales.

A.1.2. Organización de los requerimientos funcionales y no funcionales identificados, considerando su impacto, viabilidad técnica y alineación con los objetivos del proyecto.

Resultados:

R.1.1. Lista de requerimientos funcionales y no funcionales claramente definidos, priorizados y validados.

4.2 Diseño de la solución

En esta fase se establece la arquitectura de la solución, especificando el stack tecnológico, las interacciones del módulo de gestión de datos y su integración para satisfacer los requerimientos funcionales y no funcionales del proyecto.

Actividades:

A.2.1. Análisis de los requerimientos del proyecto para determinar la arquitectura de la solución.

A.2.2. Selección de las tecnologías y herramientas adecuadas para la extensión del módulo de gestión de datos.

A.2.3. Análisis del comportamiento de la solución cuando se integré el módulo de gestión de datos con la plataforma *Smart Campus UIS.*

Resultados:

R.2.1. Documentación de la arquitectura de la solución que detalla el stack tecnológico y patrones de diseño seleccionados.

R.2.2. Plan de integración que describe cómo se comunicará el módulo de gestión de datos con la plataforma *Smart Campus UIS.*

4.3 Implementación de la solución

Durante esta fase se empieza a desarrollar la solución software, integrando funcionalidades y requerimientos establecidos en fases previas. Utilizando el stack tecnológico definido y aplicando buenas prácticas que respalden el diseño de la solución.

Actividades:

A.3.1. Desarrollo de la extensión del módulo de gestión de datos de la plataforma *Smart Campus UIS.*

A.3.2. Creación de la base de datos temporal en el sistema de administración de base de datos (DBMS) elegido e integrarla con el módulo de gestión de datos de la plataforma *Smart Campus UIS.*

A.3.3. Implementar el servicio de almacenamiento de objetos y integrarlo con el módulo de gestión de datos de la plataforma *Smart Campus UIS.*

Resultados

R.3.1. Extensión del módulo de gestión de datos de la plataforma *Smart Campus UIS* integrándole una base de datos temporal y un servicio de almacenamiento de objetos.

4.4 Prueba y validación

En esta fase se realizan las pruebas para determinar si la solución software cumple con los requerimientos para los cuales fue construida y encontrar los distintos escenarios donde se comporte de forma incorrecta o no conforme a su diseño.

Actividades:

A.4.1. Definición de los casos de prueba para hacer la verificación y validación de la solución software.

A.4.2. Realizar los casos de prueba definidos utilizando herramientas de acuerdo al stack tecnológico.

Resultado:

R.4.1. Reporte de las pruebas realizadas donde se pueda verificar que la solución software cumple con los requerimientos para los cuales fue construida.

4.5 Integración y documentación

En esta fase se realiza la integración de la solución con la plataforma *Smart Campus UIS*, teniendo en cuenta que funcione correctamente y se integre con los demás componentes de la plataforma. También, se realiza la documentación técnica necesaria para facilitar el uso y el mantenimiento continuo de la solución software.

Actividades:

A.5.1. Integración de la extensión del módulo de gestión de datos con la plataforma *Smart Campus UIS.*

A.5.2. Realizar la documentación técnica que incluye detalles sobre configuración, el uso y el desarrollo de la solución software.

Resultados:

R.5.1. Extensión del módulo de gestión de datos integrada con la plataforma *Smart Campus UIS.*

R.5.2. Documentación clara sobre la solución software garantizando su escalabilidad a lo largo de su ciclo de vida.

5. Requerimientos del módulo

Para dar inicio con el desarrollo del proyecto, se definieron las funcionalidades críticas que debe incorporar el módulo, así como los parámetros técnicos que necesarios para garantizar la interoperabilidad entre los diferentes protocolos y sistemas de gestión de bases de datos. De este modo, se asegurará un manejo eficaz y seguro de la información.

5.1 Requerimientos funcionales

RF1. El servidor permite almacenar los mensajes enviados por dispositivos IoT.

RF2. El servidor permite consultar mensajes almacenados.

RF3. El servidor permite almacenar videos con transmisión en tiempo real.

RF4. El servidor permite almacenar archivos.

RF5. El servidor permite seleccionar un subtópico y reencolarlo para que sean procesados por clientes relacionados con el subtópico.

5.1.1 Especificación detallada de los requerimientos funcionales

Tabla 1

Especificación del requerimiento que establece que el servidor permite almacenar los mensajes enviados por los sensores.

ID *RF1*	**Nombre**	*El servidor permite almacenar los mensajes enviados por los sensores.*	
Complejidad	*Baja*	**Prioridad**	*5*
Entrada	*Datos del mensaje, el header para metadatos y metrics para los valores medibles del sensor:* • *Header: userUUID, deviceId, timeStamp, location* • *Metrics: measurement, value*		
Descripción			
El sistema recibirá mensajes enviados por los sensores en un formato estándar. Estos mensajes serán almacenados en las bases de datos correspondientes.			
Precondición			
El sensor debe recolectar las medidas.			
Postcondición			
Se recibe el mensaje y luego se almacena en base de datos.			
Criterios de aceptación			
Una vez se envíen los datos en el formato estándar y por el tópico indicado. Se almacenará el mensaje en la base de datos correspondientes.			

Tabla 2

Especificación del requerimiento que establece que el servidor permite consultar los mensajes almacenados.

ID	*RF2*	**Nombre**	*El servidor permite consultar mensajes almacenados.*
Complejidad	*Media*	**Prioridad**	*5*

Entrada	*Parámetros de búsqueda según motor de almacenamiento:*
	• *Para InfluxDB:*
	  ○ *measurement (nombre de la métrica)*
	  ○ *start, end (fechas en formato YYYY-MM-DD)*
	  ○ *limit (número máximo de registros)*
	  ○ *time (unidad de agrupación de tiempo, ej. 1h, 30m)*
	• *Para MongoDB:*
	  ○ *deviceId (ID del dispositivo)*
	  ○ *location (ubicación)*
	  ○ *start, end (fechas en formato YYYY-MM-DD)*
	  ○ *measurement (nombre de la métrica)*
	  ○ *limit (número máximo de registros)*
	  ○ *time (unidad de agrupación de tiempo, ej. 1h, 30m))*		

Descripción			
Este requerimiento establece que el servidor ofrece distintos mecanismos para consultar los mensajes almacenados en las bases de datos InfluxDB y MongoDB, mediante controladores REST específicos. Las consultas pueden incluir filtros como rango de fechas, nombre de la métrica, ID del dispositivo o ubicación, y pueden devolver resultados agregados como valor mínimo, máximo y promedio.			
Precondición			
1. *Deben existir mensajes almacenados en al menos una de las bases de datos.* 2. *El servicio correspondiente debe estar disponible InfluxDB o MongoDB.* ***3.*** *El cliente debe utilizar los parámetros de entrada correctamente formateados.*			
Postcondición			
1. *Se devuelven los mensajes filtrados correctamente, en una estructura definida.* 2. *Si no hay resultados se devuelve una respuesta vacia.*			
Criterios de aceptación			
1. *El sistema permite recuperar datos usando múltiples filtros (fechas, métrica, dispositivo, etc.).* ***2.*** *El sistema distingue correctamente entre consultas dirigidas a InfluxDB y MongoDB.*			

Tabla 3

Especificación del requerimiento que establece que el servidor permite almacenar videos con transmisión en tiempo real.

ID	*RF3*	**Nombre**	*El servidor permite almacenar videos con transmisión en tiempo real.*
Complejidad	*Media*	**Prioridad**	*5*
Entrada	*Transmisión de video por el protocolo RTSP*		
Descripción			
El servidor permitirá conectarse por medio del protocolo RTSP y recibir video en tiempo real para ser guardada en una base de datos de objetos S3.			
Precondición			
Se debe guardar los datos de cámara en el servidor con la URL de conexión RTSP respectiva.			
Postcondición			
El servidor grabara el video recibido durante la conexión RTSP.			
Criterios de aceptación			
Una vez guardado los datos de la cámara y empezada la grabación con el servidor. Se guardará todo el video recibido durante la conexión RTSP.			

Tabla 4

Especificación del requerimiento que establece que el servidor permite almacenar archivos.

ID	*RF4*	**Nombre**	*El servidor permite almacenar archivos.*
Complejidad	*Media*	**Prioridad**	*5*
Entrada	*Un archivo enviado desde el cliente.*		
Descripción			
Este requerimiento permite a los usuarios subir archivos al sistema mediante una solicitud. Los archivos son almacenados de manera persistente en el sistema de archivos distribuido MinIO			
Precondición			
1. *El servicio de MinIO debe estar disponible.* 2. *El archivo debe ser enviado correctamente como parte del cuerpo de la solicitud.*			
Postcondición			
El archivo será almacenado exitosamente en MinIO bajo el nombre que tenía al momento de hacer la solicitud.			
Criterios de aceptación			
Una vez almacenado el archivo, se verídica que el archivo esté disponible en el dashboard de Minio.			

Tabla 5

Especificación del requerimiento que establece que el servidor servidor permite seleccionar un subtópico y reencolarlo.

ID *RF5*	**Nombre**	*El servidor permite seleccionar un subtópico y reencolarlo para que sean procesados por clientes relacionados con el subtópico.*	
Complejidad	*Media*	**Prioridad**	*5*
Entrada	*Al mensaje se le debe agregar al header el subtopico y una propiedad que indique si se desea reencolar el mensaje:* • *Header: userUUID, deviceId, timeStamp, location, shouldRequeue, topic* • *Metrics: measurement, value*		
Descripción			
El servidor permite seleccionar un subtópico y reencolarlo para que sean procesados por aplicaciones interesadas con la información de ese medidor en específico.			
Precondición			
El mensaje enviado debe tener datos de tópico donde se debe reencolar y si se desea reencolar ese mensaje.			
Postcondición			
El mensaje se guardará en las bases de datos y se reencola el mensaje a las aplicaciones interesadas en los datos del medidor			
Criterios de aceptación			
Después de recibido el mensaje se reencola el mensaje por el tópico especificado y a cada una de las aplicaciones interesadas.			

5.2 Requerimientos no funcionales

RNF1. Compatibilidad en la recepción de datos planos con protocolos MQTT y AMQP.

RNF2. Soporte del protocolo RTSP para la recepción de videos.

RNF3. Soporte del protocolo HTTP para la recepción de archivos.

RNF4. Mantener soporte de almacenamiento de datos en MongoDB.

RNF5. Implementar base de datos temporal con el gestor InfluxDB para el almacenamiento de datos planos.

RNF6. Implementar base de datos de objetos con el gestor Minio para el almacenamiento de datos no estructurados como videos e imágenes.

RNF7. Implementar la arquitectura modular Onion para mejorar la capacidad de crecimiento del módulo de gestión de datos.

RNF8. Estandarizar el formato y estructura de los mensajes que recibe el módulo de gestión de datos y son enviados por los dispositivos IoT.

RNF9. Estandarizar el formato y estructura de los mensajes de respuesta del módulo de gestión de datos que son consultados a través de las APIs de consulta.

El análisis de los requerimientos funcionales y no funcionales destaca la necesidad de ampliar el alcance del módulo. Para ello, la integración de nuevos protocolos y la incorporación de diversos sistemas de gestión de bases de datos contribuye a mejorar la escalabilidad y optimizar

la interacción entre dispositivos IoT y aplicaciones de consumo. A partir de esta base, en el siguiente capítulo se describirá en detalle el diseño del módulo, su arquitectura y los componentes clave que garantizan su correcto funcionamiento y su adaptabilidad a futuros escenarios.

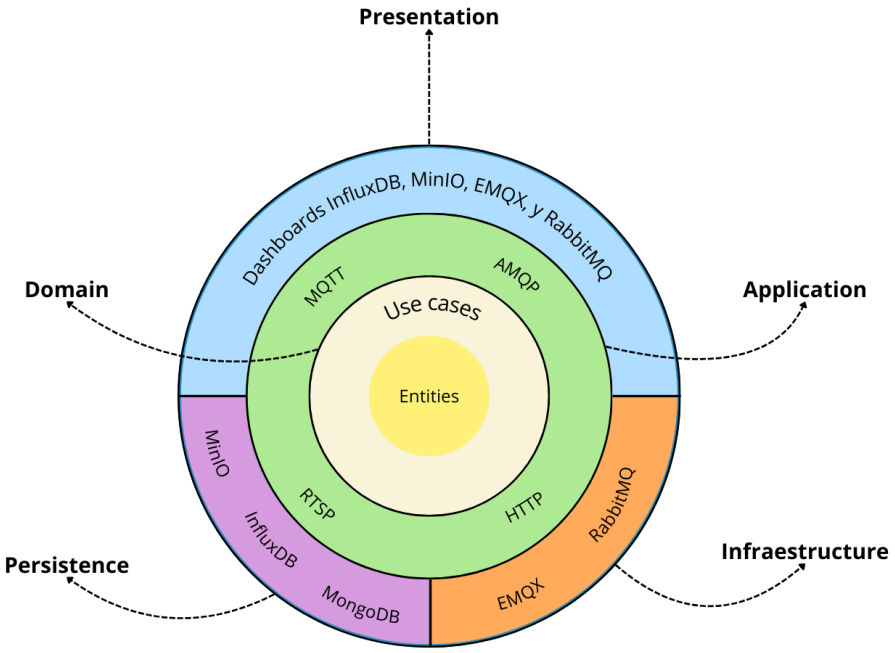
6. Diseño del módulo

Este capítulo tiene como objetivo presentar el esquema de arquitectura propuesto para el módulo de datos de la plataforma *Smart Campus UIS*, estableciendo las bases que garantizarán su escalabilidad, mantenibilidad y capacidad de evolución. Para ello, se traduce los requerimientos funcionales y no funcionales en una solución modular basada en la Aquitectura Onion, describiendo la organización de los módulos Application, Domain, Persistence y Service. Además, se describen las API diseñadas para la consulta de datos, la gestión de archivos y la gestión de cámaras para la captura de video.

6.1 Esquema de la arquitectura Onion aplicada a la solución

Figura 8

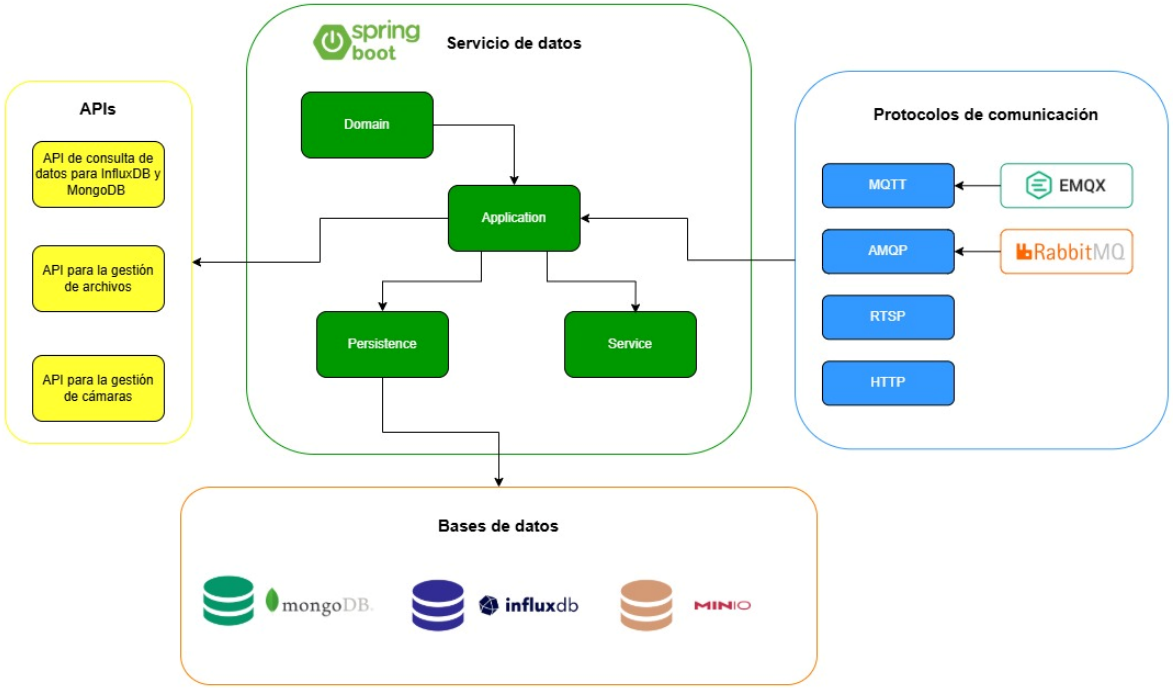
Arquitectura Onion de la solución.



6.2 Arquitectura de la solución

Figura 9

Arquitectura de la extensión realizada al módulo de datos



6.3 Descripción de la arquitectura de la solución

Para reorganizar la arquitectura del módulo de gestión de datos, se optó por buscar arquitectura modular que permitieran crecimiento continuo del módulo, que tuviera una facilidad de testear su uso, y mejorar la cohesión de las partes involucradas, entidades, bases de datos, repositorios entre otras.

Teniendo claros estos puntos se encuentran diferentes soluciones para lograrlo. La Arquitectura hexagonal y Arquitectura onion eran las más conocidas. Estas arquitecturas se basan en los principios de Arquitectura limpias haciendo que sus fundamentos sean los mismos.

Una de las principales diferencias entre las dos es que a la arquitectura hexagonal se basa en adaptadores y puertos, cosa que no se tiene en cuenta en la onion, haciendo que su implementación sea un poco más compleja.

Teniendo en cuenta que se basan en la misma ideología y la implementación de la onion era mucho más factible para resolver la problemática, se tomó la decisión final de reorganizar el módulo gestión con la arquitectura onion.

Esta arquitectura fue propuesta por Jeffrey Palermo, un arquitecto de software experimentado. Se organiza en diferentes capas con un propósito claro, de ahí el nombre que recibe. En el centro, encontramos las entidades del dominio y las reglas de negocio, rodeadas por capas que incluyen servicios de aplicación, interfaces y parte de infraestructura. La dependencia entre capas debe ser siempre de afuera hacia adentro, lo que facilita la mantenibilidad y la independencia de frameworks o tecnologías externas.

En la implementación de la arquitectura onion para el módulo de gestión de datos tenemos los siguientes módulos:

- Application

Este módulo se encarga de exponer las funcionalidades del sistema e interactúa con los módulos de dominio y persistencia. En él se definen los controladores que gestionan las solicitudes y exponen endpoints para interactuar con distintos recursos. Además, el módulo integra

componentes para gestionar la mensajería utilizando los protocolos AMQP y MQTT. Asimismo, se establecen los parámetros de conexión con los brokers de mensajería implementados (RabbitMQ y EMQX). Se incluyen también clases para manejar operaciones en hilos separados, facilitando la ejecución de tareas en paralelo. Se ha considerado el manejo de excepciones para la gestión de errores específicos y se han realizado transformaciones de los datos con el fin de separar de manera clara las responsabilidades entre capas.

- Domain

En este módulo agrupamos todas las entidades que hacen parte de la lógica de negocio. Convirtiendo este módulo en el núcleo de la arquitectura. También definimos las interfaces de repositorio, el uso de estas interfaces nos permite no depender de un tipo de base de datos o implementación concreta. Al trabajar siempre bajo el mismo contrato, mejoramos la cohesión entre módulos y aislamos nuestro núcleo de todos los componentes externos permitiendo una mejor la mantenibilidad y crecimiento del sistema.

- Persistence

En este módulo tenemos todas las configuraciones, excepciones y conexiones enfocadas en las bases de datos utilizadas por el servidor. Aquí se implementan las interfaces de repositorio respetando el contrato definido en la capa de dominio. Esto permite que el servidor pueda guardar mensajes en dos bases de datos diferentes de forma totalmente transparente, cada una con su implementación especifica, sin impactar en la lógica de negocio.

Cuando se necesite cambiar de base de datos, solo se debe mantener la implementación de estas interfaces, de este modo, no se verá afectado ninguna otra parte del servidor.

- Service

Este módulo implementa la lógica de servicios relacionada con la reencolación de mensajes a través de los protocolos AMQP Y MQTT, así como las consultas a la base de datos de InfluxDB. Este módulo depende del dominio, lo que garantiza que la lógica de negocio central permanezca separada de las preocupaciones de la infraestructura, un principio fundamental en la arquitectura onion.

6.4 API de consulta de datos

En esta sección se describe la estructura de la API destinada a la consulta de datos almacenados en InfluxDB y MongoDB.

- Para InfluxDB: Se implementa un controlador encargado de gestionar consultas relacionadas con mediciones, incluyendo la obtención de las últimas mediciones, consultas por rango de tiempo, agrupaciones por unidades de tiempo y cálculos como promedio, mínimo y máximo.
- Para MongoDB: Se implementa un controlador que permita llevar una trazabilidad de los mensajes que recibe el módulo. Se puede hacer filtrados por identificador de dispositivo, ubicación y rango de fechas, así como la recuperación de las últimas mediciones de una métrica especifica.

6.5 API para la gestión de archivos

En esta sección describe la estructura de la API destinada para la gestión de archivos con persistencia en el gestor de base de datos MinIO.

Se implementa un controlador responsable de manejar el almacenamiento de diversos tipos de archivos, tales con imágenes, documentos PDF, archivos CSV y archivos de configuración.

6.6 API para la gestión de cámaras

En esta sección se describe la estructura de la API destinada a la gestión de cámaras para la transmisión y grabación de video.

Se implementa un controlador encargado de administrar las operaciones de grabación, incluyendo inicio, pausa, reanudación y detención, así como el registro y eliminación de cámaras en el sistema.

En conclusión, hemos validado como se adoptó la Arquitectura Onion aportando cohesión y la separación de responsabilidades necesario para un crecimiento ordenado del módulo de datos, desglosando los módulos que la componen, así como las rutas de las API de datos, archivos y cámaras. Con este diseño consolidado, el siguiente paso sería materializarlo. En el capítulo de implementación de la solución abordaremos la puesta en marcha del diseño con la configuración de protocolos de comunicación, bases de datos especializadas y la integración de los brokers de mensajería.

6.7 Formato de los mensajes que recibe el módulo de datos

El formato de los mensajes enviados al módulo de datos es JSON y contiene dos partes:

- **Header:** Para los metadatos del mensaje
  - userUUID: Es el identificador único universal de cada usuario. Tipo: String.
  - deviceID: Es el identificador asignado al dispositivo cuando es registrado en la plataforma *Smart Campus UIS.* Tipo: String.
  - timestamp: Es la fecha y hora en la que se generó el mensaje, en formato ISO 8601 (UTC). Tipo: String.
  - location: Es la ubicación donde se tomó la medición. Tipo: String.
  - topic: Es el tópico del mensaje. Tipo: String.
  - shouldRequeue: Indica si se desea reencolar el mensaje. Tipo: boolean.
- **Metrics:** Una lista de las mediciones asociadas al mensaje
  - measurement: Indica el nombre de la medición. Tipo: String.
  - value: Indica el valor de la medición. Tipo: Double.

En la Figura 10 podemos observar un ejemplo de este formato.

Figura 10

Ejemplo formato mensajes enviados

6.8 Formato de los mensajes consultados de InfluxDB en el módulo de datos

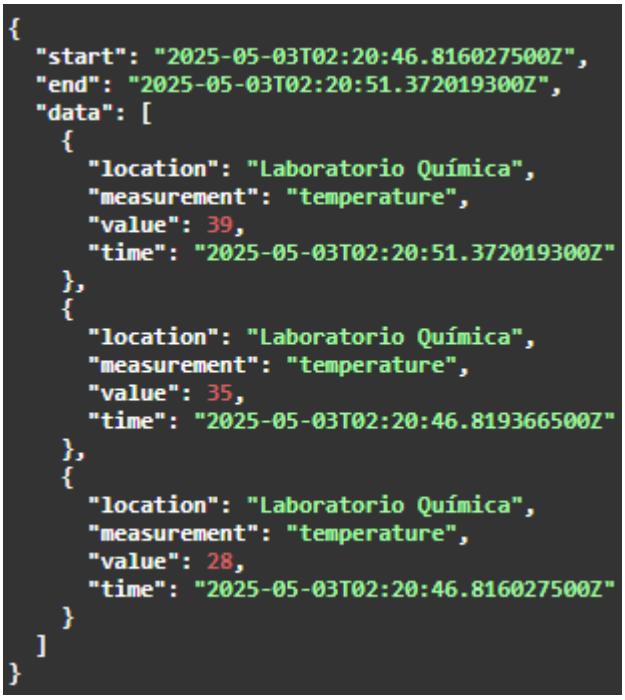
El formato de los mensajes consultados de InfluxDB al módulo de datos es JSON y contiene los siguientes atributos:

- start: Es el instante inicial del rango de tiempo consultado, en formato ISO 8601 (UTC). Tipo: String.
- end: Es el instante final del rango de tiempo consultado, en formato ISO 8601 (UTC). Tipo: String.
- Data: Contiene cada uno de los registros con los siguientes atrubutos
    - location: Es la ubicación donde se tomó la medición. Tipo: String.
    - measurement: Indica el nombre de la medición. Tipo: String.
    - value: Indica el valor de la medición. Tipo: Double.
    - times: Es la fecha y hora en la que se generó el mensaje, en formato ISO 8601 (UTC).

En la Figura 11 podemos observar un ejemplo de este formato.

Figura 11

Ejemplo formato mensajes consultados



```
{
  "start": "2025-05-03T02:20:46.816027500Z",
  "end": "2025-05-03T02:20:51.372019300Z",
  "data": [
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 39,
      "time": "2025-05-03T02:20:51.372019300Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 35,
      "time": "2025-05-03T02:20:46.819366500Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 28,
      "time": "2025-05-03T02:20:46.816027500Z"
    }
  ]
}
```

7. Implementación de la solución

Este capítulo tiene como objetivo describir con detalle cómo se materializa el diseño del módulo planteado anteriormente. En este apartado se abordan los protocolos de comunicación soportados, la integración de bases de datos especializadas. Asimismo, se especifican las rutas expuestas por la API para consulta de métricas, gestión de archivos y control de cámaras.

7.1 Protocolos

Uno de los principales objetivos del proyecto es lograr que el módulo de gestión de datos de la plataforma Smart Campus UIS permita soportar múltiples protocolos de comunicación. A continuación, se describen los diferentes protocolos que ahora soporta este módulo, su propósito y caso de uso.

- AMQP

El protocolo AMQP es muy utilizado en soluciones IoT debido a su eficiencia en él envió de mensajes. Este protocolo no se encontraba soportado por el módulo de gestión de datos. El propósito de su integración es el de recibir los mensajes enviados por los sensores.

Para soportar este protocolo se implementó el bróker RabbitMQ. Los mensajes deben ser enviados al tópico device-message, utilizando un intercambiador de tipo fanout, lo que permite su difusión a todos los consumidores suscritos.

- MQTT

El protocolo MQTT es el más elegido en soluciones IoT por su facilidad de uso, además de ser un protocolo bastante liviano, perfecto para dispositivos IoT sin tanta potencia computacional. Este protocolo ya se encontraba disponible en la plataforma, por lo que se decidió mantener su implementación. Dando la posibilidad al desarrollador de utilizar otro protocolo que puede ser útil para su caso de uso.

Si bien el protocolo MQTT se mantuvo, se escogió una implementación mucho más robusta para su soporte, por lo que se sustituyó el bróker de comunicación Mosquitto por EMQX. EMQX es una implementación que ofrece un nivel de servicio con mucha mejor calidad con respecto a Mosquitto. El tópico de envió de mensajes continúa siendo device/message.

- RTSP

El módulo de gestión de datos ahora cuenta con soporte para comunicación de video en tiempo real mediante el protocolo RTSP. Esto permite recibir video de un servidor RTSP que transmite contenido multimedia. Este protocolo fue implementado con una librería de java que internamente utiliza FFmepg. Para utilizar esta funcionalidad se debe proveer la URL de conexión con formato RTSP.

- HTTP

Para soportar él envió o guardado de archivos el módulo de gestión de datos utiliza HTTP, un protocolo estándar en la web. También funciona durante las peticiones a la API para consultar de datos en las diferentes bases de datos que maneja el módulo.

7.2 Bases de datos

InfluxDB, MongoDB y MinIO fueron seleccionadas como base de datos especializadas para abordar los distintos tipos de datos que gestiona la plataforma Smart Campus UIS, cumpliendo así con los objetivos de flexibilidad, escalabilidad y eficiencia del módulo de datos.

- InfluxDB fue utilizada como base de datos temporal para almacenar series de datos provenientes de sensores y dispositivos IoT. Su arquitectura está diseñada para la gestión de datos con marcas de tiempo, lo que permite realizar consultas en intervalos de tiempo específicos, facilitando el análisis histórico y el monitoreo en tiempo real de los datos. Esto responde a la necesidad identificada de contar con una solución más adecuada que MongoDB para consultas basadas en tiempo, esenciales en un entorno de señores y dispositivos que generan datos continuamente.

- MongoDB se mantuvo como base de datos documental, dado su enfoque flexible y su capacidad para almacenar datos estructurados y semi-estructurados en formato JSON. Su inclusión se justifica por la necesidad de conservar la compatibilidad con la versión del

módulo anterior y gestionar datos que no requiere una estructura temporal estricta, como configuraciones, metadatos o descripciones asociadas a dispositivos. Además, se decidió utilizar MongoDB para almacenar los mensajes recibidos a través del sistema, tratándolos como logs que permiten llevar un historial detallado de las comunicaciones, facilitando tareas de trazabilidad y análisis posterior del flujo de datos. De esta manera, MongoDB permiten mantener la extensibilidad del módulo y su integración de otros servicios de la plataforma Smart Campus UIS.

- MinIO fue incorporado como sistema de almacenamiento de objetos para gestionar datos no estructurados como imágenes y videos. Su compatibilidad con el protocolo y su enfoque en la alta disponibilidad y escalabilidad lo convierten en una solución óptima para gestionar contenido multimedia. Con MinIO, se garantiza el soporte a protocolos que transmiten archivos pesados, resolviendo la limitación que tenía el módulo de datos basada únicamente en MQTT y almacenamiento documental.

7.3 Lenguaje de programación

El módulo de gestión de datos poseía una implementación en Java por medio de Spring Boot, aunque se evalúo hacer un cambio a JavaScript y NestJs para buscar una solución más modular, se mantuvo el soporte de estas tecnologías por el mayor conocimiento sobre las mismas y se optó por reorganizar el módulo por una arquitectura más modular, que facilitara su mantenimiento y escalabilidad a futuro.

7.4 Brokers de mensajería

Para la extensión del módulo de gestión de datos de la plataforma Smart Campus UIS, fue necesario implementar un enforque que permita soportar múltiples protocolos de comunicación, como AMQP y MQTT, y garantizar una transmisión de datos eficiente y escalable. Con la implementación de estos dos protocolos, se utilizaron dos brokers de mensajería: RabbitMQ y EMQX, cada uno se seleccionó por características específicas y la capacidad de adaptación a distintos tipos de datos y escenarios de uso.

- RabbitMQ es elegido para el manejo del protocolo AMQP, resulta ideal para escenarios donde se manejan grandes cargas de trabajo. Este bróker ofrece características como enrutamiento, confirmación de entrega, persistencia de mensajes y control de flujos, elementos clave para aplicaciones que requieren una alta fiabilidad en la transmisión de datos

- EMQX fue seleccionado como bróker para el manejo del protocolo MQTT. Dado que la plataforma Smart Campus UIS ya utilizaba MQTT como medio de comunicación entre dispositivos y el backend, la incorporación de EMQX representa una evolución que proporcionar mayor escalabilidad, clustering y monitoreo en tiempo real, lo que permite mantener y robustecer la infraestructura existente.

7.5 API de Consulta de Datos

La API define rutas que facilitan el acceso a los datos de series temporales y al historial de mensajes. Se presentan las siguientes rutas específicas:

Tabla 6

Endpoints API de Consulta de Datos

Base de Datos	Endpoint	Metodo	Acción
InfluxDB	/influx/measurement/{measurement}/min	GET	Obtiene el valor mínimo de una métrica.
	/influx/measurement/{measurement}/max	GET	Obtiene el valor máximo de una métrica.
	/influx/measurement/{measurement}/last	GET	Obtiene una cantidad especifica de mediciones de una métrica.
	/influx/measurement/{measurement}/average	GET	Obtiene el promedio de una métrica en un rango de tiempo específico.
	/influx/date/units/{time}	GET	Obtiene las mediciones de las métricas agrupadas por unidad de tiempo.
	/influx/by-time-range	GET	Obtiene las mediciones de todas las métricas en un rango de tiempo específico.

	/influx/by-time-range/measurement/{measurement}	GET	Obtiene las mediciones de una métrica en un rango de tiempo específico.
MongoDB	/mongo/measurement/last	GET	Obtiene una cantidad especifica de mensajes de una métrica en un rango de tiempo específico.
	/mongo/measurement/by-time-range	GET	Obtiene los mensajes de mediciones de una métrica en un rango de tiempo específico.
	/mongo/location/{location}	GET	Obtiene los mensajes de mediciones de una ubicación específica.
	/mongo/deviceId/{deviceId}	GET	Obtiene los mensajes de mediciones de un dispositivo en específico.
	/mongo/date/units/{time}	GET	Obtiene los mensajes de mediciones agrupadas por unidad de tiempo.

7.6 API para la gestión de archivos

La API expone un endpoint para subir y almacenar archivos:

Tabla 7

Enpoint API para la gestión de archivos

Base de Datos	Endpoint	Método	Acción
MinIO	/file/save	POST	Subir y almacenar archivos

7.7 API para la gestión de cámaras

La API dispone de endpoints para iniciar, pausar, reanudar y detener la grabación de las cámaras, mediante las siguientes rutas:

Tabla 8

Endpoints API para la gestión de cámaras

Base de Datos	Endpoint	Método	Acción
MongoDB	/camera/add	POST	Registra una nueva cámara en el sistema.
	/camera/list	GET	Devuelve una lista de todas las cámaras registradas.
	/camera/delete	DELETE	Elimina una cámara específica del sistema utilizando su ID.

MinIO	/camera/stream?IdCamera	GET	Inicia la transmisión en vivo de una cámara específica.
	/camera/stop?IdCamera	GET	Detiene la grabación activa de una cámara, finalizando el proceso de almacenamiento de video.
	/camera/start?IdCamera	GET	Comienza la grabación de una cámara, iniciando un proceso en segundo plano que almacena el video durante un período definido.
	/camera/resume?IdCamera	GET	Reanuda la grabación de una cámara.
	/camera/pause?IdCamera	GET	Pausa temporalmente la grabación de una cámara.

Todas las API expuestas anteriormente se encuentran documentadas por medio de Swagger, una herramienta de código abierto que permite hacer documentación de APIs de manera automatizada.

La implementación se encuentra disponible en el siguiente repositorio de GitHub[1]. Este contiene el código fuente organizado según la arquitectura Onion descrita en el capítulo de "Diseño del módulo".

Para concluir, en este capítulo se ha concretado el diseño del módulo de datos. La integración de diversos protocolos de comunicación y sistemas de gestión de bases de datos permite que este módulo sea más eficiente en la transmisión y gestión de la información, cumpliendo así con los requisitos funcionales y no funcionales. A continuación, en el capítulo "Validación de la solución" se presentarán los resultados de las evaluaciones funcionales y de las pruebas unitarias, lo cual aportará fiabilidad a la implementación.

[1] Enlace al repositorio de GitHub: https://github.com/JulianCastillo14/data_microservice.git

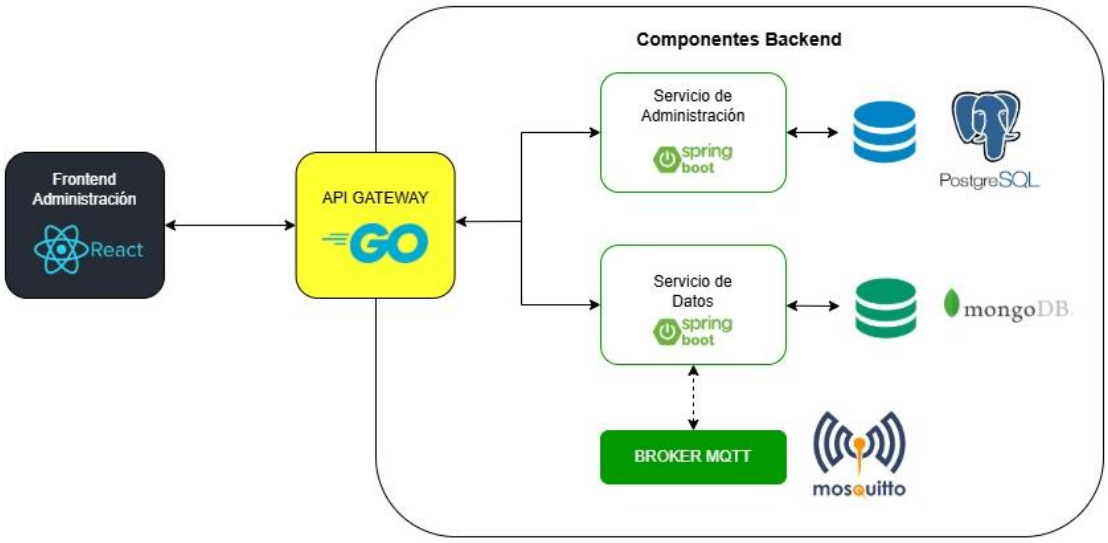
7.8 Arquitectura de la plataforma Smart Campus UIS antes y después de la implementación

En esta sección podemos observar gráficamente el cambio en la plataforma *Smart Campus UIS* tras la extensión del módulo de datos:

Smart Campus UIS antes de la extensión:

Figura 1

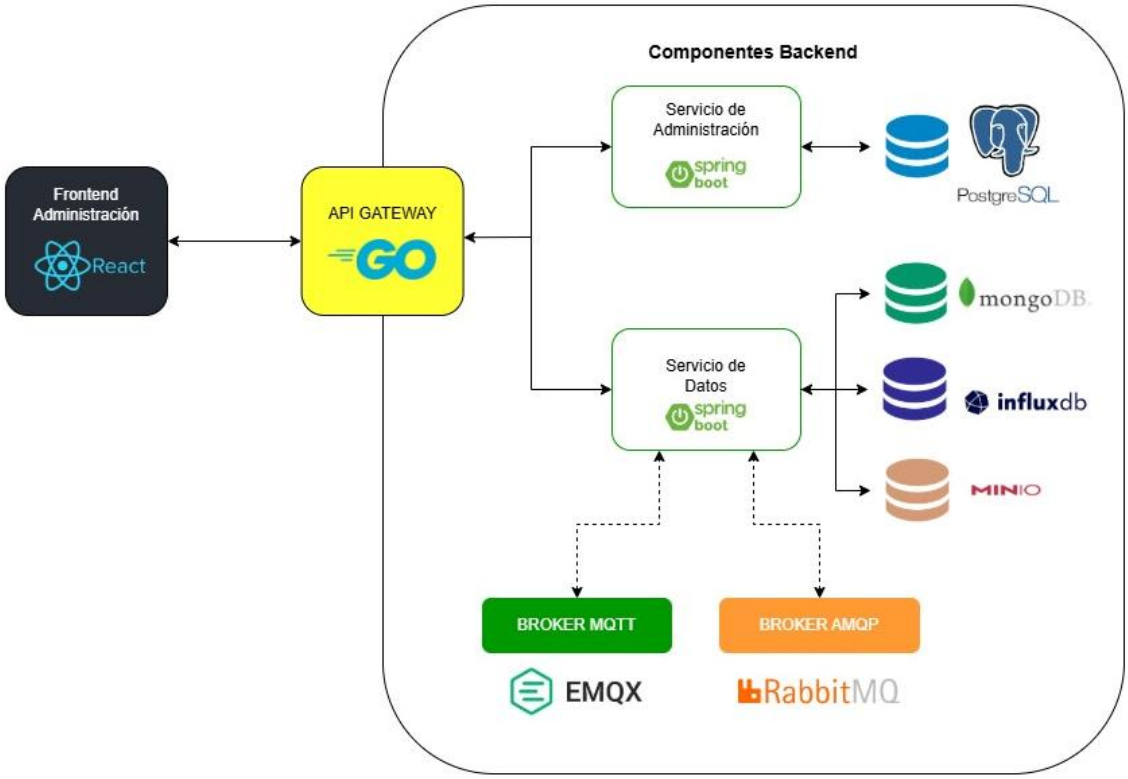
Arquitectura de Smart Campus antes de la extensión



Smart Campus UIS después de la extensión:

Figura 12

Arquitectura de Smart Campus después de la extensión



8. Validación de la solución

En este capítulo se presenta la validación técnica de la solución, abarcando unas evaluaciones funcionales y pruebas unitarias. Se evaluaron operaciones clave como el almacenamiento, consultas de datos, gestión de cámaras y control de video, utilizando herramientas como Mockito y Junit.

8.1 Evaluación funcional

Tabla 9

Resumen de la evaluación para la función de guardar mensaje sin re encolado.

Funcionalidad	*Guardar mensaje sin re encolado* **Cumple** *Si*
Descripción	*Verifica que al guardar un mensaje cuyo Header tiene el valor de false en el atributo shouldRequeue, se guarde correctamente y no se re encole el mensaje.*
Acción	1. *Se configura el entorno de prueba en @BeforeEacg inyectando mocks y estableciendo el bucket.* 2. *Se crea un mensaje con:* *shouldRequeue = false* 3. *Simular el almacenamiento del mensaje.*
Resultado esperado	*Se retorna el mismo objeto de mensaje recibido.*
Observaciones	*Se utiliza Mockito para simular la interacción con InfluxDB (mediante writeApiBlocking) y se inyecta el fluxRecordMapper mediante reflexión para configurar correctamente el objeto a testear.*

Tabla 10

Resumen de la evaluación para la función de guardar mensaje con re encolado.

Funcionalidad	*Guardar mensaje con re encolado*	**Cumple**	*Si*
Descripción	*Verifica que al guardar un mensaje cuyo Header tiene el valor de true en el atributo, se guarde correctamente y se ejecute el reencolado del mensaje.*		
Acción	1. *Se configura el entorno de prueba en @BeforeEacg inyectando mocks y estableciendo el bucket* 2. *Se crea un mensaje con:* *shouldRequeue = ftrue* 3. *Simular el almacenamiento del mensaje.* 4. *Simular el reencolado mediante requeueMessage.*		
Resultado esperado	*Se retorna el mismo objeto de mensaje recibido.*		
Observaciones	*Se utiliza Mockito para simular la operación de escritura y el reencolado. Es fundamental que el mock de messageRequeueService se configure para capturar la invocación con el mensaje exacto.*		

Tabla 11

Resumen de la evaluación para la función de guardar datos de una cámara.

Funcionalidad	*Guardar datos de una cámara*	**Cumple**	*Si*
Descripción	*Se envía los datos de la cámara en el formato JSON con la estructura establecida.*		
Acción	*Se valida que no exista una cámara con el mismo nombre o URL y se guarda en base de datos.*		
Resultado esperado	*Se almacena la cámara exitosamente.*		
Observaciones	*Método para añadir cámara en el ControllerCamera.*		

Tabla 12

Resumen de la evaluación para la función de obtener lista de las cámaras.

Funcionalidad	*Obtener lista de las cámaras*	**Cumple**	*Si*
Descripción	*Se hace petición para recibir la lista de cámaras agregadas anteriormente.*		
Acción	*Se recibe la petición post para listar de cámaras agregadas.*		
Resultado esperado	*Se reciban el listado de cámaras con la estructura asignada al DTO.*		
Observaciones	*Método para listar cámaras en el ControllerCamera.*		

Tabla 13

Resumen de la evaluación para la función de eliminar una cámara.

Funcionalidad	*Eliminar Cámara*	**Cumple**	*Si*
Descripción	*Se hace petición para eliminar una de las cámaras agregadas anteriormente.*		
Acción	*Se recibe un IdCamera relacionada con la cámara que se desea eliminar.*		
Resultado esperado	*Se obtienen una petición con estado de respuesta 200 y un mensaje de confirmación de la eliminación.*		
Observaciones	*Método para eliminar cámara en el ControllerCamera.*		

Tabla 14

Resumen de la evaluación para la función de empezar video.

Funcionalidad	*Empezar video*	**Cumple**	*Si*
Descripción	*Se hace petición al servicio de data para permitir que una cámara comience a guardar video.*		
Acción	*Se envía la petición al endpoint de empezar con un idCamera.*		
Resultado esperado	*Se obtienen un estado de respuesta de 200 y un json con datos de la cámara y su estado actualizado a grabando.*		
Observaciones	*Método para empezar con el guardado de la cámara en el ControllerCamera.*		

Tabla 15

Resumen de la evaluación para la función de pausar video.

Funcionalidad	*Pausar video*	**Cumple**	*Si*
Descripción	*Se hace petición al servicio de data para no permitir que una cámara siga guardando video.*		
Acción	*Se envía la petición al endpoint de pausa con un idCamera*		
Resultado esperado	*Se obtienen un estado de respuesta de 200 y un json con datos de la cámara y su estado actualizado a parado.*		
Observaciones	*Método para pausar el guardado de la cámara en el ControllerCamera.*		

Tabla 16

Resumen de la evaluación para la función de reanudar video.

Funcionalidad	*Reanudar video*	**Cumple**	*Si*
Descripción	*Se hace petición al servicio de data para continuar con el guardado de video.*		
Acción	*Se envía la petición al endpoint de reanudar con un idCamera.*		
Resultado esperado	*Se obtienen un estado de respuesta de 200 y un json con datos de la cámara y su estado actualizado a parado.*		
Observaciones	*Método para reanudar el guardado de la cámara en el ControllerCamera.*		

Tabla 17

Resumen de la evaluación para la función de parar video.

Funcionalidad	*Parar video*	**Cumple**	*Si*
Descripción	*Se hace petición al servicio de data para parar el guardado de video.*		
Acción	*Se envía la petición al endpoint de parar con un idCamera.*		
Resultado esperado	*Se obtienen un estado de respuesta de 200 y un json con datos de la cámara y su estado actualizado a parado.*		
Observaciones	*Método para parar el guardado de la cámara en el ControllerCamera.*		

8.2 Pruebas unitarias

Tabla 18

Resumen prueba unitaria PUO1.

ID DE PRUEBA PU01	
Valor de prueba	**Acción**
Measurement = C02 From = fecha actual con 1800 segundos menos To = fecha actual	Obtener registros de una métrica en un rango de tiempo de la Base de datos de MongoDB
Resultado esperado	**Resultado obtenido**
Figura 13 *Resultado esperado de la prueba PU01.*	**Figura 14** *Resultado obtenido de la prueba PU01.*


```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```



```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```

Tabla 19

Resumen prueba unitaria PUO2.

ID DE PRUEBA PU02	
Valor de prueba	**Acción**
Limit = 5 Measurement = Temperature	Obtener los “n” últimos registros de una métrica de la base de datos de MongoDB
Resultado esperado	**Resultado obtenido**
Figura 15 *Resultado esperado de la prueba PU02.*	**Figura 16** *Resultado obtenido de la prueba PU02.*


```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```



```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```

Tabla 20

Resumen prueba unitaria PUO3.

ID DE PRUEBA PU03	
Valor de prueba	**Acción**
Location = CENTIC	Obtener los registros de métricas por ubicación del sensor de la base de datos de MongoDB
Resultado esperado	**Resultado obtenido**
Figura 17 *Resultado esperado de la prueba PU03.*	**Figura 18** *Resultado obtenido de la prueba PU03.*


```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```



```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```

Tabla 21

Resumen prueba unitaria PUO4.

ID DE PRUEBA PU04	
Valor de prueba	**Acción**
DevidiceId = 4545	Obtener los registros de métricas por deviceId del sensor de la base de datos de MongoDB
Resultado esperado	**Resultado obtenido**
Figura 19 *Resultado esperado de la prueba PU04.*	**Figura 20** *Resultado obtenido de la prueba PU04.*


```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```



```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```

Tabla 22

Resumen prueba unitaria PUO5.

ID DE PRUEBA PU05	
Valor de prueba	**Acción**
From = fecha actual con 3600 segundos menos To = fecha actual	Obtener los registros de métricas en un rango de tiempo de la base de datos de MongoDB
Resultado esperado	**Resultado obtenido**
Figura 21 *Resultado esperado de la prueba PU05.*	**Figura 22** *Resultado obtenido de la prueba PU05.*


```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```



```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```

Tabla 23

Resumen prueba unitaria PUO6.

ID DE PRUEBA PU06	
Valor de prueba	**Acción**
Time = 10m	Obtener los registros de métricas por magnitud de tiempo de la base de datos de MongoDB
Resultado esperado	**Resultado obtenido**
Figura 23 *Resultado obtenido de la prueba PU06.*	**Figura 24** *Resultado obtenido de la prueba PU06.*


```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```



```
[
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":20.0
         },
         {
            "measurement":"CO2",
            "value":10.0
         }
      ]
   },
   {
      "headers":{
         "userUUID":"25418",
         "deviceId":"4545",
         "timeStamp":"2025-03-30T21:08:16.446966773Z",
         "location":"CENTIC",
         "topic":"Ambiente",
         "shouldRequeue":true
      },
      "metrics":[
         {
            "measurement":"Temperature",
            "value":22.0
         },
         {
            "measurement":"CO2",
            "value":8.0
         }
      ]
   }
]
```

Tabla 24

Resumen prueba unitaria PUO7.

ID DE PRUEBA PU07	
Valor de prueba	**Acción**
start = "2023-01-01T00:00:00Z" end = "2023-01-02T00:00:00Z" measurement = temperature	Obtener el valor mínimo de una métrica en un rango de tiempo de la base de datos de InfluxDB
Resultado esperado	**Resultado obtenido**
15.0	15.0

Tabla 25

Resumen prueba unitaria PUO8.

ID DE PRUEBA PU08	
Valor de prueba	**Acción**
start = "2023-01-01T00:00:00Z" end = "2023-01-02T00:00:00Z" measurement = temperature	Obtener el valor máximo de una métrica en un rango de tiempo de la base de datos de InfluxDB
Resultado esperado	**Resultado obtenido**
30.0	30.0

Tabla 26

Resumen prueba unitaria PUO9.

ID DE PRUEBA PU09	
Valor de prueba	**Acción**
start = "2023-01-01T00:00:00Z" end = "2023-01-02T00:00:00Z" measurement = temperature	Obtener el valor promedio de una métrica en un rango de tiempo de la base de datos de InfluxDB
Resultado esperado	**Resultado obtenido**
23.5	23.5

Tabla 27

Resumen prueba unitaria PU10.

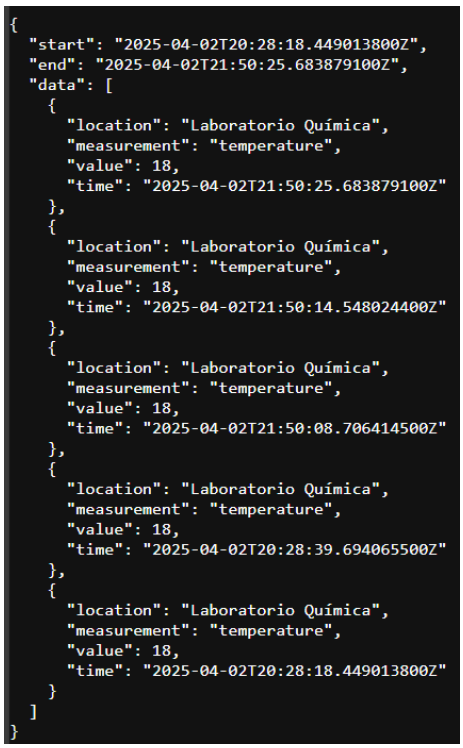
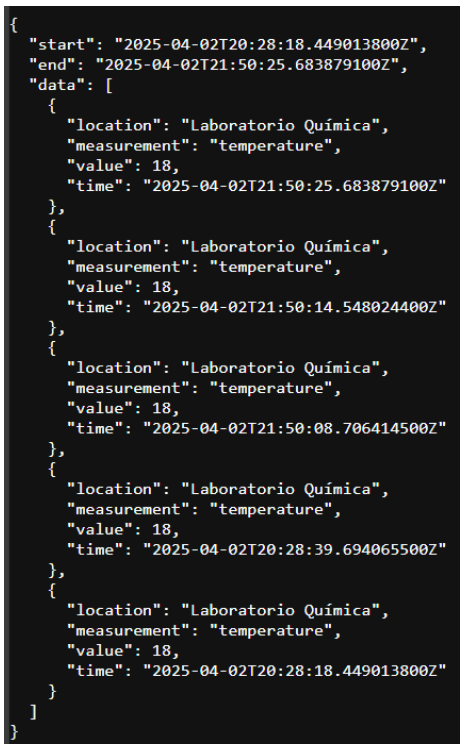
ID DE PRUEBA PU010	
Valor de prueba	**Acción**
limit = 5 measurement = temperature	Obtener los “n” últimos registros de una métrica de la base de datos de InfluxDB
Resultado esperado	**Resultado obtenido**
Figura 25 *Resultado esperado de la prueba PU10.* 	**Figura 26** *Resultado obtenido de la prueba PU10.* 

Tabla 28

Resumen prueba unitaria PU11.

ID DE PRUEBA	**PU011**
Valor de prueba	**Accion**
start = " 2025-04-02" end = " 2025-04-03 " measurement = temperature	Obtener registros de una métrica en un rango de tiempo de la Base de datos de InfluxDB
Resultado esperado	**Resultado obtenido**
Figura 27 *Resultado esperado de la prueba PU11.*	**Figura 28** *Resultado obtenido de la prueba PU11.*


```
{
  "start": "2025-04-02T20:27:30.533556100Z",
  "end": "2025-04-02T21:50:25.683879100Z",
  "data": [
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T20:27:30.533556100Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T20:28:18.449013800Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T20:28:39.694065500Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T21:50:08.706414500Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T21:50:14.548024400Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T21:50:25.683879100Z"
    }
  ]
}
```



```
{
  "start": "2025-04-02T20:27:30.533556100Z",
  "end": "2025-04-02T21:50:25.683879100Z",
  "data": [
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T20:27:30.533556100Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T20:28:18.449013800Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T20:28:39.694065500Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T21:50:08.706414500Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T21:50:14.548024400Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T21:50:25.683879100Z"
    }
  ]
}
```

Tabla 29

Resumen prueba unitaria PU12.

ID DE PRUEBA PU012	
Valor de prueba	**Accion**
start = " 2025-04-02" end = " 2025-04-03 "	Obtener los registros de métricas en un rango de tiempo de la base de datos de InfluxDB
Resultado esperado	**Resultado obtenido**
Figura 29 *Resultado esperado de la prueba PU12.*	**Figura 30** *Resultado obtenido de la prueba PU12.*

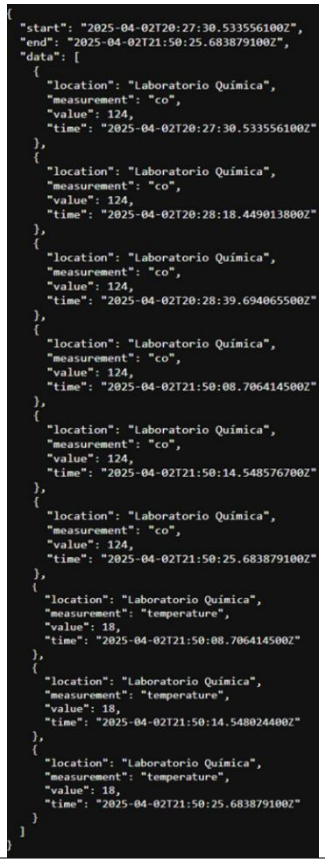
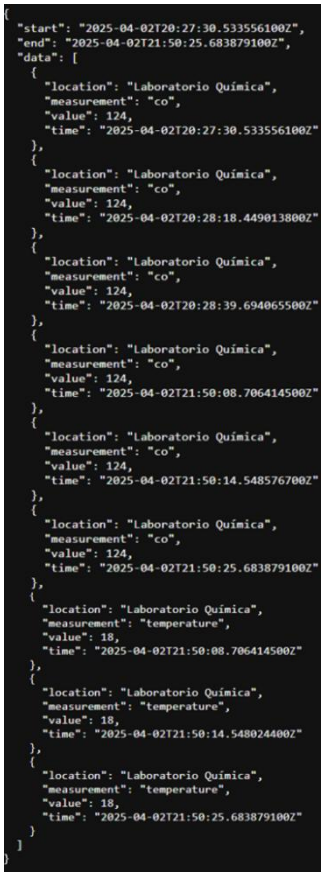

```
{
 "start": "2025-04-02T20:27:30.533556100Z",
 "end": "2025-04-02T21:50:25.683879100Z",
 "data": [
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T20:27:30.533556100Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T20:28:18.449013800Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T20:28:39.694065500Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T21:50:08.706414500Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T21:50:14.548576700Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T21:50:25.683879100Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "temperature",
     "value": 18,
     "time": "2025-04-02T21:50:08.706414500Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "temperature",
     "value": 18,
     "time": "2025-04-02T21:50:14.548024400Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "temperature",
     "value": 18,
     "time": "2025-04-02T21:50:25.683879100Z"
   }
 ]
}
```



```
{
 "start": "2025-04-02T20:27:30.533556100Z",
 "end": "2025-04-02T21:50:25.683879100Z",
 "data": [
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T20:27:30.533556100Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T20:28:18.449013800Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T20:28:39.694065500Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T21:50:08.706414500Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T21:50:14.548576700Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "co",
     "value": 124,
     "time": "2025-04-02T21:50:25.683879100Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "temperature",
     "value": 18,
     "time": "2025-04-02T21:50:08.706414500Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "temperature",
     "value": 18,
     "time": "2025-04-02T21:50:14.548024400Z"
   },
   {
     "location": "Laboratorio Química",
     "measurement": "temperature",
     "value": 18,
     "time": "2025-04-02T21:50:25.683879100Z"
   }
 ]
}
```

Tabla 30

Resumen prueba unitaria PU13.

ID DE PRUEBA PU13	
Valor de prueba	**Accion**
Time = 24h	Obtener los registros de métricas por magnitud de tiempo de la base de datos de InfluxDB
Resultado esperado	**Resultado obtenido**
Figura 31 *Resultado esperado de la prueba PU13.* 	**Figura 32** *Resultado obtenido de la prueba PU13.* 


```
{
  "start": "2025-04-02T20:27:30.533556100Z",
  "end": "2025-04-02T21:50:25.683879100Z",
  "data": [
    {
      "location": "Laboratorio Química",
      "measurement": "co",
      "value": 124,
      "time": "2025-04-02T20:27:30.533556100Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "co",
      "value": 124,
      "time": "2025-04-02T20:28:18.449013800Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "co",
      "value": 124,
      "time": "2025-04-02T20:28:39.694065500Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "co",
      "value": 124,
      "time": "2025-04-02T21:50:08.706414500Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "co",
      "value": 124,
      "time": "2025-04-02T21:50:14.548576700Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "co",
      "value": 124,
      "time": "2025-04-02T21:50:25.683879100Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T21:50:08.706414500Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T21:50:14.548024400Z"
    },
    {
      "location": "Laboratorio Química",
      "measurement": "temperature",
      "value": 18,
      "time": "2025-04-02T21:50:25.683879100Z"
    }
  ]
}
```

Los resultados obtenidos confirman que la solución cumple con los requerimientos funcionales y no funcionales. Se observó la fiabilidad de la implementación de la solución al evidenciarse la coincidencia entre los resultados esperados y obtenidos.

También se debe aclarar que debido a que simultáneamente se desarrollaron otros proyectos relacionados con el módulo de gestión de datos, las pruebas de integración no se llevaron a cabo, por lo tanto, este punto quedo en la línea de trabajo para el futuro.

9. Conclusiones

Al definir las necesidades del módulo para gestionar diversos tipos de datos —logro alcanzado mediante la integración de nuevos protocolos de comunicación y la implementación de distintos modelos de almacenamiento— se alcanzó una mayor flexibilidad en la captura y procesamiento de la información, cumpliendo con los requerimientos funcionales y no funcionales establecidos.

A partir de estos requerimientos, se diseñó una solución orientada a favorecer la evolución del módulo sin comprometer la lógica de negocio central. Para ello, se adoptó la arquitectura Onion, la cual permitió reducir el acoplamiento entre las distintas capas del módulo de datos, facilitando su mantenibilidad y escalabilidad futuras.

Con base en el diseño planteado, se implementó la API del módulo, permitiendo la ejecución de consultas que demostraron la utilidad de emplear bases de datos especializadas, como lo fue InfluxDB para series temporales y MongoDB para datos estructurados y semiestructurados. Esta implementación deja una herramienta de gran utilidad para futuros casos de uso que necesiten la recuperación de datos como, por ejemplo, los valores de una métrica en un rango de tiempo determinado.

Además, la incorporación de la base de datos MinIO permitió almacenar datos no estructurados. MinIO admite la gestión de tipos de archivos, como imágenes, documentos PDF, registros de sensores en formato CSV o archivos de configuración. Esta característica es especialmente útil para dispositivos IoT que puedan transmitir video en tiempo real. Esto refuerza la flexibilidad y adaptabilidad de la solución implementada.

Finalmente, sobre la implementación realizada, se llevó a cabo una evaluación funcional y pruebas unitarias exitosas, que tenían el propósito de verificar que el módulo es capaz de recibir, procesar y almacenar datos de forma eficiente, abarcando datos estructurados, semiestructurados y no estructurados.

10. Trabajo a futuro

A pesar de los avances alcanzados durante el desarrollo del proyecto, se abren múltiples líneas de investigación y mejoras que pueden realizarse en futuros trabajos.

- Integración de nuevos protocolos: Ampliar el soporte a otros protocolos para dispositivos IoT con recursos limitados, pueden mejorar la interoperabilidad entre el módulo de gestión de datos y los dispositivos IoT.
- Ampliar la API: Incorporar un nueva API que permitan aprovechar plenamente las capacidades de consulta de MinIO. Esto permitirá una mejor gestión de diversos tipos de datos no estructurados, como imágenes, registros de sensores y archivos de configuración.
- Análisis de datos: Implementar una herramienta avanzada en la visualización de los datos. Esto permitirá el análisis de los datos almacenados, y ayudará con la toma de decisiones mediante interpretaciones más profundas.
- Pruebas de integración: Realizar pruebas de integración con la plataforma Smart Campus UIS para asegurar la correcta operatividad del módulo de gestión de datos con los demás componentes de la plataforma.
- Implementación en entornos reales: Desplegar y usar el sistema en ambientes reales para obtener retroalimentación directa, identificar posibles mejoras que ayuden a la mejora continua del módulo de gestión de datos.

Referencias Bibliográficas

Abdelilah Bouslama, Yassin Laaziz, Abdelhak Tali, Mohamed Eddabbah. (2019, enero). Abdelmalek Essaâdi University. https://www.researchgate.net/publication/335225229_AWS_and_IoT_for_real-time_remote_medical_monitoring

Análisis profundo de AWS DynamoDB: una base de datos NoSQL para aplicaciones de alto rendimiento. (2023, 2 octubre). Chritopher Adamson. https://medium.com/@christopheradamson253/deep-dive-into-aws-dynamodb-a-nosql-database-for-high-performance-applications-4c80d1410533

Avi. (2020, noviembre 18). 7 potente base de datos de series temporales para la solución de supervisión. Geekflare Spain. https://geekflare.com/es/time-series-database/

Big Data and precision agriculture: a novel spatio-temporal semantic IoT data management framework for improved interoperability. (s/f). https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00729-0

Bróker MQTT: Cómo funciona, opciones populares e inicio rápido (2023, 21 junio). EMQ Technologies. https://emqx.medium.com/mqtt-broker-how-it-works-popular-options-and-quickstart-c4385c3dd6a9

Clean Coder Blog. (2012, 13 agosto). https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Clases de almacenamiento de Amazon S3. (s/f). Amazon.com. Recuperado el 18 de octubre de 2024, de https://aws.amazon.com/es/s3/storage-classes/

Cloud Storage for. (s/f). Firebase. Recuperado el 18 de octubre de 2024, de https://firebase.google.com/docs/storage?hl=es-419

Corradini, F., Fedeli, A., Fornari, F., Polini, A., Re, B., & Ruschioni, L. (2023). X-IoT: a model-driven approach to support IoT application portability across IoT platforms. Computing. https://doi.org/10.1007/s00607-023-01155-z

Introducción a MongoDB: Documentos y NoSQL. (2021, 19 abril). Oscar Fernandez https://oscarfmdc.medium.com/introducci%C3%B3n-a-mongodb-documentos-y-nosql-aprenderbigdata-com-9f756bd13be3

Jiménez, H., Cárcamo, E., & Pedraza, G. (2020). Plataforma Software Extensible para Campus Inteligente Basada en Microservicios. RISTI: Revista Ibérica de Sistemas e Tecnologias de Informação, E38, 270-282.

Kumar, C. (2020, octubre 19). Pruebe MinIO - Almacenamiento de objetos de alto rendimiento autoalojado compatible con S3. Geekflare Spain. https://geekflare.com/es/minio-object-storage/

[MS-RTSP]: Overview. (2024, 23 abril). Microsoft Learn. https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rtsp/d9c6f2be-6778-4531-971d-2d295b568d52

Otar. (2023, septiembre 29). The role of a back-end web developer: Skills and responsibilities. The White Label Agency. https://thewhitelabelagency.com/back-end-developer-role/

Patrón de arquitectura hexagonal - AWS Guía prescriptiva. (s. f.). https://docs.aws.amazon.com/es_es/prescriptive-guidance/latest/cloud-design-patterns/hexagonal-architecture.html

Qué son las bases de datos documentales y ventajas de utilizarlas. (2024, 2 diciembre). Lluís Soler Gomis. https://www.softwaredoit.es/software-gestion-documental/que-son-las-bases-de-datos-documentales.html

¿Qué es Azure Cosmos DB para NoSQL? (2025, 6 abril). Microsoft.com. https://learn.microsoft.com/es-es/azure/cosmos-db/nosql/overview

¿Qué es un broker de mensajería? (2023, 17 Julio). Diego Cortes. https://medium.com/@diego.coder/que-es-un-broker-de-mensajer%C3%ADa-8aa8ab7988e8#:~:text=Un%20broker%20de%20mensajer%C3%ADa%20o,intercambio%20de%20mensajes%20entre%20ellos.

Quincozes, V. E., Quincozes, S. E., Kazienko, J. F., Gama, S., Cheikhrouhou, O., & Koubaa, A. (2024). A survey on IoT application layer protocols, security challenges, and the role of explainable AI in IoT (XAIoT). International Journal of Information Security, 23(3), 1975–2002. https://doi.org/10.1007/s10207-024-00828-w

Simanjuntak, E., & Surantha, N. (2022). Multiple time series database on microservice architecture for IoT-based sleep monitoring system. Journal of Big Data, 9(1). https://doi.org/10.1186/s40537-022-00658-4

The Onion Architecture: part 1. (2018, 4 julio). Programming With Palermo. https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/

Tran, K. T. M., Pham, A. X., Nguyen, N. P., & Dang, P. T. (2024). Analysis and performance comparison of IoT message transfer protocols applying in real photovoltaic system. International Journal of Networked and Distributed Computing, 12(1), 131–143. https://doi.org/10.1007/s44227-024-00021-4

What is Object Storage? Use cases & benefits. (s/f). Google Cloud. Recuperado el 18 de octubre de 2024, de https://cloud.google.com/learn/what-is-object-storage

Ye, F., Liu, Z., Zhu, S., Zhang, P., & Chen, Y. (2020). Research of benchmarking and selection for TSDB. En Algorithms and Architectures for Parallel Processing (pp. 642–655). Springer International Publishing. https://link.springer.com/chapter/10.1007/978-3-030-38961-1_54

Apéndices

Apéndice A. Repositorio de la implementación

Enlace al repositorio de GitHub:

https://github.com/JulianCastillo14/data_microservice.git

Apéndice B. Documentación para configurar el proyecto para utilizarlo en una máquina local

A continuación, se presentan los pasos necesarios para configurar el proyecto en una máquina local.

Requisitos:

Se debe contar con las siguientes herramientas:

- Git para clonar el repositorio del proyecto.
- Docker para gestionar los contenedores que permiten el funcionamiento del proyecto.
- Intellij IDEA para utilizar el código fuente del proyecto.
- Postman para la gestión de las cámaras para envió de video.
- Dispositivo que transmita video por el protocolo RTSP.
- Simulador para el envío de datos, puede ser mocker que hace parte de la plataforma Smart Campus UIS o Node-Red

Pasos:

1. Clonar el repositorio

Abre una terminal para clonar el repositorio que contiene el código fuente desarrollado para el proyecto, para esto ejecuta:

```
git clone https://github.com/JulianCastillo14/data_microservice.git
```

2. Levantar contenedores Docker

Inicia los siguientes servicios, ejecutando los comandos relacionados con cada recurso:

InfluxDB

```
docker run -d --name influxdb -p 8086:8086 -v "$PWD/data:/var/lib/influxdb2" -v "$PWD/config:/etc/influxdb2" -e DOCKER_INFLUXDB_INIT_MODE=setup -e DOCKER_INFLUXDB_INIT_USERNAME=admin -e DOCKER_INFLUXDB_INIT_PASSWORD=admin1234 -e DOCKER_INFLUXDB_INIT_ORG=smart-campus -e DOCKER_INFLUXDB_INIT_BUCKET=messages -e DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=8_kiyghJzhqlZENCLnTbc-90hkNfq1U5DS4pRCdMPQdhLY48LW33hsM73nQl1zs_sLVc5IPvJNWsUHZHimZfJw== influxdb:2.7
```

MongoDB

```
docker run -d --name mongodb -p 27017:27017 -v ./mongodb_data:/data/db -e MONGO_INITDB_ROOT_USERNAME=root -e MONGO_INITDB_ROOT_PASSWORD=password -e MONGO_INITDB_DATABASE=iot mongo
```

MinIO

```
docker run -d -p 9000:9000 -p 9001:9001 --name minio -v "$PWD/mnt/data:/data" -e MINIO_ROOT_USER=admin -e MINIO_ROOT_PASSWORD=password quay.io/minio/minio server /data --console-address ":9001"
```

RabbitMQ

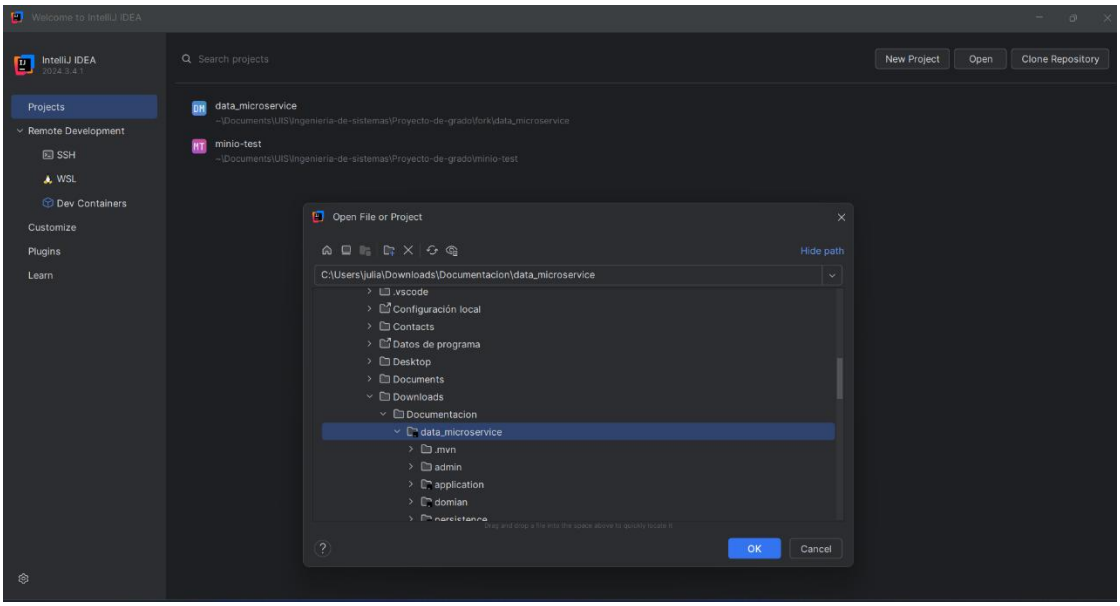
```
docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 -e
RABBITMQ_DEFAULT_USER=admin -e
RABBITMQ_DEFAULT_PASS=password rabbitmq:4.0-management
```

EMQX

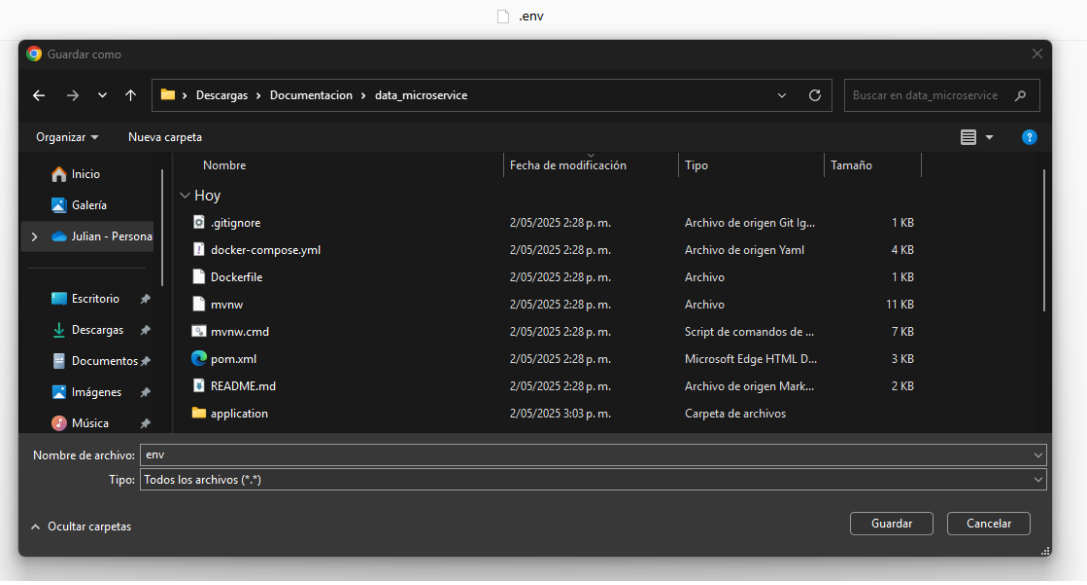
```
docker run -d --name emqx -p 1883:1883 -p 8083:8083 -p 8084:8084 -p 8883:8883 -p
18083:18083 emqx:5.8.6
```

3. Cargar el proyecto con IntelliJ IDEA

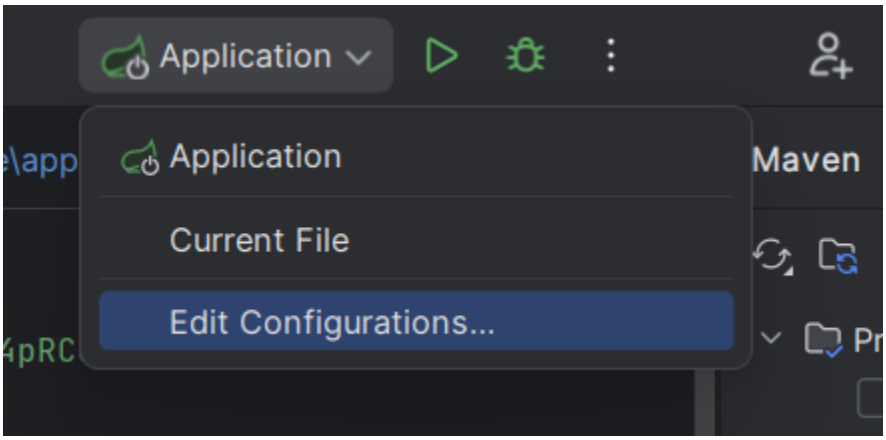
- Abre el IntelliJ IDEA
- Selecciona OPEN
- Abre el directorio del proyecto clonado



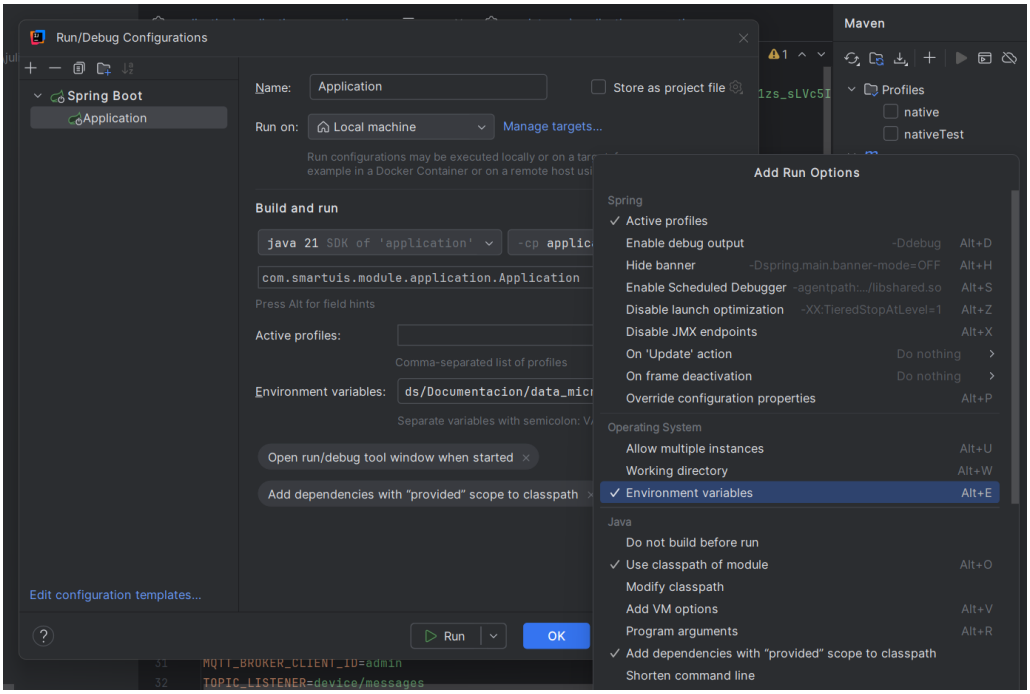
Descarga el archivo -env que contiene las variables de entorno de prueba, el cual se encuentra en el siguiente enlace archivo_env y guardalo en el directorio de data_microservice.



- Carga el archivo .env en el intellijIDEA
- Vea a Run > Edit Configurations

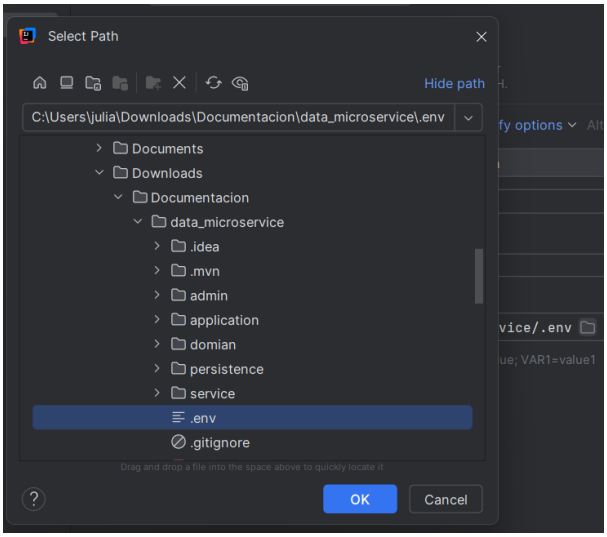


- Haz clic en Modify Options > Environment variables



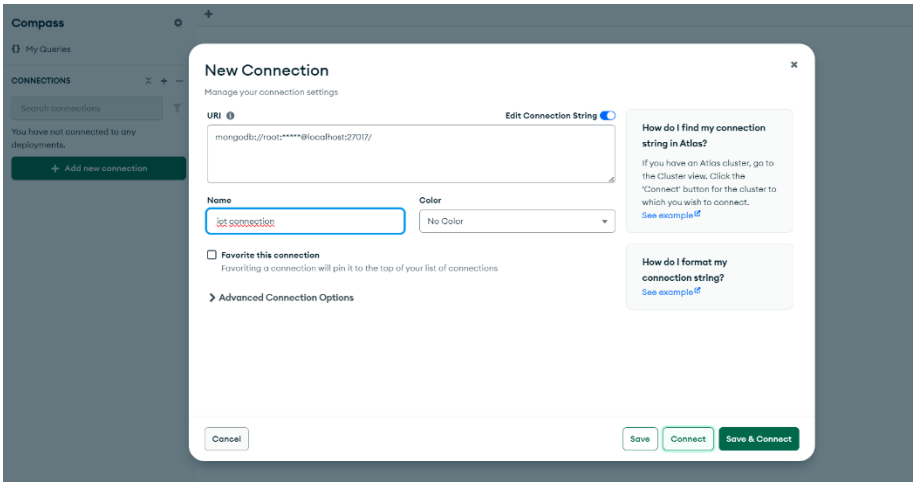
- Selecciona el archivo .env

- Haz clic en Apply y luego en RUN

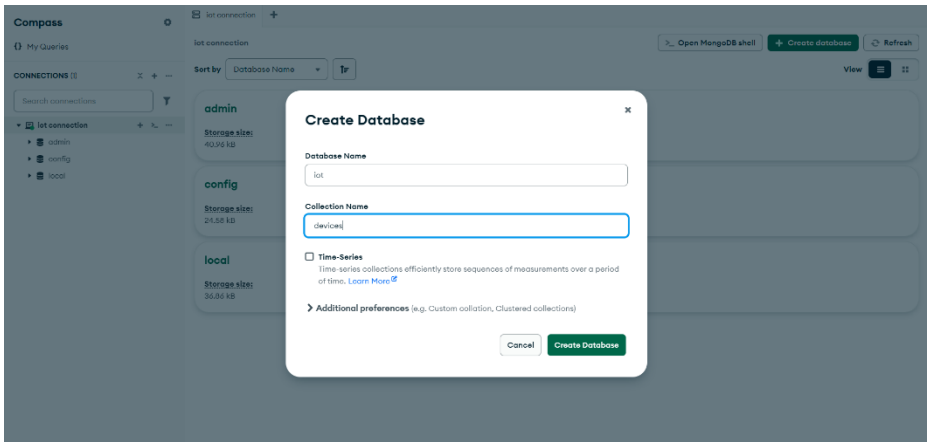


4. Configurar MongoDB

- Abre MomngoDB Compass
- Añade una conexión con la URL: mongodb://root:password@localhost:27017/



- Crea una base de datos con el nombre iot y con una colección que se llame devices

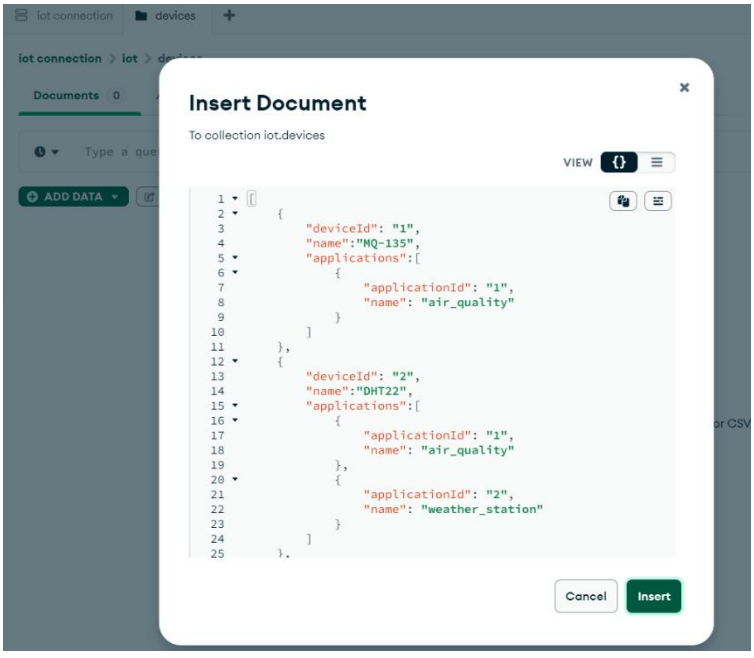


- Descarga el documento que se encuentra en el siguiente enlace devices.json

devices.json

```
[
    {
        "deviceId": "1",
        "name":"MQ-135",
        "applications":[
            {
                "applicationId": "1",
                "name": "air_quality"
            }
        ]
    },
    {
        "deviceId": "2",
        "name":"DHT22",
        "applications":[
            {
                "applicationId": "1",
                "name": "air_quality"
            },
            {
                "applicationId": "2",
                "name": "weather_station"
            }
        ]
    },
```

- Inserta el documento dando clic en ADD DATA > Insert Document

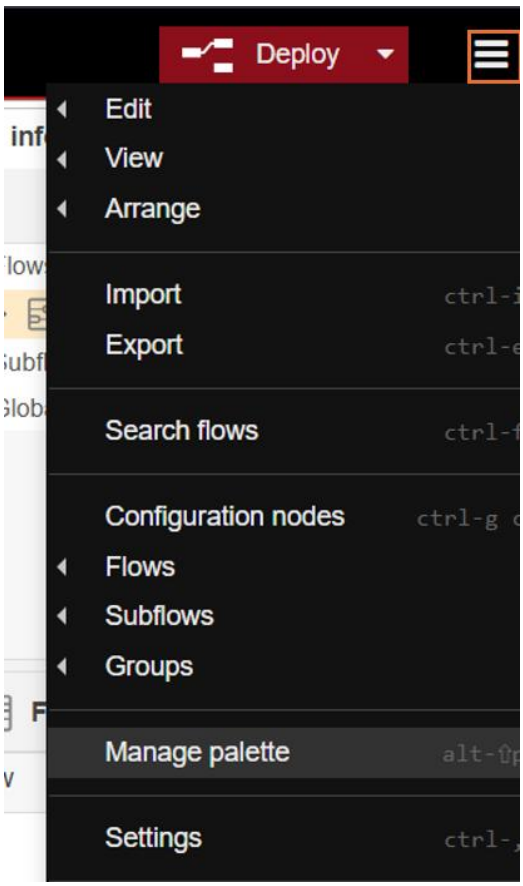


5. Configurar el simulador de datos Node-Red

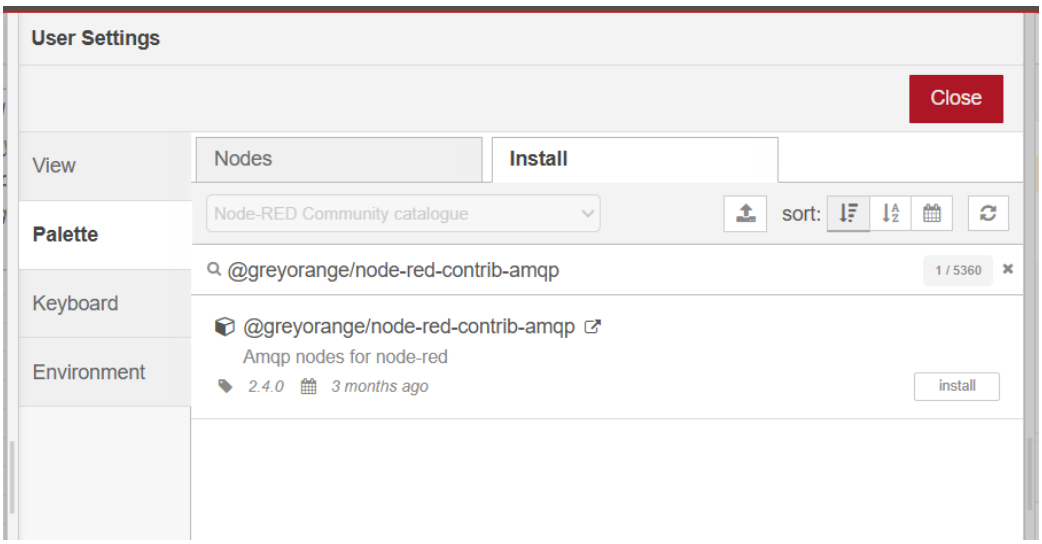
- Levanta un contenedor con el siguiente comando

```
docker run -d -p 1880:1880 -v node_red_data:/data --name mynodered nodered/node-red
```

- Accede al Dashboard con http://localhost:1880/
- Haz clic en las tres líneas
- Ve a Manage palette > Install

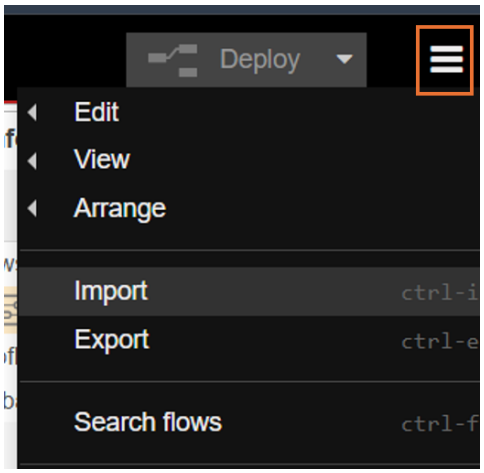


- Busca e instala @greyorange/node-red-contrib-amqp

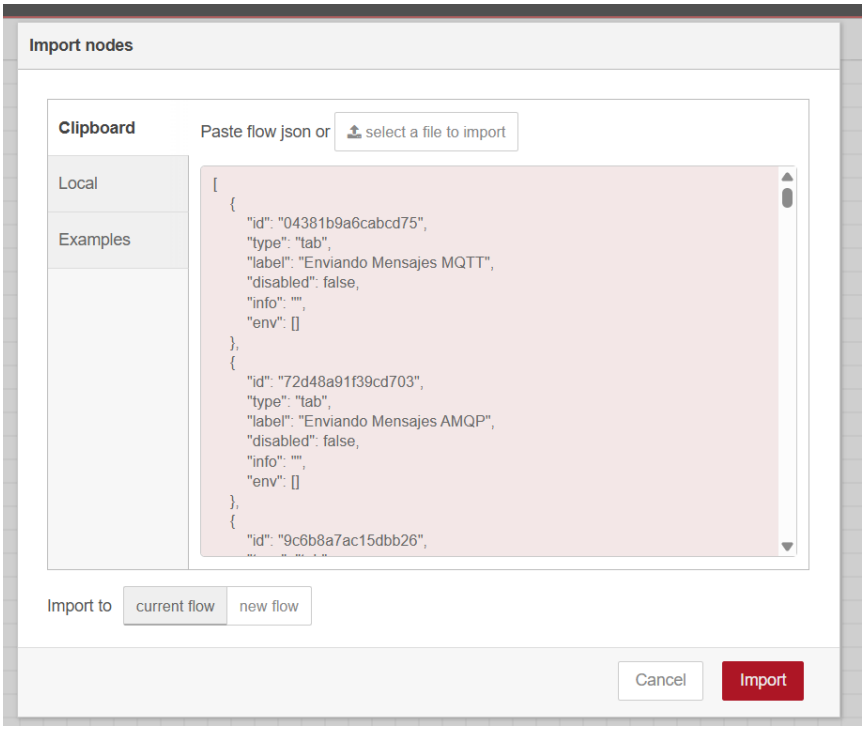


- Descarga la plantilla el siguiente enlace plantilla_node-red

- Haz clic en las tres líneas y luego en Import



- Selecciona el archivo y cárgalo

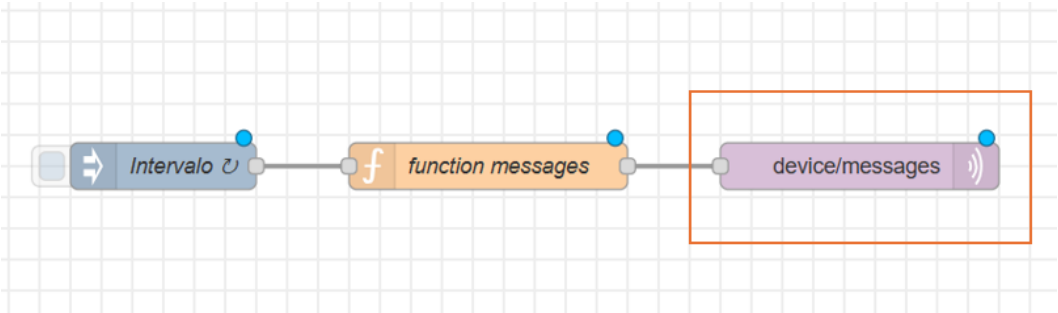


Configurar Ips

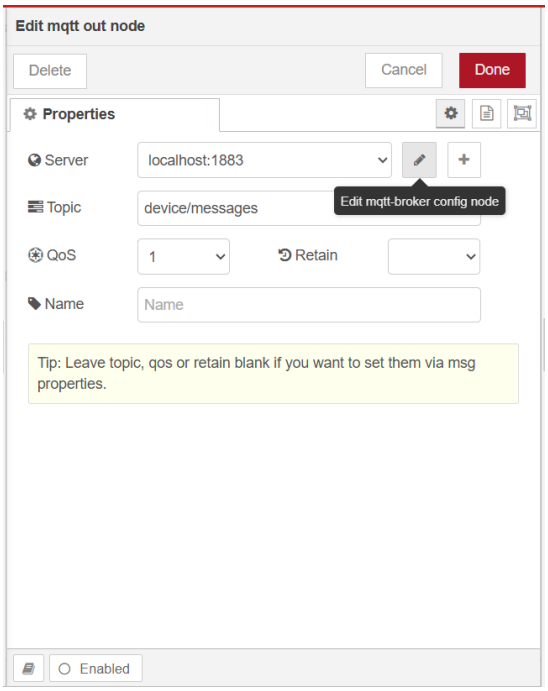
- Obtén la Ip del contenedor de EMQX, ejecuta:

```
docker inspect emqx
```

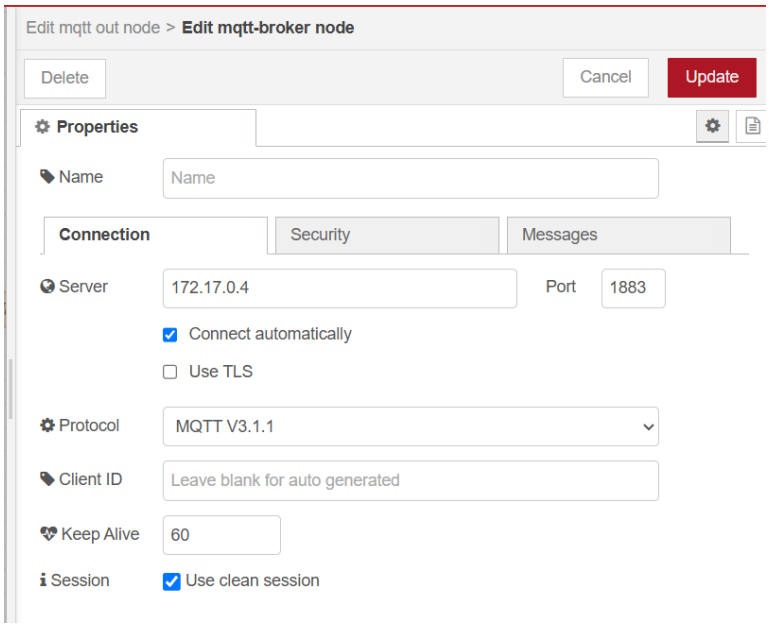
- En el Flow de **Enviando Mensajes MQTT,** edita el nodo *device/messages*



- Haz clic en el símbolo de editar



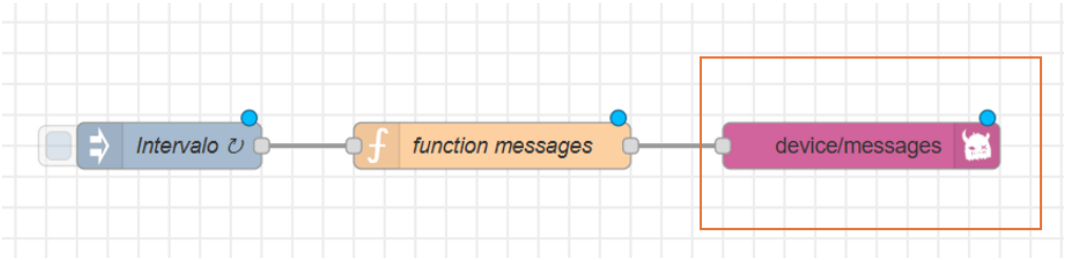
- Actualiza la IP del servidor y haz clic en Update > Done



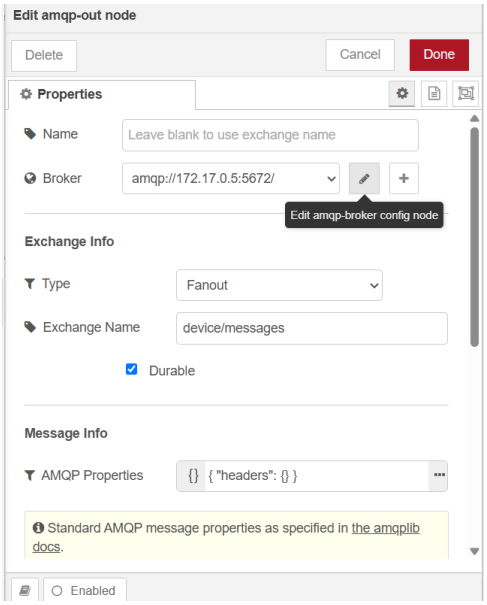
- Obtén la Ip del contenedor de RabbiMQ, ejecuta:

```
docker inspect rabbitmq
```

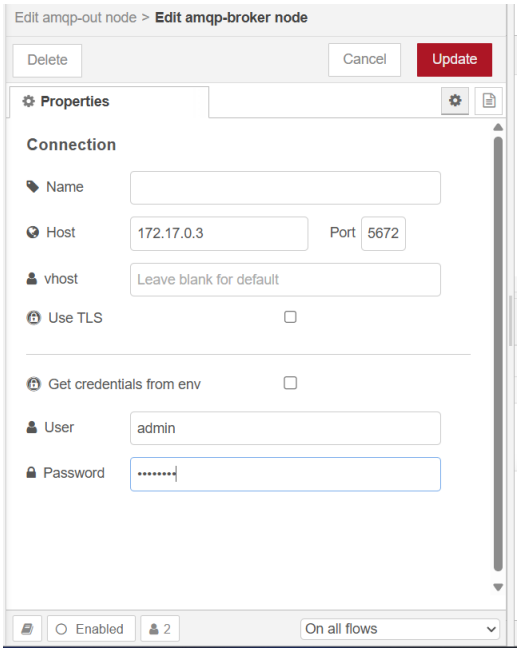
- En el Flow de **Enviando Mensajes AMQP,** edita el nodo *device/messages*



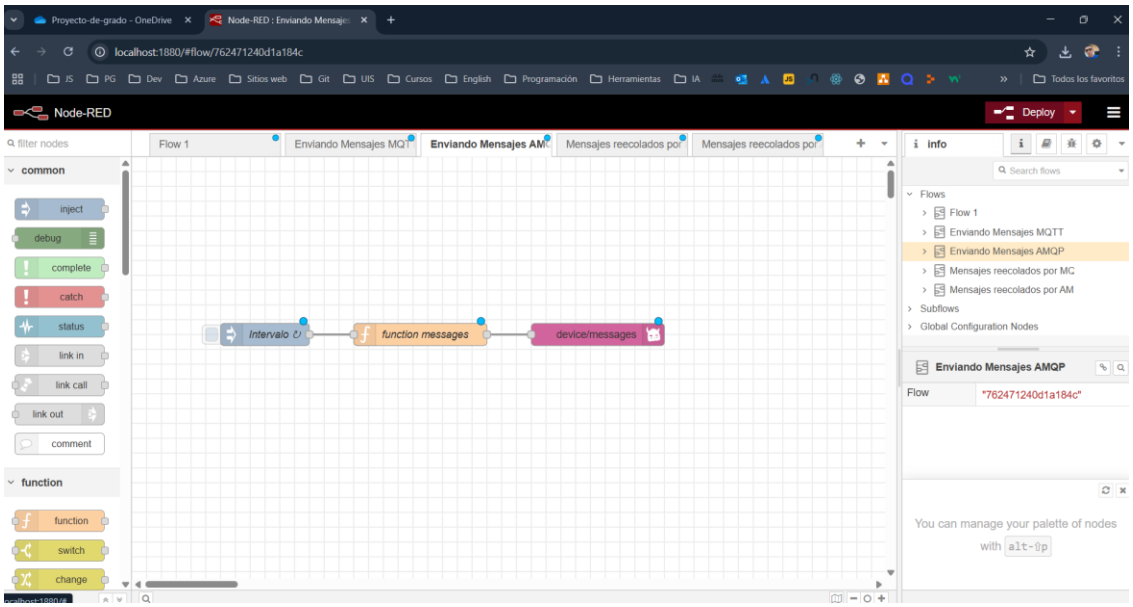
- Haz clic en el símbolo de editar



- Actualiza la IP del HOST y haz clic en Update > Done



- Haz clic en Deploy para iniciar el envió de mensajes

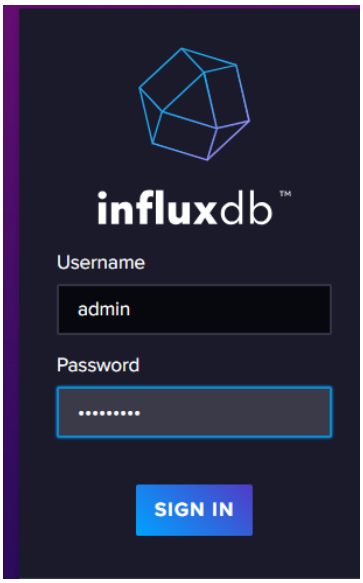


6. Verificación de almacenamiento de los mensajes

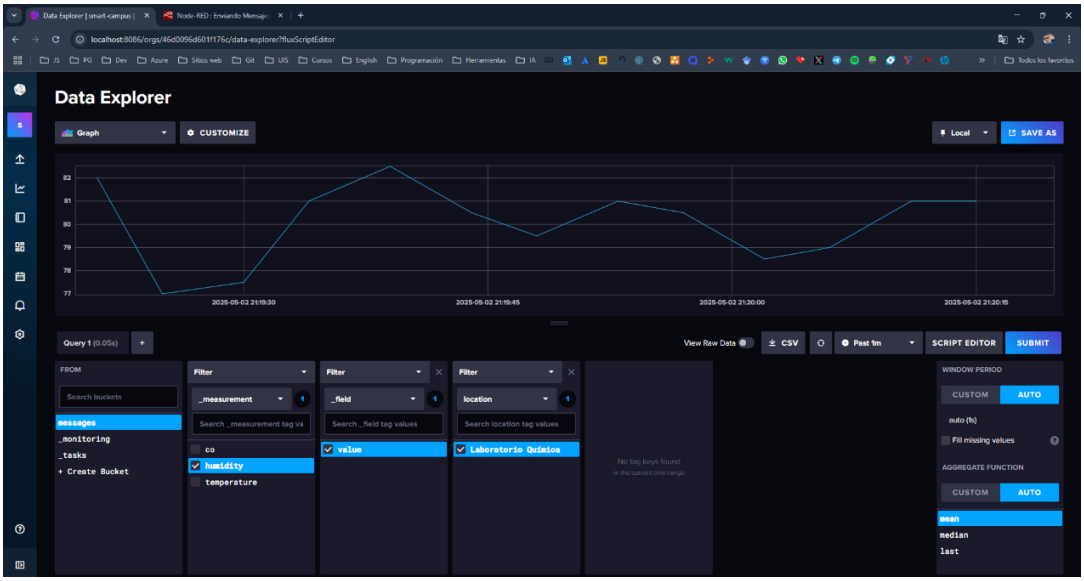
- Accede al Dashboard de InfluxDB con http://localhost:8086/
- Credenciales:

  Usuario: admin

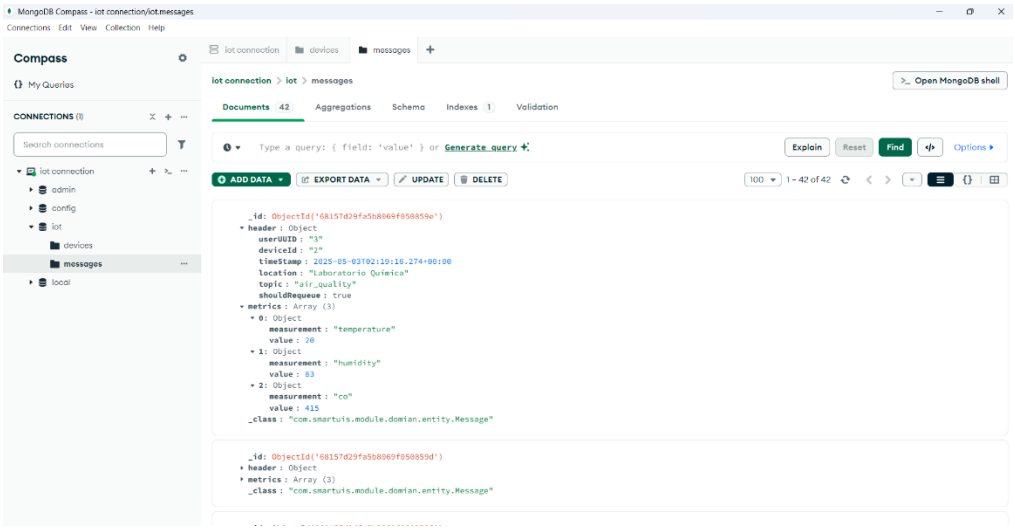
  Contraseña: admin1234



- Consulta el bucket messages haciendo clic en
- Selecciona los parámetros deseados
- Haz clic en Submit para visualizar los mensajes



- También puedes visualizar los mensajes en MongoDB Compass con la conexión que anteriormente se hizo

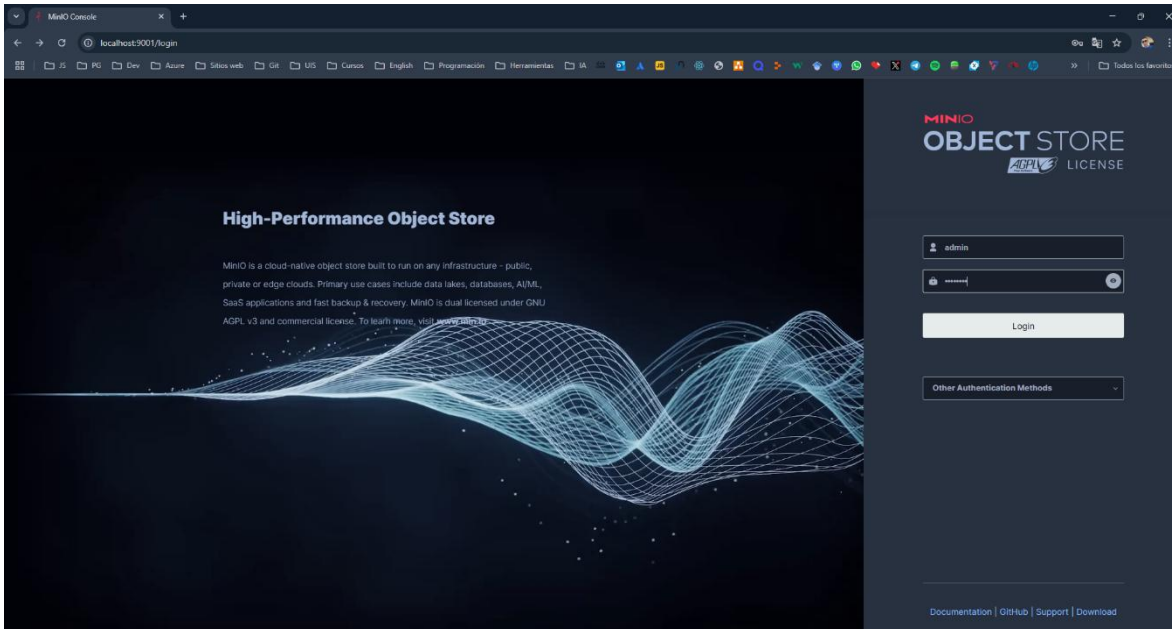


7. Configurar almacenamiento en MinIO

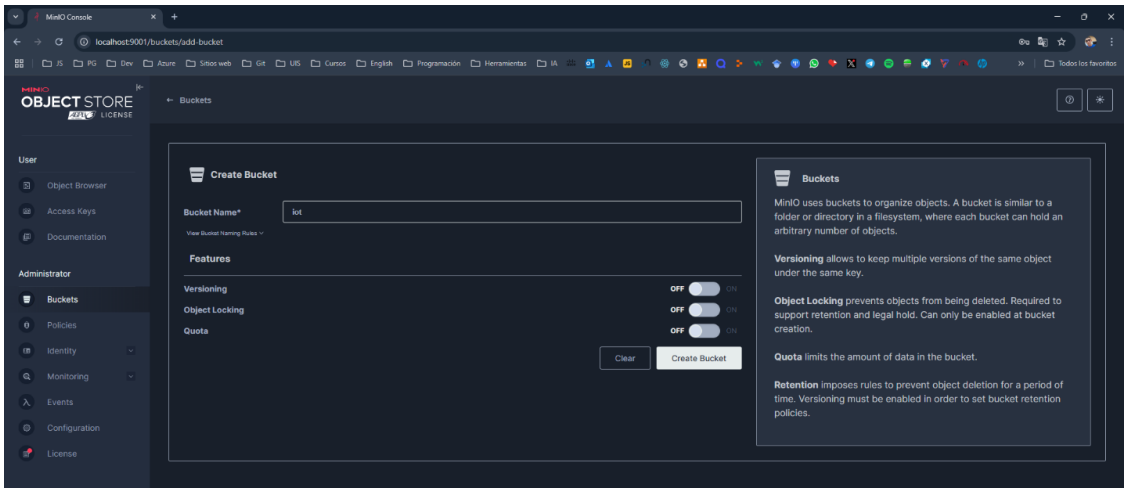
- Accede al Dashboard de MinIo con http://localhost:9001/login
- Credenciales:

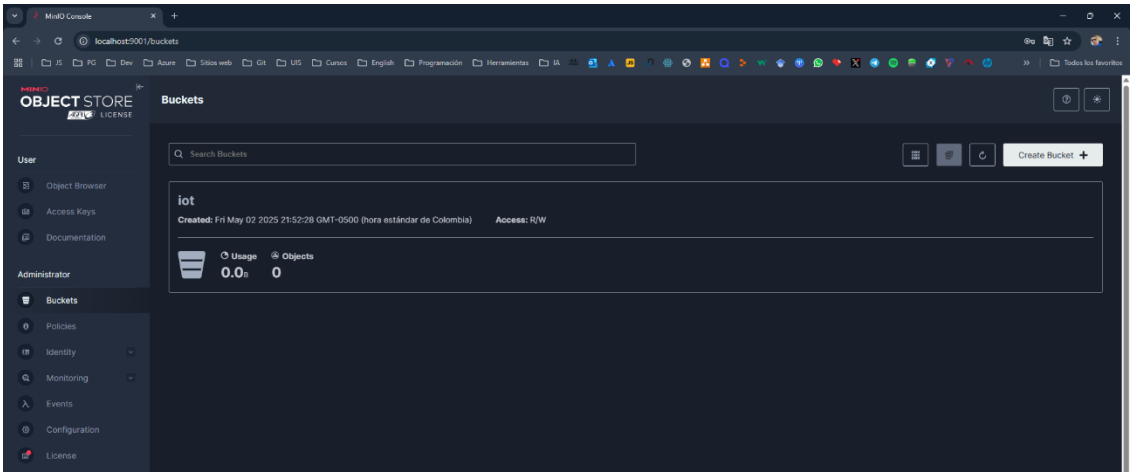
  Usuario: admin

  Contraseña: password

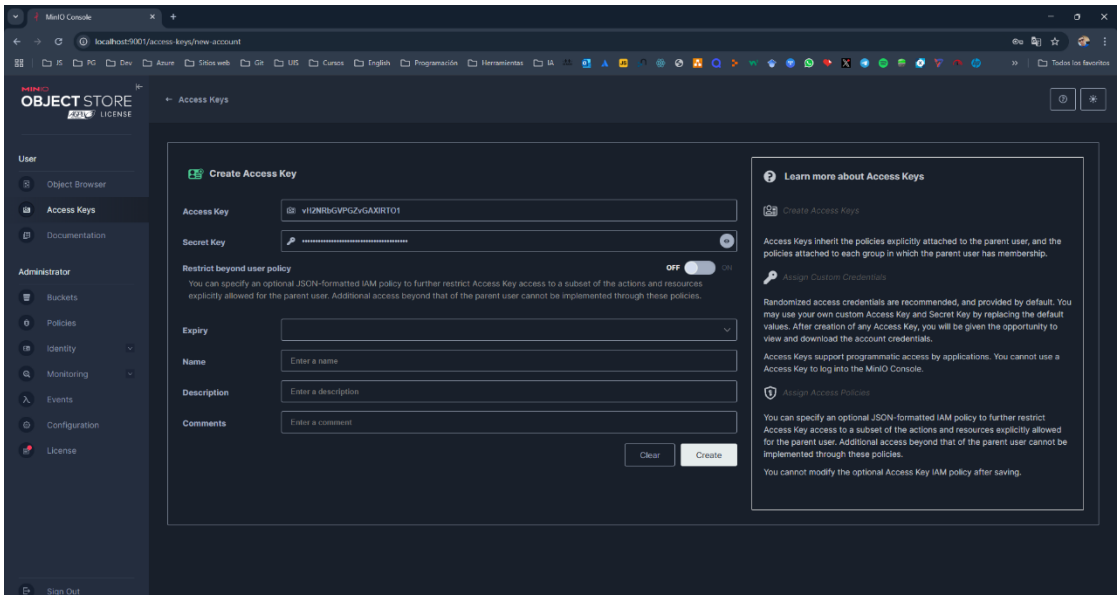


- Haz clic en Buckets
- Crea un bucket llamado iot

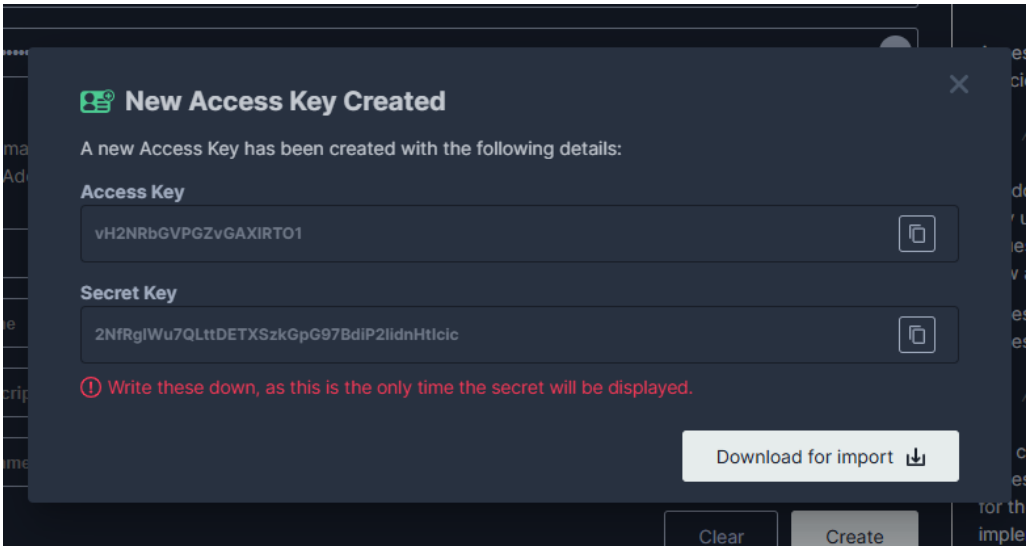




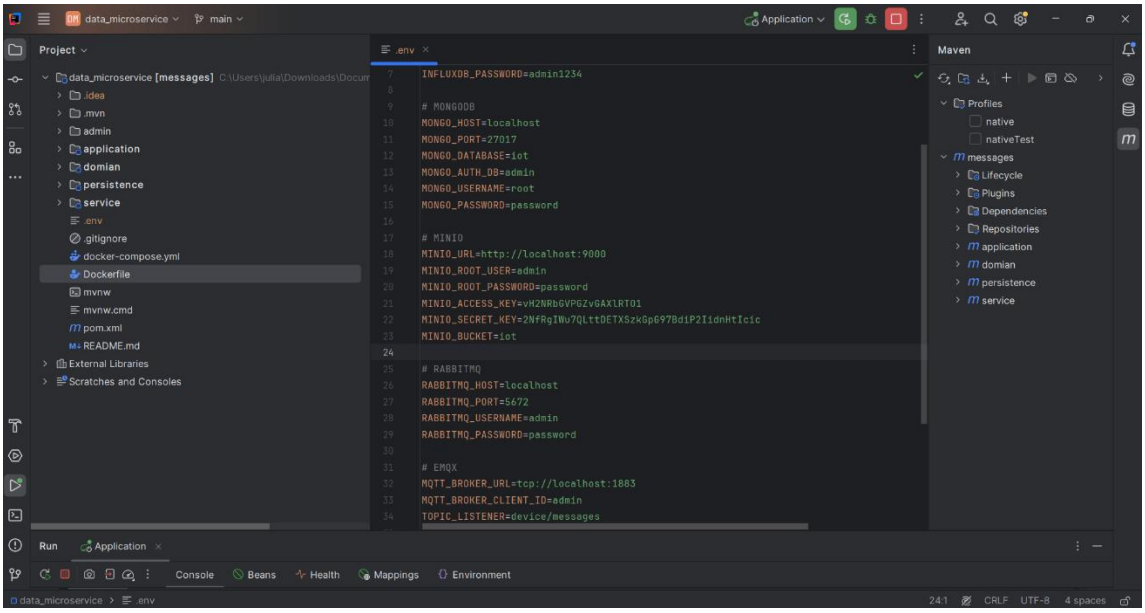
- Ve a AccesKey > Create Access Key
- Haz clic en Create



- Copia las Keys

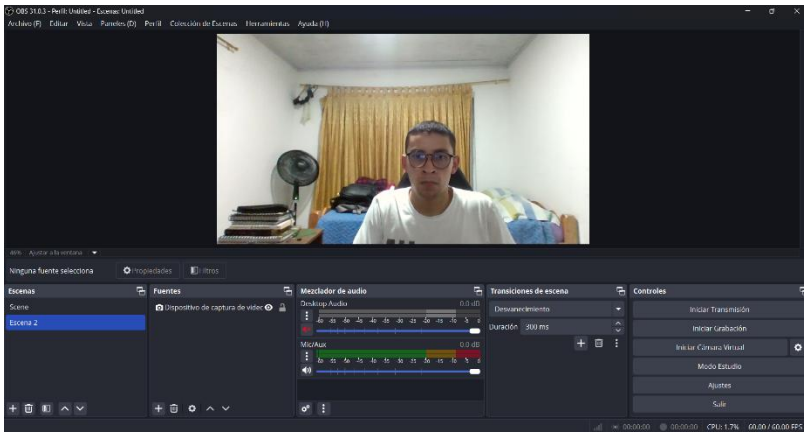


- Pega las Keys en el archivo .env

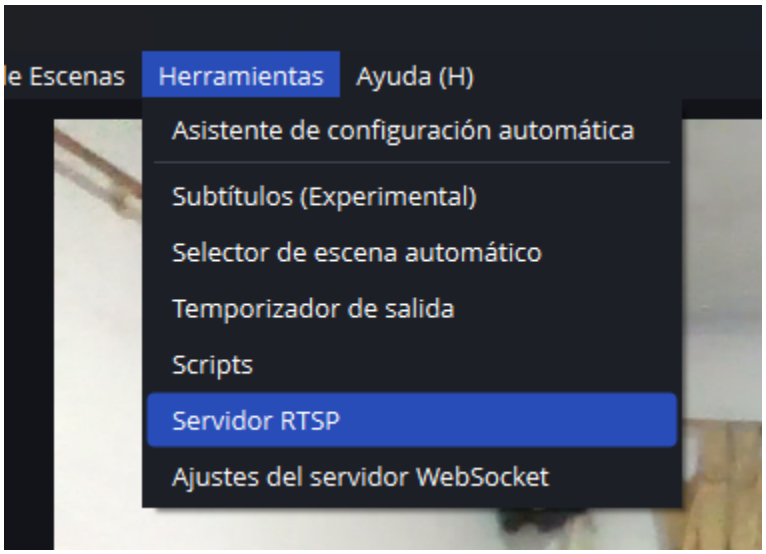


8. Envió de video por RTSP con OBS Studio

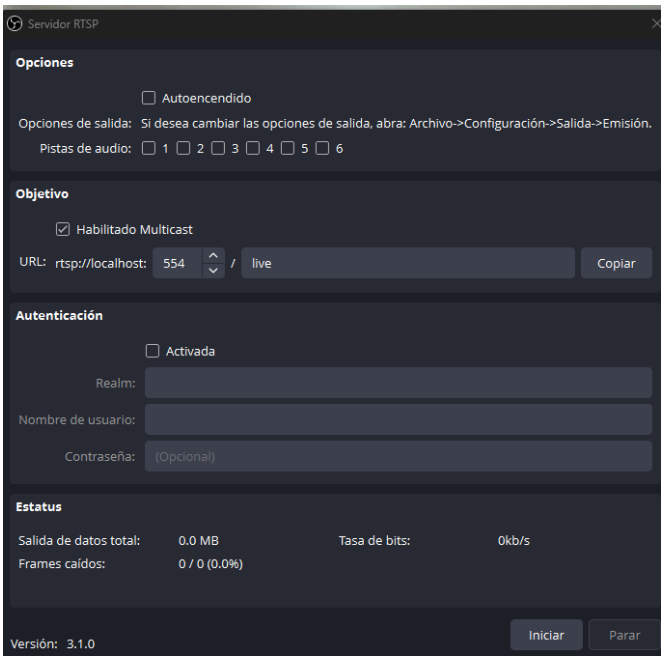
- Abre OBS Studio
- Crea una escena con una fuente (ej. Dispositivo de captura de video)



- Ve a Herramientas > Servidor RTSP



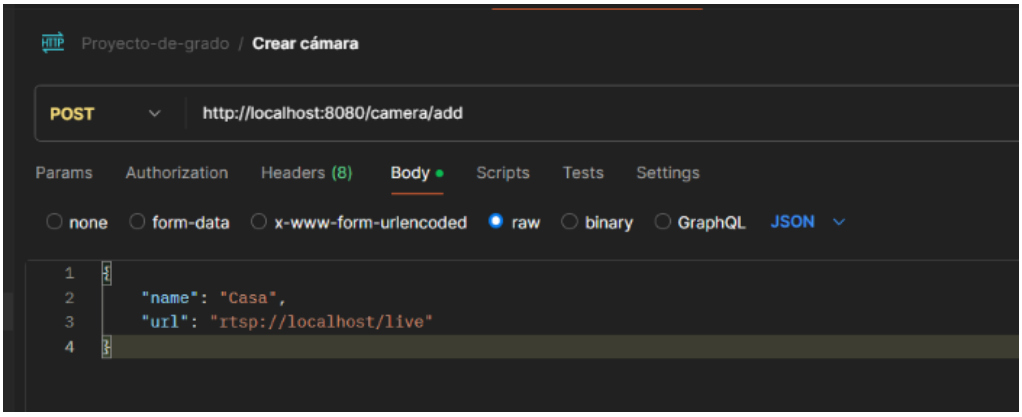
- Copia la URL RTSP rtsp://localhost/live
- Haz clic en Iniciar



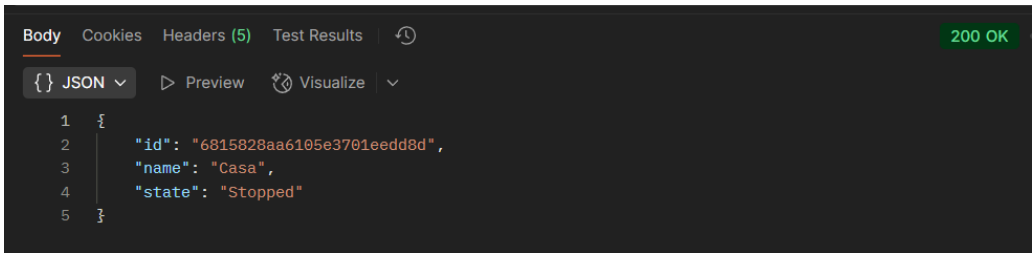
9. Crear cámara con Postman

- Realiza una petición POST a http://localhost:8080/camera
- Con el cuerpo:

```
{
    "name": "NombreCamara",
    "url": "URL_RTSP"
}
```



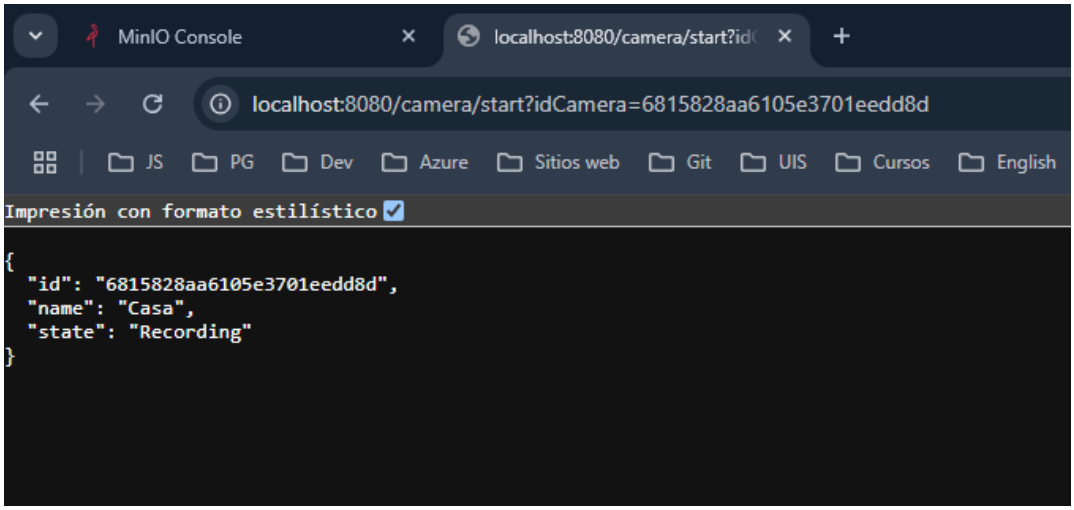
- Copia el id de la cámara



10. Iniciar grabación

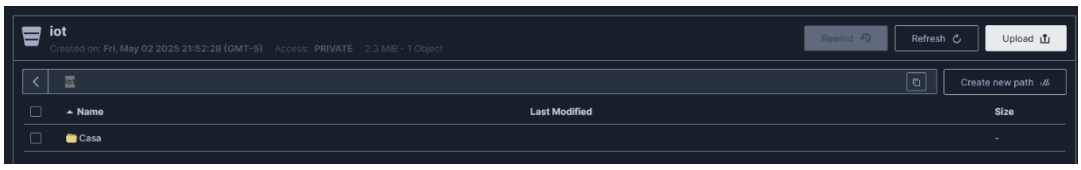
- En el navegador, accede a:

  http://localhost:8080/camera/start?idCamera=<id_de_la_camara>

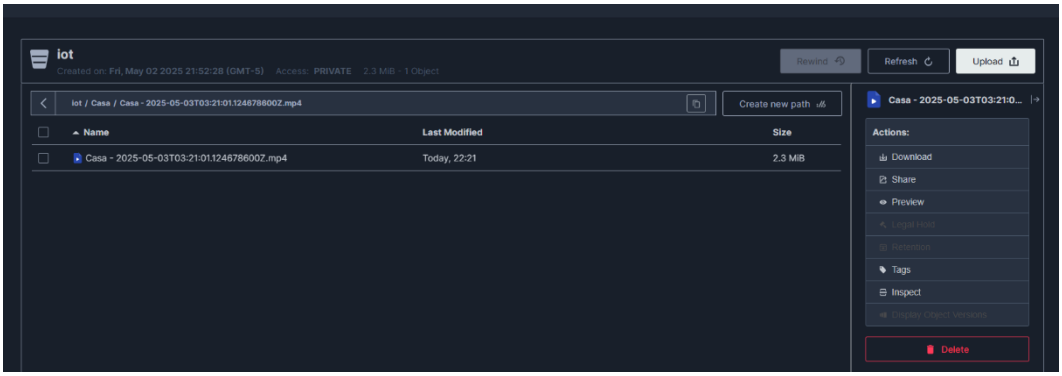


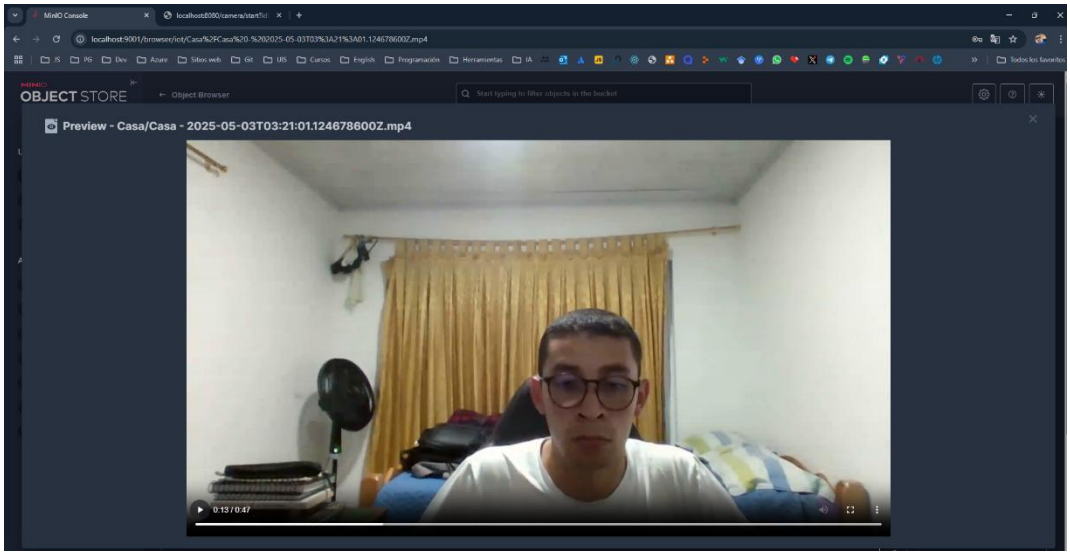
- Espera 1 minuto, que es el tiempo definido con la duración en la que se van a empezar a almacenar los videos transmitidos, definido en el archivo .env en la variable de entorno VIDEO_DURATION_MINUTES

- Verifica en el DashBoard de Minio en Object Browser > Bucket iot que se haya almacenado una carpeta con el nombre de la cámara



- Al ingresar podemos ver el video





11. Funcionalidades adicionales

- Explora la documentación con: http://localhost:8080/swagger-ui.html
- Video demostrativo: https://youtu.be/1CqmGb6i98w