UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y COMUNICACIONES

Tema:

API REST PARA LA TRANSMISIÓN DE INFORMACIÓN Y CONTROL DE REDES DE SENSORES IOT

Trabajo de Titulación Modalidad: Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Electrónica y Comunicaciones.

ÁREA: Electrónica y Comunicaciones

LÍNEA DE INVESTIGACIÓN: Tecnología de la información y sistemas de control.

AUTOR: Giancarlo Culcay Oñate.

TUTOR: Ing. Victor Santiago Manzano Villafuerte, Mg.

Ambato – Ecuador

marzo - 2022

APROBACIÓN DEL TUTOR

En mi calidad de tutor del Trabajo de Investigación, con el tema: API REST PARA LA TRANSMISIÓN DE INFORMACIÓN Y CONTROL DE REDES DE SENSORES IOT, desarrollado bajo la modalidad Proyecto de Investigación por el señor Giancarlo Culcay Oñate, estudiante de la Carrera de Ingeniería en Electrónica y Comunicaciones de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 15 del Reglamento para obtener el Título de Tercer Nivel, de Grado de la Universidad Técnica de Ambato, y el numeral 7.4 del respectivo instructivo.

Ambato, marzo 2022.

Firmado electrónicamente por:
VICTOR SANTIAGO MANZANO VILLAFUERTE

Ing. Victor Santiago Manzano Villafuerte, Mg.

TUTOR

AUTORÍA

El presente Proyecto de Investigación titulado: API REST PARA LA TRANSMISIÓN DE INFORMACIÓN Y CONTROL DE REDES DE SENSORES IOT es absolutamente original, autentico y personal. En tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, marzo 2022.

Giancarlo Culcay Oñate

C.C. 1804933677

AUTOR

APROBACIÓN TRIBUNAL DE GRADO

En calidad de par calificador del Informe Final del Trabajo de Titulación presentado por el señor Giancarlo Culcay Oñate, estudiante de la Carrera de Ingeniería en Electrónica y Comunicaciones, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado API REST PARA LA TRANSMISIÓN DE INFORMACIÓN Y CONTROL DE REDES DE SENSORES IOT, nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 17 del Reglamento para obtener el Título de Tercer Nivel, de Grado de la Universidad Técnica de Ambato, y al numeral 7.6 del respectivo instructivo. Para cuya constancia suscribimos, conjuntamente con la señora Presidenta del Tribunal.

Ambato, marzo 2022.

Firmado electrónicamente por:
ELSA PILAR URRUTIA

Ing. Pilar Urrutia, Mg.
PRESIDENTA DEL TRIBUNAL

Firmado electrónicamente por:
JOSE VICENTE MORALES LOZADA

Ing. PhD. José Vicente Morales Lozada
PROFESOR CALIFICADOR

Firmado electrónicamente por:
EDGAR FREDDY ROBALINO PENA

Ing. Edgar Freddy Robalino Peña, Mg
PROFESOR CALIFICADOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato, para que haga uso de este Trabajo de Titulación como un documento disponible para la lectura, consulta y procesos de investigación.

Cedo los derechos de mi Trabajo de Titulación en favor de la Universidad Técnica de Ambato, con fines de difusión pública. Además, autorizo su reproducción total o parcial dentro de las regulaciones de la institución.

Ambato, marzo 2022.

Giancarlo Culcay Oñate

C.C. 1804933677

AUTOR

DEDICATORIA

El presente proyecto de investigación está dedicado a mis padres por ser un ejemplo de esfuerzo, perseverancia, por su ayuda incondicional y motivarme a cumplir mis metas.

AGRADECIMIENTO

Agradezco a Dios por darme salud, sabiduría e iluminarme para resolver cada problema y reto en mi vida y formación académica.

A mis padres por su constante apoyo, por brindarme todo lo necesario y creer en mí.

A mis docentes universitarios que con sus esfuerzos, enseñanzas y experiencias me han brindado conocimiento y valores.

Al Ingeniero Santiago Manzano por brindarme su apoyo y consejos en el desarrollo de este trabajo.

A mis compañeros y amigos por su apoyo y de los cuales he aprendido mucho.

Giancarlo Culcay Oñate

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN EJECUTIVO

La cantidad de datos que el Internet de las cosas genera está creciendo de forma exponencial, cada vez son más los dispositivos llamados “cosas” que se conectan a internet y proporcionan datos de importancia para diferentes sectores, acceder a estos miles de datos es complicado con las técnicas convencionales, además en estos últimos años por el tema de la pandemia fue necesario que los dispositivos se configuren a distancia y faciliten mucho trabajo.

Este proyecto analiza las tecnologías para el funcionamiento de las redes de sensores, IoT, protocolos de comunicación para el IoT y desarrolla una API REST con Node.js y Express para que los usuarios puedan crear los recursos (nodos, sensores, datos) cada uno relacionado con el otro. Se usa los servicios en la nube para desplegar un contenedor de broker: EMQ X el cual sirve como servidor para las conexiones de los dispositivos IoT, mediante la suscripción y publicación de datos ordenados por su id de sensor. Como la seguridad de estas redes ha dado mucho que hablar, las conexiones MQTT son autenticadas mediante un token de seguridad y no se permiten conexiones anónimas.

Este proyecto usa soluciones serverless, la aplicación API REST se encuentra alojada en el servicio de App Service de Azure y cualquier cliente que desea obtener datos de la API debe tener un usuario registrado y cada vez que realice las solicitudes debe contar con un token de acceso.

Se usa bases de datos de MongoDB por su rendimiento, documentos ligeros y permitir datos sin estructura fija para las características de los sensores y datos que pueden tener diferentes valores, y que cambian continuamente.

Los usuarios pueden realizar las operaciones CRUD con los recursos, y para obtener los datos pueden especificar el límite, offset, fecha de los datos que deseen obtener.

Palabras clave: API REST, nube, datos, JSON, mqtt, control

ABSTRACT

The amount of data that the Internet of things generates is growing exponentially, there are more and more devices called "things" that connect to the Internet and provide important data for different sectors, accessing these thousands of data is complicated with conventional techniques, also in recent years due to the pandemic it was necessary for the devices to be configured remotely and facilitate a lot of work.

This project analyzes the technologies for the operation of sensor networks, IoT, communication protocols for the IoT and develops a REST API with Node.js and Express so that users can create the resources (nodes, sensors, data) each related to the other. Cloud services are used to deploy a broker container: EMQ X which serves as a server for IoT device connections, by subscribing and publishing data ordered by their sensor id. As the security of these networks has been much talked about, MQTT connections are authenticated using a security token and anonymous connections are not allowed.

This project uses serverless solutions, the API REST application is hosted in the Azure App Service and any client that wishes to obtain data from the API must have a registered user and each time they make the requests they must have an access token.

MongoDB databases are used for their performance, lightweight documents and allow data without fixed structure for the characteristics of sensors and data that can have different values, and that changes continuously.

Users can perform CRUD operations with the resources, and to get the data they can specify the limit, offset, date of the data they want to get.

Keywords: REST API, cloud, data, JSON, mqtt, control

CAPITULO I

MARCO TEÓRICO

Tema de Investigación

"API REST PARA LA TRANSMISIÓN DE INFORMACIÓN Y CONTROL DE REDES DE SENSORES IOT"

1.2 Antecedentes investigativos

De acuerdo con la investigación realizada en los repositorios de las diferentes Universidades Nacionales e Internacionales, así como en artículos científicos publicados en diferentes revistas, se ha encontrado proyectos de investigación relacionados con servicios API REST, manejo de redes de sensores IoT, las cuales se describen a continuación:

En el año 2017, en Loja-Ecuador, el Ing. Manuel Fernando Quiñones Cuenca en su trabajo de titulación de maestría: **"SISTEMA DE MONITOREO DE VARIABLES MEDIO AMBIENTALES USANDO UNA RED DE SENSORES INALÁMBRICOS Y PLATAFORMAS DE INTERNET DE LAS COSAS"** propone un sistema para la recolección de datos meteorológicos usando una Red de Sensores Inalámbricos (RSI), capaz de transmitir los datos en tiempo real. Para la viabilidad del diseño e implementación de prototipos propone la construcción de dos sistemas basados en DigiMesh y Wi-Fi, realiza un nodo central conformado por dos módulos de comunicación, el primero interactúa con los nodos sensores para la entrega de los valores recolectados y el segundo retransmite la información obtenida de los nodos sensores a una plataforma IoT en donde gestiona y visualiza los datos obtenidos por los nodos. Usa los protocolos HTTP, MQTT; para el almacenamiento de los datos construye una trama JSON que envía a las plataformas IoT: Ubidots, Phant y

ThingSpeak, en las cuales evalúa su prototipo, logrando almacenar los datos meteorológicos de forma continua. [1]

En el año 2018, en Lima (Perú), Longa Chevarría Bryan Henry en su trabajo de titulación: **"REST API FOR MANAGEMENT OF ELECTRONIC DEVICES"** de la Universidad Peruana de Ciencias Aplicadas, diseña el proyecto "RAPIMED" que plantea la creación de una interface en la web que a través de servicios RESTful el cual permite que cualquier aplicación capaz de solicitarlos pueda comunicarse y controlar los diferentes dispositivos electrónicos de bajo nivel, también contempla conexión al servidor y la comunicación con el cliente final. Para esto usa el microcontrolador Arduino que se conecta a un servidor en la nube mediante conexión serial al PC, usa un sensor HC-SR04 para medir distancias, realiza una conexión especializada que hace posible controlarlo, para el desarrollo de la API REST usa ASP.Net y Swagger para la creación del catálogo de métodos de la API [2]

En el año 2018 Gabriele Bianchini en su trabajo de fin de grado: **"DESARROLLO DE UN API REST PARA TRANSMISIÓN DE DATOS DE SENSORES GPS"** desarrolla un API REST capaz de almacenar y servir los datos devueltos por un GPS a través de una serie de antenas de campo. El API hace de cliente/servidor obteniendo la información del GPS y almacenándola en una base de datos no relacional, para después mostrarla a través de una interfaz de usuario, esto le facilita el tratamiento de los datos para un futuro. Para el desarrollo, usa el Stack MEAN, realiza el despliegue con Docker y usa el lenguaje Javascript tanto en la parte de cliente (Angular) como el servidor (Node.js). Al final obtiene una API capaz de almacenar una gran cantidad de información la cual es accedida de forma rápida y sencilla, además que permite la visualización por el usuario. Consigue desarrollar un software escalable, el cual necesita más servidores para desplegar más nodos. [3]

En el año 2019, en Santiago de Cali se registra un proyecto de investigación con el tema: **"ARQUITECTURA REST PARA LA PLATAFORMA UAO-IOT"**, presentado por los señores Javier Benavidez y Jesús García de la Universidad

Autónoma de Occidente, en el cual realizan el diseño e implementación del protocolo REST para su plataforma UAO-IoT que ya contaba únicamente con el protocolo MQTT, implementan la arquitectura de 3 capas: adquisición, procesamiento y aplicación; realizan la API REST con la arquitectura MVC (Modelo-Vista-Controlador), esta API gestiona las solicitudes como el almacenamiento y recuperación de información de la base de datos, realizadas tanto por la capa de aplicación y adquisición. La capa de adquisición consiste únicamente en realizar las conexiones a la plataforma para el envío de datos desde una Raspberry Pi con su respectiva librería HTTP. Para la capa de procesamiento selecciona Node.js y Express, MongoDB para la gestión de base de datos. Para la capa de aplicación usa el framework Angular que ofrece arquitectura MVC, velocidad y rendimiento. El desarrollo del proyecto permite a los usuarios de la plataforma, utilizar las funciones de REST para la información de los sensores, permitiendo construir proyectos de IoT, aumentando la cantidad de dispositivos, velocidad y gestión de la información. [4]

En el año 2019 en la conferencia "International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)" los señores Hittu Garg y Mayank Dave publican el artículo **"SECURING IOT DEVICES AND SECURELYCONNECTING THE DOTS USING REST API AND MIDDLEWARE"** en el cual presentan el rol de la API REST, también destacan el concepto de middleware que conecta estos dispositivos y la nube para el tema de seguridad y privacidad de los datos, Introducen un sistema de IoT seguro que no permite infiltración de atacantes en la red a través de dispositivos IoT y también para proteger los datos en tránsito desde los dispositivos de IoT a la nube. En el modelo propuesto, El middleware se utiliza principalmente para exponer los datos del dispositivo a través de REST y para ocultar detalles y actuar como una interfaz para el usuario para interactuar con los datos del sensor. Los dispositivos de IoT están aislados y no tienen interacción con el mundo exterior. Están conectados a una puerta de enlace y esta puerta de enlace actúan como una interfaz para el Internet para los dispositivos IoT. [5]

En el año 2019, en la conferencia “8th International Conference On Software Process Improvement (CIMPS)” el Dr. Rodolfo Omar Domínguez García, Mtro. José Roberto Lomeli Huerta, Mtra. Miriam González Dueñas publican el artículo **“API LIKE SERVICES FOR MULTIPLE SYSTEMS ORIENTED TO IOT”** presenta el desarrollo de una API-REST orientada al IoT, que engloba múltiples servicios a través de una comunicación multiplataforma. El objetivo de la API-REST del CUValles es permitir una gestión eficiente de los datos que son generados a partir de diversas aplicaciones como controles de acceso, sistemas de detección, sistemas de soporte a la toma de decisiones, entre otras. La implementación desarrollada brinda un proceso estandarizado de acceso a los diferentes sistemas que interactúan en el CUValles. [6]

Debido a que las aplicaciones están desarrolladas bajo diferentes lenguajes y diferentes sistemas de bases de datos, lo que eleva la complejidad de la integración de la información, proponen la solución con el uso de una API REST. La construcción del ASMS-IoT se basó en las buenas prácticas de una REST API, las cuales incluyen: recursos (sustantivos), los métodos (GET, PUT, POST) y las representaciones (XML, JSON). [6]

1.2.1 Contextualización del Problema

Grandes compañías tienen miles de dispositivos IoT que no están directamente relacionados con el negocio, pero que se conectan a su red todos los días, esta situación expone un reto para las organizaciones empresariales desde el punto de vista de la seguridad y la administración. Esto se debe, en parte, a que la cantidad de dispositivos conectados a internet, desde sensores, cámaras y máquinas, crece a un ritmo exponencial (se calcula que habrá unos 41.600 millones dispositivos IoT en 2025). Mientras mayor sea número de “cosas” conectadas más datos generados que hay que administrar. [7]

La interconexión de dispositivos bajo el paradigma de internet de las cosas (IoT) presenta retos debido a la cantidad de dispositivos IoT con múltiples plataformas de hardware y lenguajes de programación. La heterogeneidad de los protocolos requiere el desarrollo de soluciones como las capas API-REST, que facilitan la comunicación

multiplataforma, su flexibilidad las vuelve una opción imprescindible para el desarrollo de sistemas de complejidad elevada por el número de elementos que interactúan en él. [6]

Muchas compañías ya se han sumado a la tendencia del IoT, volviendo la infraestructura más compleja. El cual debe tener control y donde la seguridad es un reto. Las instituciones públicas como el Gobierno también hacen uso del IoT para lograr implementar numerosos servicios, como el alumbrado, seguridad, tráfico, vigilancia, etc. [8]

Se necesita el desarrollo de API REST que ofrecen grandes ventajas para el desarrollo, aumenta la capacidad de conectar aplicaciones a dispositivos, ya sea individualmente o en conjunto, permite independización del cliente y el servidor, la cual mejora la portabilidad de la interfaz, la escalabilidad de los proyectos, existe una mayor posibilidad de incrementar nuevas características, facilitar la exposición del sistema a diferentes plataformas (los usuarios pueden ser apps, otros dispositivos IoT, navegadores, programas, entre otros). [6] [4]

1.2.2 Fundamentación teórica

1.2.2.1 Internet de las cosas (IoT)

Internet de la Cosas, es un concepto que está revolucionando la Industria, con una visión de un mundo en el que miles de millones de objetos con inteligencia integrada, medios de comunicación y capacidades de detección y actuación se conectan a través de redes IP [9], es decir la interconexión de objetos y dispositivos a través del internet, como un pequeño sensor ubicado en una industria enviando datos importantes de los procesos, hasta un dispositivo que envía la posición del usuario, todo tiende a estar conectado con el fin de obtener información, para almacenar, monitorear y tomar acciones automatizadas en tiempo real. [10]

Los dispositivos llamados “cosas” son objetos cotidianos conectados a través de Internet, permiten la interacción entre el mundo físico y el digital, el mundo digital interactúa con el mundo físico a través de sensores y actuadores, estos sensores envían y reciben datos del entorno sin depender de los humanos. El procesamiento de datos puede tener lugar en el borde de la red o en un servidor remoto o en la nube. Las

capacidades de almacenamiento y procesamiento de un objeto de IoT están restringidos por los recursos disponibles, limitados al tamaño, energía, potencia y capacidad computacional. [5]

El almacenamiento, procesamiento y análisis de datos son requisitos fundamentales, necesarios para enriquecer los datos sin procesar de IoT y transformarlos en información útil. Luego, la información útil se difundirá a los dispositivos pertinentes y a los usuarios interesados o se almacenará para su posterior procesamiento y acceso. [9]

Empresas de todo el mundo están invirtiendo miles de millones en IoT para resolver problemas industriales (IIoT). El IIoT se refiere a objetos industriales equipados con sensores, que se comunican automáticamente a través de una red, sin ninguna interacción persona a persona o persona a computadora, para intercambiar información y tomar decisiones inteligentes con el apoyo de analítica avanzada (figura 1). [11]

Figura 1 Aplicaciones IoT [11]

La arquitectura de IoT consta de tres capas, como cosas, puerta de enlace y nube, que comprenden sensores, dispositivos y objetos. La Figura 2 muestra la arquitectura de IoT. La puerta de enlace (Gateway) consta de protocolos de IoT.

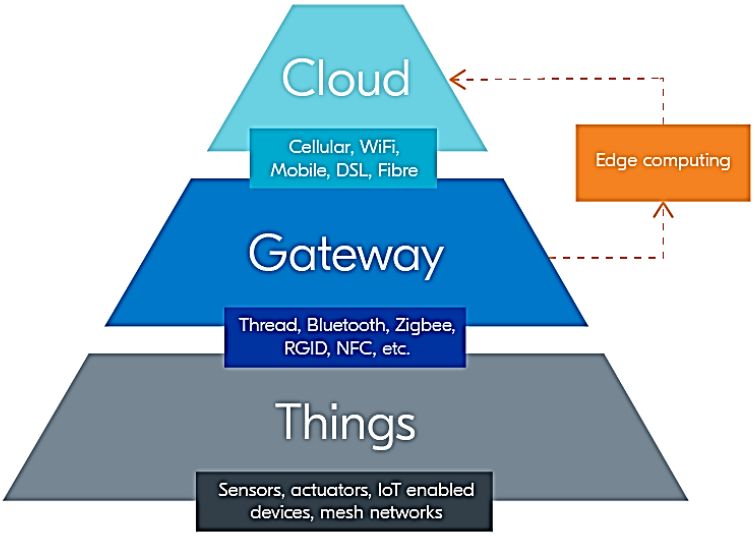


Figura 2 Arquitectura IoT [12]

1.2.2.2 Redes de Sensores Inalámbricos

Las Redes Inalámbricas de Sensores (WSN, por sus siglas en inglés), es un conjunto de dispositivos llamados nodos interconectados entre sí que monitorean variables físicas para enviar información del entorno utilizando tecnologías y protocolos de tipo inalámbrico hasta una estación base o servidor. En las WSN cada nodo se considera como un dispositivo autónomo que puede tener como elementos una fuente de alimentación, unidad de procesamiento, memoria, sensores y dispositivo de comunicación inalámbrica. [13]

En la última década se han utilizado las WSN para diferentes aplicaciones, por ejemplo: redes para el control del tráfico vehicular, rastreo del desplazamiento de personas, gestión de la salud, monitoreo de áreas urbanas, monitoreo ambiental y del clima, sector militar, fabricación, agricultura, construcción, transporte, etc. [13] , [14]

Una de las principales características de las WSN es su limitada capacidad en relación con el ancho de banda, procesamiento de datos, memoria y baterías. Respecto al procesamiento de los datos, en general, las WSN se diseñan con un enfoque centralizado, es decir, los nodos actúan como simples recolectores de información y el

procesamiento de la información se realiza en otro equipo de cómputo donde se reciben y almacenan los datos.

Una de las propiedades más atrayentes de las redes de sensores es su bajo consumo ya que provee a los dispositivos de una enorme autonomía. Esto facilita que los sensores se ubiquen en localizaciones poco accesibles o incluso integrados dentro de construcciones. No obstante, una de los mayores obstáculos de las redes de sensores es que son difíciles de desplegar y gestionar. [15]

Sensor

Un sensor es un dispositivo de hardware que detecta o mide magnitudes físicas, por ejemplo, temperatura, sonido, presión, luz, etc. Y las convierte a señales eléctricas.
En la figura 3 se puede observar el diagrama de bloques de un dispositivo sensor.

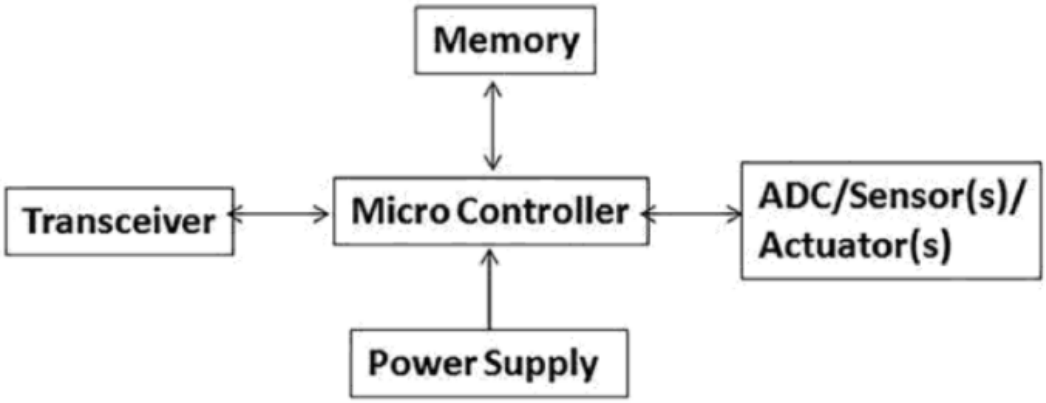


Figura 3 Diagrama de bloques de un dispositivo sensor [14]

Los nodos de la red de sensores inalámbricos no disponen de una fuente de alimentación interrumpida, por lo cual se alimentan de batería externas, por ello se debe seleccionar un estándar de comunicaciones inalámbrico que asegure un alcance considerable en la red y a la vez tenga un consumo bajo de energía.

Estándares de comunicación

Los estándares de comunicaciones son la primera elección que se debe especificar cuándo se va a diseñar una red inalámbrica, debido a que la fidelidad de la comunicación dependerá del estándar que se vaya a utilizar.

IEEE 802.11 es un conjunto de especificaciones PHY / MAC para implementar redes de área local inalámbricas (WLAN) en varias bandas de frecuencia, incluidas las bandas de 900 MHz y 2.4, 3.6, 5 y 60 GHz. La versión básica del estándar fue lanzada en 1997 y ha tenido numerosas modificaciones posteriores. El estándar y sus enmiendas proporcionan la base para los productos de redes inalámbricas que utilizan la marca Wi-Fi. [9]

IEEE 802.15 definió la capa física (PHY) y la subcapa de acceso medio (MAC) para baja complejidad, conectividad WPAN de bajo consumo de energía y baja tasa de bits. El estándar IEEE 802.15.4, aprobado en 2003 y modificado varias veces en los años siguientes, contribuye a todos estos objetivos. [9]

802.15.4 se denomina a un conjunto de protocolos de alto nivel de comunicaciones inalámbricas, opera en las frecuencias 868 MHz en Europa, 915 en Estados Unidos y 2,4 GHz en todo el mundo. El protocolo ZigBee ofrece una menor tasa de transferencia de datos de WiFi y Bluetooth, también ofrece un consumo eléctrico muy bajo y permite crear redes Ad-Hoc, en estrella, etc. Este protocolo está limitado por 65535 dispositivos divididos en subredes de 255, las cuales se comunicarán entre ellas por medio del coordinador de la red. Permite realizar una encriptación de datos de 128 bits. [16]

El estándar IEEE 802.15.4 proporciona conectividad de baja velocidad de datos entre dispositivos que consumen un mínimo de energía y se conectan en distancias cortas, se utiliza para el seguimiento, monitoreo, control, automatización, detección, aplicaciones para el hogar, entornos médicos (BAN), entre otros. [16]

Tecnologías inalámbricas para WSN

La comunicación inalámbrica en WSN se basa principalmente en tecnologías estandarizadas sobre 802.11 (redes de área local inalámbricas) y 802.15 (Redes de área personal inalámbricas). Hace uso de tecnologías como BlueTooth, Zigbee (IEEE 802.15.4), UWB (Ultra Wide Band), Wi-Fi, LoRa, SigFox, entre otras que sirven para la transmisión e información, monitoreo y accionamiento.

Tabla 1 Comparación de las diferentes tecnologías inalámbricas utilizadas en IoT [17]

Parámetros	**Bluetooth**	**LoRa**	**Wi-fi**	**WiMAX**	**ZigBee**
Autentificación	Llave de autentificación compartida	CCM	WPA2	CBC-MAC	CBC-MAC/Extensión de CCM
Protección de datos	16 bit CRC	128 bit CRC	32 bit CRC	128 bit CRC	16 bit CRC
Velocidad de datos	1-24 Mbps	0.3-50 kbps	1 Mbps - 6.75 Gbps	1 Mbps - 1 Gbps	250 kbps
Encriptación	Cifrado de flujo E0	Cifrado de bloque AES	Cifrado de flujo RC4 (WEP), AES cifrado de bloque	3DES, bloque AES	Cifrado de flujo, cifrado de bloque AES
Consumo de energía	Medio	Muy bajo	Alto	Medio	Muy bajo
Banda de frecuencias	2.4 GHz	868/900 MHz	5-60 GHz	2-66 GHz	868/915 GHz
Propagación	FHSS	CSS (Espectro de dispersió	DSSS, CCK, OFDM	OFDM	DSSS

		n Chirp)			
Estándar	IEEE 802.15.1	IEEE 802.15.4 g	IEEE 802.11 a/c/b/d/g / n	IEEE 802.16	IEEE 802.15.4
Topología	Malla, estrella, árbol	Estrella de estrellas	Estrella, P2P	Red de acceso por radio, malla	Árbol, P2P, estrella y malla
Alcance	8-10 metros	Menor a 30 km	20-100 m	Menor a 50 km	10-300 m

WSN se considera una de las tecnologías clave para implementar el Internet de las cosas (figura 4). Los dispositivos trabajan de manera colaborativa para recoger los datos y enviarlos a un colector central, eligiendo la ruta de comunicaciones optima (de dispositivo a dispositivo) a través de la red hasta llegar a su destino. Las redes de sensores son comúnmente bidireccionales, permitiendo configurar los dispositivos, enviar comandos, o actuar en su entorno. En este último caso, se les conoce como WSAN (del inglés Wireless Sensors and Actuator Networks). [15]

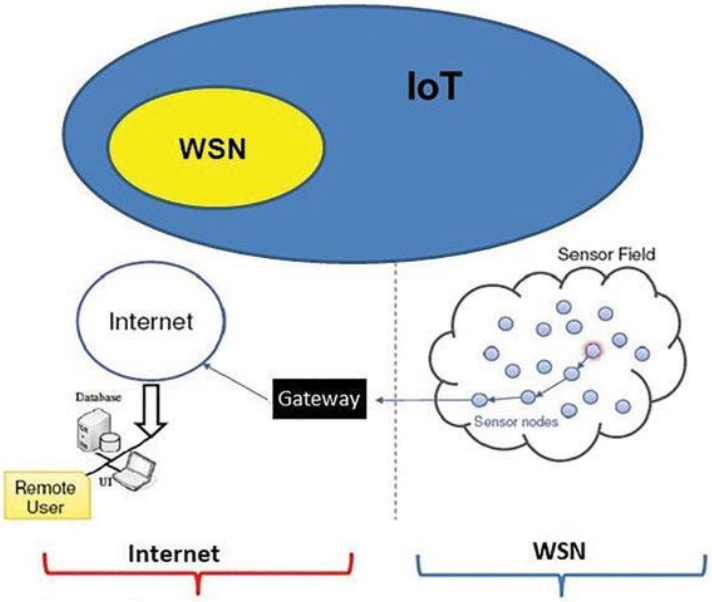


Figura 4 Conexión entre WSN e IoT [14]

WSN presenta una serie de ventajas, pero como toda tecnología también tiene sus desventajas, ver Tabla 2.

Tabla 2 Ventajas y desventajas de WSN [14]

Ventajas	Desventajas
WSN se puede configurar sin una infraestructura fija.	WSN es de baja velocidad en comparación con la comunicación por cable
Dado que es una red ad-hoc, puede agregar/eliminar nuevos dispositivos en cualquier momento.	La calidad de la comunicación puede fallar debido a muchos factores entre ellos ambientales.
La topología de la red es lo suficientemente flexible como para realizar cambios, permite escalabilidad.	Costoso para redes de sensores muy grandes
La estación base del nodo central se puede utilizar para retransmitir los datos detectados.	Recursos limitados como energía, ancho de banda, potencia de procesamiento, memoria, etc.
WSN es adecuado para lugares no tripulados o no accesibles	Mayor posibilidad a ataques en la red ya que maneja tecnología inalámbrica

Topología de la red inalámbrica

Para WSN es importante la topología de red, el cual consta del camino y la dispersión de sus elementos, en la actualidad hay muchas topologías para la red inalámbrica, tomando en cuenta que se pueden modificar dependiendo las necesidades, topologías como conexión punto a punto, estrella, malla, árbol.

- Una topología en estrella mantiene un enlace entre varios nodos con un punto central manteniendo la comunicación activa hasta los clientes
- La topología en malla consta de la interconexión a muchos clientes en el cual es permitido el cambio de información en una zona extensa donde hay numerosos clientes.
- La topología árbol normalmente los diferentes nodos se conectan a un backbone combinando las topologías estrella y bus con sus respectivas configuraciones.

Constitución de las WSN

- **Sensores:** son de distintos tipos cuya función es tomar la información presente en el medio y convertirlas en señales eléctricas
- **Nodos de sensor:** toman la información del sensor y la envían a la estación base
- **Gateway:** Elementos para la interconexión entre la red de sensores y una red TCP/IP.
- **Estación base:** recolector de datos.
- **Red inalámbrica**: bajo un estándar de conectividad

1.2.2.3 Protocolos de Comunicación IoT

Los protocolos de comunicación son reglas que posibilitan que dos o más dispositivos o sistemas se puedan comunicar entre sí, a través de diversos medios con un formato definido, ya sean implementados por hardware o software. [18]

Los dispositivos IoT se comunican mediante protocolos de comunicación específicos o que se pueden aplicar a su contexto. Los protocolos de IoT garantizan que un

dispositivo, puerta de enlace o servicio pueda comprender la información enviada por otro dispositivo, esta debe superar limitaciones de ancho de banda, velocidad, escalabilidad, etc. Se han diseñado y optimizado diferentes protocolos para distintos escenarios y usos. Es fundamental usar el protocolo adecuado en el contexto adecuado. [18] [19]

Otros aspectos importantes son la seguridad y la facilidad de implementación, la seguridad de los dispositivos IoT puede ser crítico en ciertos casos, y su implementación en diferentes entornos dado que en un lugar puede haber cientos de dispositivos.

Los protocolos de comunicación más utilizados son HTTP, MQTT y CoAP. Además, para comunicarnos, existen diferentes redes como LoRa o SigFox. Son redes WAN para el IoT y una alternativa a los sistemas tradicionales de comunicación. [20]

1.2.2.4 MQTT

MQTT es un protocolo ligero de transporte de mensajería de publicación/suscripción, diseñado para la telemetría e ideal para conectar dispositivos pequeños a redes con bajo ancho de banda, alta latencia o poco fiables.

Es un protocolo de mensajería asíncrona, desacopla el emisor y el receptor del mensaje tanto en el espacio como en el tiempo y, por lo tanto, es escalable en entornos de red con problemas de confiabilidad. Su flexibilidad permite dar soporte a diversos escenarios de aplicaciones para dispositivos y servicios IoT. [21]

Fue diseñado por Andy Stanford-Clark (IBM) y Arlen Nipper en 1999 para conectar sistemas de telemetría de oleoductos por satélite, fue lanzado libre de regalías en 2010 y se convirtió en un estándar OASIS en 2014. MQTT se está convirtiendo rápidamente en uno de los principales protocolos para las implementaciones de IOT (Internet de las cosas). [22]

MQTT es eficiente en términos de ancho de banda, independiente de los datos y tiene reconocimiento de sesión continua ya que usa TCP. Tiene la finalidad de minimizar

los requerimientos de recursos del dispositivo y, a la vez, tratar de asegurar la confiabilidad y cierto grado de seguridad de entrega con calidad del servicio. [23]

Broker MQTT

MQTT requiere el uso de un Broker central, también llamado servidor como se muestra en la figura 5:

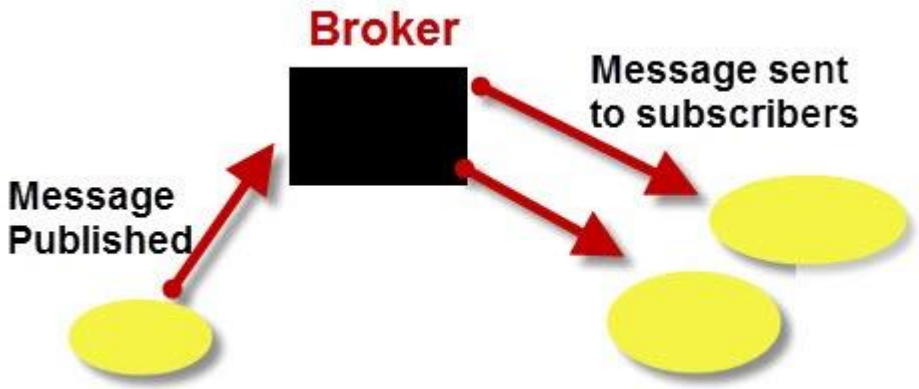


Figura 5 Broker MQTT [24]

- MQTT utiliza TCP/IP para conectarse al broker.
- TCP es un protocolo orientado a la conexión con corrección de errores y garantiza que los paquetes se reciban en orden.
- La mayoría de los clientes MQTT se conectarán al broker y permanecerán conectados incluso si no están enviando datos.
- El trabajo de un broker MQTT es filtrar los mensajes en función del tema y, a continuación, distribuirlos a los suscriptores.
- No existe una conexión directa entre un editor y un suscriptor.
- Los brokers MQTT normalmente no almacenan mensajes.
- El puerto que usa MQTT normalmente es el 1883
- Los clientes MQTT publican un mensaje “keepalive” a intervalos regulares (generalmente 60 segundos) que indica al intermediario que el cliente todavía está conectado. [24]

Calidad y seguridad de las comunicaciones

La calidad del servicio o QoS, y la seguridad, determina la robustez ante fallos e intromisiones.

En cuanto a su calidad se pueden determinar 3 niveles diferentes:

- **QoS 0:** el mensaje solo se envía una vez, y en caso de fallo no se entregaría. Se usa cuando no es crítico.
- **QoS 1:** el mensaje se enviará varias veces como se necesite hasta que llegue al cliente. Un inconveniente es que el cliente podría recibir varias veces un mismo mensaje.
- **QoS 2:** similar al anterior, pero se garantiza que solo se entrega una única vez. Se suele usar para sistemas más críticos donde se necesita mayor fiabilidad.

Con el tema de la seguridad de MQTT, se puede usar la autenticación del usuario y contraseña usando SSL/TLS. Aunque muchos dispositivos IoT con capacidades limitadas podrían tener problemas con la sobrecarga de trabajo al usar este tipo de comunicación segura, por lo que muchos aparatos IoT que usan MQTT usan contraseñas y usuarios en texto plano, también se puede configurar el broker para aceptar conexiones anónimas, lo cual no es recomendable ya que permitiría a cualquier usuario infiltrarse en la comunicación, implicando mayor riesgo. [18]

Muchos proyectos MQTT y servicios de IoT proporcionan un broker MQTT como los siguientes:

Tabla 3 Brokers MQTT

Broker	Descripción y características principales
Mosquitto 	Servidor MQTT de código abierto para que la comunidad realice pruebas, altamente portátil, ligero. Programado en C, multiplataforma. No admite agrupación de clústeres, problemas para escalamiento. [25]

	• Proporciona soporte completo de MQTT v3.1 v3.1.1 y en v5.0 algunas características no se utilizan directamente [26]
Jmqtt 	Jmqtt es un broker MQTT implementado por Java y Netty, admite persistencia y clúster. [25]
EMQ X 	Broker MQTT de código abierto altamente escalable y permite la creación de cluster con múltiples nodos lo que facilita el manejo de miles de conexiones. Tiene varias opciones de configuración, plugins. Provee un dashboard para su administración. • Compatibilidad MQTT V3.1/V3.1.1 y V5.0 • Compatibilidad con mensajes QoS0, QoS1, QoS2 • Compatibilidad con conversaciones persistentes, mensajes sin conexión, mensajes retenidos • Compatibilidad con conexiones TCP/SSL • Compatibilidad con la interfaz de publicación de mensajes HTTP • Compatibilidad con autenticación de nombre de usuario y contraseña [27]

HiveMQ  	Bróker MQTT listo para la empresa para proporcionar un movimiento rápido, eficiente y confiable de datos hacia dispositivos IoT. Tiene ediciones comerciales y de código abierto • HiveMQ es compatible con las versiones de MQTT: 3.1, 3.1.1, 5.0 • Sesiones limpias y persistentes • Mensajes en cola • Compatibilidad con mensajes QoS0, QoS1, QoS2 • Campos De nombre de usuario / contraseña • Intervalos de expiración de sesiones y mensajes [28]
HBMQTT	Construido sobre el marco de E/S asíncrono estándar de Python, implementa el conjunto completo de especificaciones del protocolo MQTT 3.1.1 • Admite el flujo de mensajes QoS 0, QoS 1 y QoS 2 • Soporte SSL a través de TCP y websocket • Autenticación a través de un archivo de contraseña, con opción para agregar más métodos • Sistema de plugins [29]

RabbitMQ 	Broker de mensajería AMQP Open Source, que tiene complemento con MQTT (incluido en la versión 3.x en adelante) [25]

Elaborado por el investigador

1.2.2.5 El modelo de publicación y suscripción

El protocolo MQTT define los tipos de entidades en la red: un intermediario de mensajes y un número de clientes. El intermediario es un servidor que recibe todos los mensajes de los clientes y luego los redirige a clientes de destino. Un cliente como un sensor o una aplicación puede interactuar con el intermediario para enviar y recibir mensajes. [23]

1. El cliente se conecta con el intermediario. Se puede suscribir a cualquier "tema" de mensajes de intermediario. Esta conexión puede ser una conexión TCP/IP o TLS cifrada.
2. El cliente publica el mensaje, sobre un tema, enviando el mensaje y el tema al intermediario.
3. Después, el intermediario reenvía el mensaje a todos los clientes que están suscritos a ese tema o tópico.

Ejemplo del modelo de publicación y suscripción:

Los mensajes de MQTT están organizados por temas, los sensores publicarán sus lecturas bajo el tema "sensor_data" y se suscribirán al tema "config_change". Las aplicaciones que guardan datos de sensor se suscribirán al tema "sensor_data". Se puede recibir órdenes del administrador del sistema para modificar las configuraciones de los sensores, como la sensibilidad y la frecuencia de muestreo, y publicar estos cambios en el tema "config_change" [30], ver figura 6.

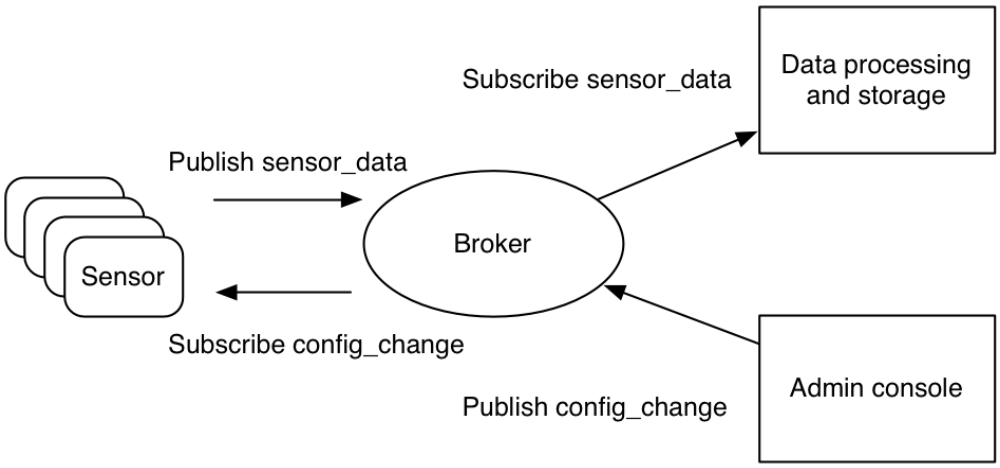


Figura 6 Modelo de publicación y suscripción de MQTT [30]

MQTT es liviano. Tiene una cabecera simple para especificar el tipo de mensaje, un tema basado en texto y una carga útil binaria y arbitraria con cualquier formato de datos como: JSON, XML, cifrado binario o Base64. [30]

1.2.2.6 Cloud Computing

La computación en la nube (Cloud Computing) como paradigma proporciona a las empresas y usuarios necesidades informáticas (como software, almacenamiento de datos, capacidad de procesamiento, etc.) a través de Internet que son fácilmente escalables bajo demanda. Los documentos, correos electrónicos y otros datos, así como las aplicaciones informáticas, se almacenarán "en la nube", es decir, en línea o internet. [31]

La definición del concepto de Cloud Computing más completa y ampliamente aceptada es la proporcionada por el National Institute of Standars and Technology:

"La computación en nube es un modelo que permite el acceso ubicuo, conveniente y bajo demanda a un conjunto compartido de recursos informáticos de cómputo configurables (por ejemplo: redes, servidores, almacenamiento, aplicaciones y servicios) que pueden aprovisionarse y liberarse rápidamente con un mínimo esfuerzo de gestión o interacción con el proveedor de servicios". NIST [32]

Para las empresas, las ventajas consisten principalmente en una reducción de la complejidad y los costes de mantenimiento de los sistemas. Sin embargo, tiene también inconvenientes si se producen pérdidas de conectividad o discontinuidad del servicio. Por lo tanto, debe proporcionarse el respaldo adecuado. Además, para garantizar la seguridad deben adoptarse las debidas medidas. [31]

La computación en la nube aplicado al loT tiene la ventaja de reducir los costes de capital, la movilidad y la capacidad global para el acceso, la facilidad de despliegue, flexibilidad y escalabilidad, así como la reducción de la infraestructura necesaria. [31]

1.2.2.7 Tipos de servicio de informática en la nube

Los tipos de informática en la nube son modelos de implementación de servicios que le permiten elegir el nivel de control sobre su información y los tipos de servicios que tiene que proporcionar. Hay tres tipos principales de servicios de informática en la nube y tiene la particularidad que se basan unos en otros. [33]

El primer tipo de informática en la nube es **infraestructura como servicio (IaaS)**, que se utiliza para el acceso a almacenamiento y capacidad informática mediante internet. Es la categoría más elemental de los tipos de informática en la nube y permite rentar infraestructura de TI como servidores, máquinas virtuales, almacenamiento, redes a un proveedor de servicios en la nube con el modelo de pago por uso. [33]

El segundo tipo de informática en la nube es **plataforma como servicio (PaaS)** en la cual los desarrolladores mediante herramientas pueden crear y hospedar aplicaciones web. PaaS está diseñado para dar acceso a los usuarios a los componentes que necesitan para desarrollar y utilizar con rapidez aplicaciones web o móviles a través de Internet, sin preocuparse por configurar y administrar la infraestructura de servidores. [33]

El tercer tipo de informática en la nube es **software como servicio (SaaS)**, es un método de entrega de aplicaciones de software a través de Internet donde los

proveedores de servicios en la nube hospedan y administran las aplicaciones, lo que facilita tener la misma aplicación en todos los dispositivos a la vez. [33]

Desde hace unos años han surgido nuevos modelos de servicios que combinan diferentes características y que ofrecen nuevas funcionalidades. Este tipo de nuevos modelos de servicio son muy variado, y cubre servicios de contenedores de aplicaciones, funciones o almacenamiento entre otros.

Función como servicio (FaaS), es un nuevo modelo de servicio que permite desarrollar, ejecutar y gestionar funcionalidades de aplicaciones evitando la complejidad de la infraestructura para el soporte, desarrollo y ejecución de una aplicación. Construir una aplicación siguiendo este modelo es seguir la arquitectura "serverless" o sin servidor. Este modelo se utiliza principalmente para la construcción de microservicios. FaaS es relativamente joven y su modelo fue seguido por AWS Lambda, Google Cloud Functions, Microsoft Azure Functions y muchos otros. [34]

Un servicio de FaaS significa que se puede cargar en la nube funcionalidades modulares que se ejecutan de forma independiente. Antes de esto, se necesitaba escalar un servidor monolítico para manejar la carga de trabajo.

Almacenamiento como servicio (StaaS), permite utilizar servicios de almacenamiento virtual a través de un software para la gestión del almacenamiento como servicio con coste por GigaByte almacenado y transferencia de datos. Este tipo de almacenamiento está en auge entre las pequeñas y medianas empresas, ya que no se requiere un presupuesto inicial para configurar los discos duros, los servidores o el propio personal de TI, además es una excelente técnica para mitigar los riesgos en la recuperación de desastres al proporcionar almacenamiento de datos a largo plazo y mejorar la estabilidad en cuanto al acceso a los mismos, cuenta con problemas relativos a la localización de los datos y la regulación de los mismos. [34]

Contenedores como servicio (CaaS). En lugar de virtualizar la pila de hardware como en el caso de las máquinas virtuales en el IaaS, los contenedores se virtualizan a nivel de sistema operativo, con múltiples contenedores ejecutándose directamente sobre el

núcleo del Sistema Operativo (SO). Son más ligeros, su inicio es más rápido y utilizan una fracción de la memoria en comparación con el arranque de todo un sistema operativo. [34]

Los contenedores pueden ejecutarse prácticamente en cualquier lugar, lo que facilita enormemente el desarrollo y la implementación

1.2.2.8 Microservicios

El enfoque tradicional para el diseño de aplicaciones se centraba en la arquitectura monolítica, en la cual todos los elementos y procesos estaban contenidos en un solo servicio. Este enfoque tiene sus desventajas: cuanto más grande es la aplicación, más difícil es solucionar los problemas que se presentan, y agregar nuevas funciones rápidamente [35]. Si un proceso experimenta alta demanda, se debe escalar toda la arquitectura lo cual es ineficiente, si se necesita agregar o mejorar características se dificulta la implementación ya que aumenta el riesgo de disponibilidad debido a un error de un proceso. [36]

Microservicios se refiere a una forma de diseñar software como un conjunto de pequeños servicios independientes que cooperan entre sí mediante protocolos livianos de comunicación. [37]

Mediante la arquitectura de microservicios, se crea una aplicación con componentes separados que ejecutan cada proceso de aplicación como un servicio. Estos servicios se comunican a través de una interfaz bien definida mediante API. Debido a que se ejecutan de forma independiente, cada servicio se puede actualizar, implementar y escalar para satisfacer las funciones específicas de una aplicación. [36]

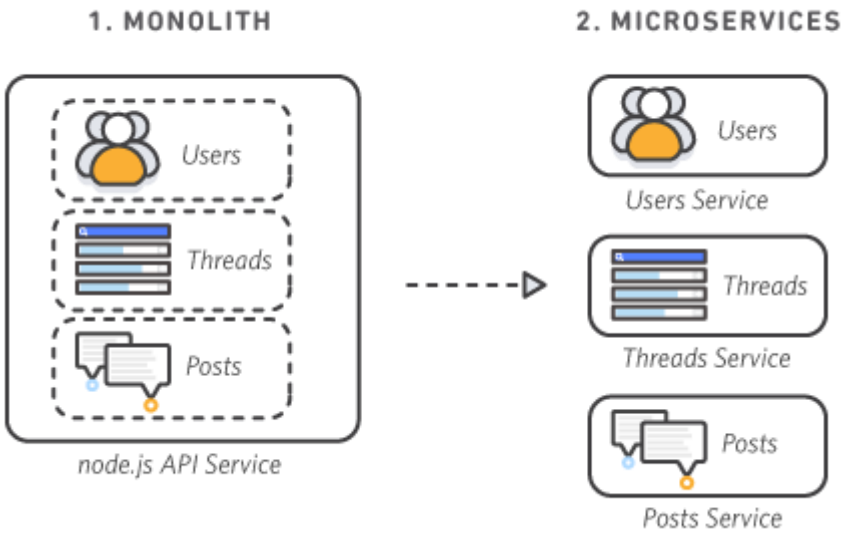


Figura 7. Aplicación monolítica en microservicios [36]

Una arquitectura de microservicios ofrece cada uno de los servicios de forma aislada, siendo más fáciles de mantener y de responder frente a niveles de escalabilidad horizontal y vertical. Permite replicar aquellas instancias de microservicios que sean necesarios, evita replicar una copia total de la aplicación, como es en el caso de la arquitectura monolítica.

Tabla 4 Ventajas y desafíos de los microservicios [20]

Ventajas	Desafíos
Agilidad: Permite actualizar un servicio sin volver a implementar toda la aplicación. Facilita la corrección de errores y administración de versiones.	**Complejidad:** Cada servicio es más sencillo, pero el sistema como un todo es más complejo. Además, se debe gestionar cada servicio.
Equipos de trabajo pequeños: Un microservicio debe ser lo suficientemente pequeño como para que un solo equipo de características lo pueda administrar, esto favorece la agilidad y productividad.	**Desarrollo y pruebas:** Las herramientas existentes no siempre están diseñadas para trabajar con dependencias de servicios.

Base de código pequeña: minimiza las dependencias y resulta más fácil agregar nuevas características a los servicios.	**Falta de gobernanza:** puede acabar con tantos lenguajes y marcos de trabajo diferentes que la aplicación puede ser difícil de mantener.
Mezcla de tecnologías: Los equipos pueden elegir una o varias tecnologías que mejor se adapten al servicio.	**Congestión y latencia de red:** El uso de muchos servicios puede congestionar la comunicación interservicios.
Aislamiento de errores: Si un microservicio individual no está disponible, no interrumpe toda la aplicación.	**Integridad de datos:** Cada microservicio es responsable de la conservación de sus propios datos. Como consecuencia, la coherencia de los datos puede suponer un problema
Escalabilidad: Los servicios se pueden escalar de forma independiente.	**Administración:** Para tener éxito con los microservicios se necesita una cultura de DevOps consolidada.
Aislamiento de los datos: Al verse afectado solo un microservicio, es mucho más fácil realizar actualizaciones del esquema. En una aplicación monolítica, las actualizaciones del esquema pueden ser muy complicadas, ya que las distintas partes de la aplicación pueden modificar los mismos datos.	**Control de versiones:** Es posible que varios servicios se actualicen en cualquier momento, podrían surgir problemas con la compatibilidad

Los microservicios complementan muy bien a las arquitecturas de aplicaciones basadas en la nube al permitir que los equipos de desarrollo de software aprovechen diferentes modelos, como escenarios de programación basados en eventos, y que estos escalen automáticamente. [38]

Un paradigma emergente es utilizar clústeres de contenedores para implementar pequeños servicios. Los contenedores permiten aislar, encapsular e implementar microservicios y orquestar horizontalmente un grupo de contenedores dentro de una aplicación. [38]

1.2.2.9 Contenedores y Virtualización

El despliegue de software en la nube, a menudo se apoya sobre tecnologías de virtualización, las máquinas virtuales (VM), contienen el sistema operativo y sus dependencias, además de virtualizaciones de hardware, y las configuraciones de cada programa. Esto implica redundancia y produce un uso ineficiente de recursos que aumenta si la máquina virtual debe escalar horizontalmente. [37]

Esto ha motivado la aparición de otras formas de virtualización más livianas, como los contenedores que, no requieren un sistema operativo completo dedicado para cada contenedor. En esencia, contienen solamente los binarios, bibliotecas y middleware necesarios para ejecutar la aplicación desplegada. Estos son empaquetados en una "imagen" que es administrada y ejecutada por un gestor de contenedores por ejemplo Docker. [37]

En la Figura 8 se muestra una comparación de alto nivel entre los conceptos de VM y contenedor.

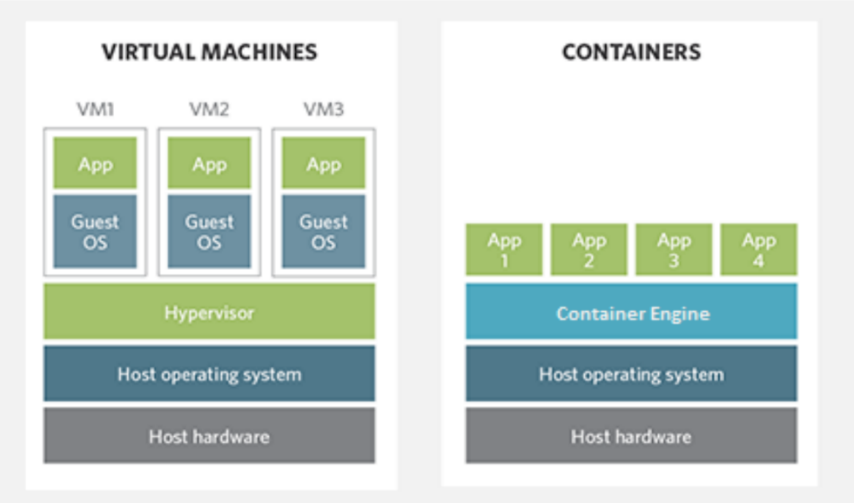


Figura 8 Comparación de VM y contenedores [37]

La palabra contenedor se define como "un objeto para contener/resguardar o el transporte de algo". La idea detrás de los contenedores de "software" es similar. Son imágenes aisladas e inmutables que proporcionan funcionalidad diseñada en la mayoría de los casos para ser accesibles sólo a través de sus APIs. Es una solución para hacer que el software funcione de forma fiable y en casi cualquier entorno. No importa dónde se estén ejecutando (computador portátil, servidor de pruebas o de producción, centro de datos, etc.), el resultado siempre debe ser el mismo. [39]

Los microservicios están muy relacionados al concepto de contenedor, una unidad estándar de software que empaqueta el código junto a todas de sus dependencias para que el servicio o aplicación se ejecute de forma rápida y fiable un cualquier entorno informático, es posible escalar rápidamente, y el reinicio de contenedores frente a fallas con un bajo impacto en la disponibilidad. [37] [40]

1.2.2.10 Docker

El lanzamiento de Docker en 2013 inició una revolución en el desarrollo de aplicaciones, al democratizar los contenedores de software. Docker desarrolló una tecnología de contenedor Linux, que es portátil, flexible y fácil de implementar.

Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones, permite separar las aplicaciones de la infraestructura para que pueda entregar software rápidamente. Con Docker, se puede administrar la infraestructura de la misma manera que se administra las aplicaciones. [41]

Una imagen de contenedor de Docker es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación. Las imágenes de contenedor se convierten en contenedores en tiempo de ejecución y, en el caso de los contenedores de Docker, las imágenes se convierten en contenedores cuando se ejecutan en Docker Engine. [42]

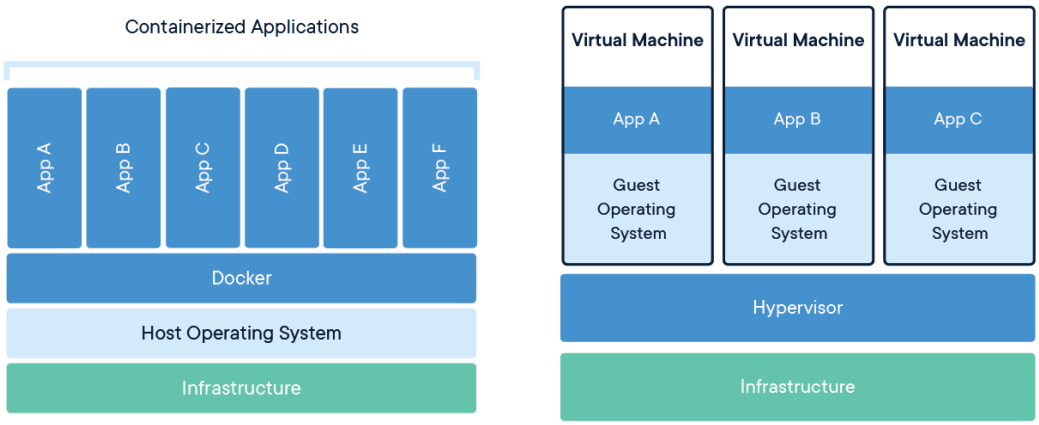


Figura 9 Contenedores Docker y máquinas virtuales [42]

1.2.2.11 API REST

Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas. API significa interfaz de programación de aplicaciones [43]

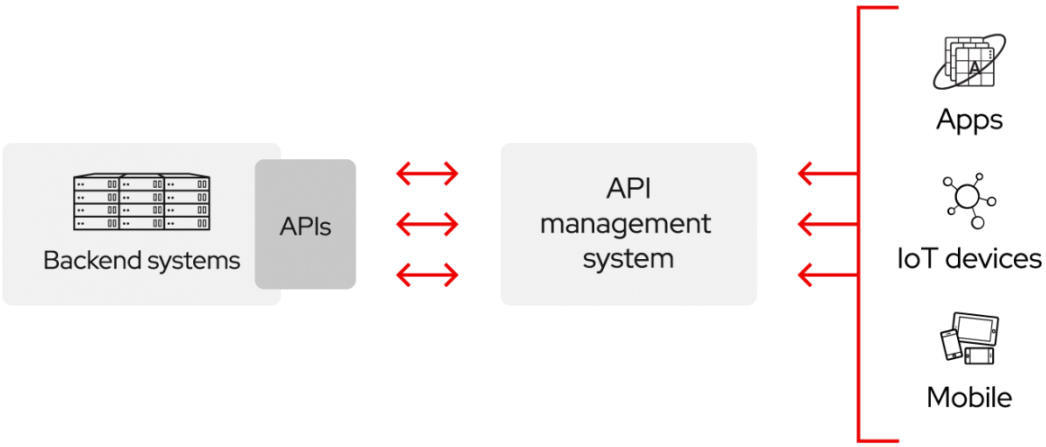


Figura 10 Sistema de Gestión API [43]

El propósito de una API es intercambiar datos entre diferentes sistemas, la mayoría de las veces estos intercambios de datos tienen como objetivo automatizar procesos y creación de nuevas funcionalidades. En el ámbito web, podríamos decir que una API es un servicio backend que se utiliza para conectar dos aplicaciones. [44]

Las API brindan acceso a los recursos mientras mantiene la seguridad y el control. Su seguridad depende de una gestión adecuada incluido el uso de puertas de enlace API. Para conectarse a las API y crear aplicaciones que usen los datos o la funcionalidad que brindan, puede usar una plataforma de integración distribuida que conecte cosas, como sistemas heredados e Internet de las cosas (IoT). [43]

Las APIs REST se distinguen por que se basan en el protocolo de aplicación HTTP. Es decir, utilizan códigos y métodos de respuesta HTTP para una función específica y ampliamente reconocida. Y nos permite a través de la URI, la estructuración de los recursos disponibles.

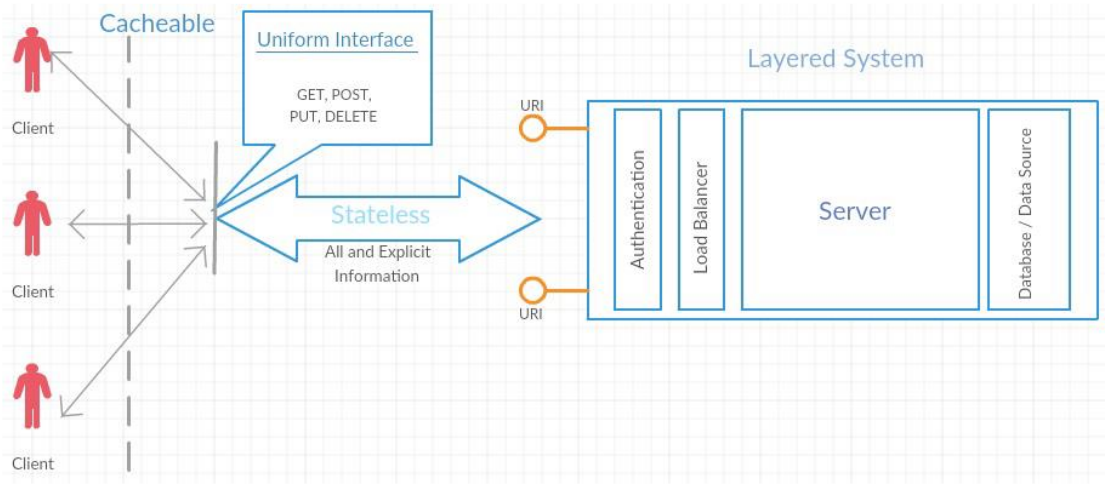


Figura 11 Arquitectura REST [45]

La petición se envía desde el cliente al servidor vía HTTP en forma de URL web con sus respectivas cabeceras. Usando un método HTTP, después, se envía una respuesta desde el servidor en forma de recurso, que puede ser algo como HTML, XML, JSON.

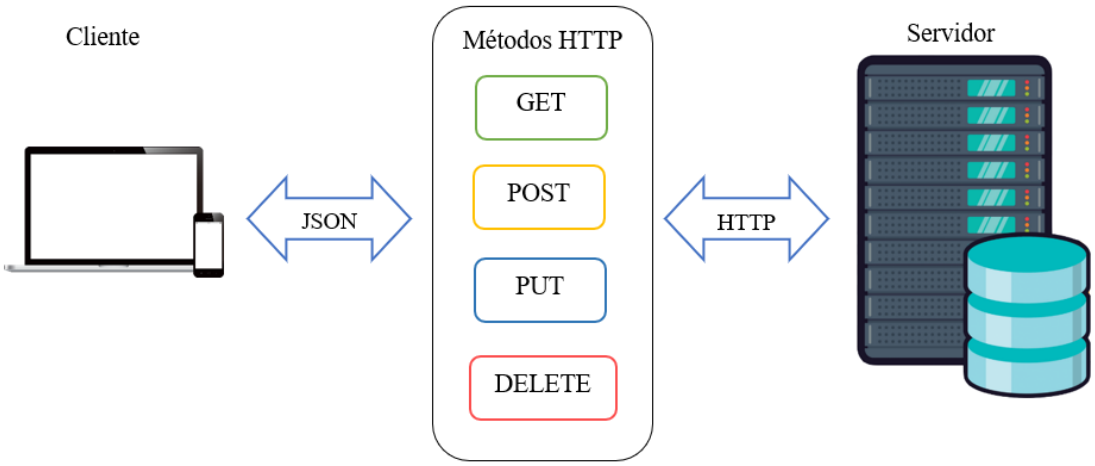


Figura 12 Ejemplo petición cliente-servidor

Elaborado por el investigador

Entre las ventajas de una API REST está la posibilidad de crear un cliente / servidor en diferentes idiomas (lenguajes de programación). Esto nos da la capacidad de enviar y recibir información en diferentes formatos, aunque JSON (JavaScript Object Notation) se usa comúnmente (Figura 14). Tienen mejor rendimiento que los servicios SOAP, debido a su peso ligero y su capacidad para almacenar en caché fácilmente llamadas a través de encabezados de control HTTP.

```
GET /users/007

{
    "id": "007",
    "name": {
        "first_name": "James",
        "last_name": "Bond"
    },
    "address": {
        "street": "New Bond Street",
        "city": {
            "name": "London",
            "post code": "W1S 1S"
        }
    }
}
```

Figura 13 Ejemplo de respuesta JSON de una API [30]

Las API web que funcionan con las limitaciones de arquitectura REST se llaman API RESTful. La diferencia más básica entre SOAP y REST es que SOAP es un protocolo, mientras que REST es un estilo de arquitectura. Esto significa que no hay ningún estándar oficial para las API RESTful, las API son RESTful siempre que cumplan con las 6 limitaciones principales:

- Arquitectura cliente-servidor
- Sistema sin estado
- Capacidad de almacenamiento en caché
- Sistema en capas
- Disponibilidad del código según se solicite
- Interfaz uniforme

Estas limitaciones son más sencillas que un protocolo definido previamente. Por eso, las API de RESTful son cada vez más frecuentes que las de SOAP.

La principal ventaja de las API REST es que podemos desarrollar una API en el backend y utilizarla en cualquier dispositivo, ahorrando así mucho tiempo de desarrollo. [44]

Identificadores de recursos

Las llamadas al API se implementan como peticiones HTTP

Para asegurarse de que una aplicación esté manejando el recurso correcto, es necesario un mecanismo para identificar unívocamente un recurso en la red, identificadores uniformes de recursos (URI) [9], definidos en RFC 3986 [46]

Se puede utilizar un URI para direccionar un recurso, de modo que pueda ser localizado, recuperado y manipulado, es una relación 1: N entre un recurso y URI: un recurso puede ser asignado a varios URI, pero un URI apunta exactamente a un recurso. Los URI pueden ser de dos tipos:

- Un nombre de recurso uniforme (URN): especifica el nombre de un recurso, por ejemplo: urn:ietf:rfc:2616

- Un localizador de recursos uniforme (URL) especifica cómo ubicar el recurso, la URL representa el recurso, por ejemplo: http://example.com/books/123

Las URNs usan el esquema urn, mientras que los recursos web usan el esquema http. Las URL incluyen toda la información necesaria para abordar correctamente el recurso. Una URL tiene la forma que se muestra en la Figura 15.

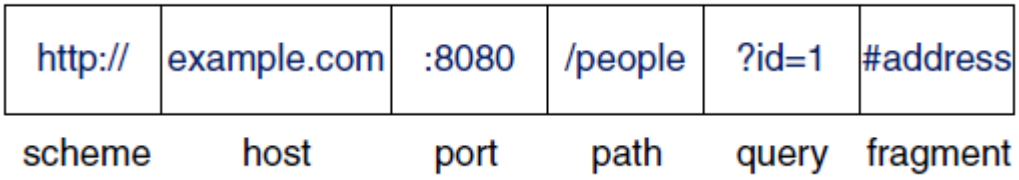


Figura 14 Ejemplo estructura de URL genérica [9]

El host puede ser una dirección IP o un nombre de dominio completo, que debe resolverse mediante el sistema DNS. La consulta contiene información coincidente para filtrar el resultado. Finalmente, el fragmento se puede utilizar para identificar una parte específica del recurso. Los URI deben ser cortos y no exponer ninguna noción específica del formato utilizado para representar el recurso objetivo. Por ejemplo, http://ejemplo.com/people/123 es un buen URL, mientras que http://example.com/people/123.xml y http://example.com/people/123.json no lo son. [9]

1.2.2.12 Tecnologías usadas para la creación de una API REST

Los lenguajes de programación y frameworks más usados para la creación de una API REST:

Flask

Flask es un micro framework, simple escrito en Python que permite crear aplicaciones web pequeñas o medianas rápidamente y con un mínimo número de líneas de código, ideal para pincipiantes. Flask depende del motor de plantillas Jinja y del kit de herramientas Werkzeug WSGI.

Django

Django es un framework de aplicaciones web escrito en Python con el cual se pueden abordar todo tipo de proyectos web escalables, seguros, mantenibles y de alta calidad, es gratuito y de código abierto. Django se encarga de gran parte de las complicaciones del desarrollo web, por lo que no es necesario reinventar código, así permite un desarrollo ágil. [47]

Características:

- **Completo:** Django provee casi todo lo que los desarrolladores deseen que tenga por defecto, sigue principios de diseño consistentes.
- **Versátil:** Puede construir casi cualquier tipo de sitio web, funciona con cualquier framework en el lado del cliente, y retorna contenido en muchos formatos.
- **Seguro:** Django ayuda a los desarrolladores evitar varios errores comunes de seguridad, permite protección contra algunas vulnerabilidades de forma predeterminada.
- **Escalable:** En Django cada parte de la arquitectura es independiente de las otras, y por lo tanto puede ser reemplazado o cambiado si es necesario.
- **Mantenible:** El código usa principios y patrones de diseño para fomentar la creación de código mantenible y reutilizable.
- **Portable:** Django está escrito en Python, por lo que se puede ejecutar en muchas plataformas.

[47]

Node.js

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, permite establecer y gestionar múltiples conexiones al mismo tiempo. [48]

Características:

- **Asíncrono y controlado por eventos:** Node.js es asíncrono, permite manejar múltiples peticiones en la web.
- **Muy rápido:** al estar construido en el motor JavaScript V8 de Google Chrome, la biblioteca Node.js es muy rápida en la ejecución de código.
- **Sin almacenamiento en búfer:** las aplicaciones Node.js nunca almacenan en búfer ningún dato. Estas aplicaciones simplemente generan los datos en fragmentos.
- **Procesos en un solo hilo, pero altamente escalable:** ayuda al servidor a responder sin bloqueos y hace que el servidor sea altamente escalable en comparación con los servidores tradicionales que crean hilos limitados para manejar las solicitudes.
- Simplicidad
- Buena integración con servicios basados en JSON
- Licencia basada en código abierto.
  [48] , [49]

Puede hacer uso de express que es el framework web más popular de Node.

Express es un marco de código abierto del lado del servidor creado para Node.js, proporciona complementos, código de plantilla, paquetes de middleware y funcionalidad de enrutamiento para un desarrollo web más rápido y eficiente. Además, permite integraciones de bibliotecas para facilitar la personalización. [50]

Express es minimalista pero los desarrolladores han creado paquetes de middleware compatibles para abordar casi cualquier problema de desarrollo web.

Entre las aplicaciones de interés de node.js se puede destacar:

- **Las aplicaciones de Internet de las Cosas:** En IoT múltiples sensores envían un gran número de peticiones. Node.js es capaz de gestionar estas peticiones concurrentes con rapidez.
- **Aplicaciones basadas en REST API**: JavaScript se utiliza tanto en el frontend como en el backend de los sitios. Así, un servidor puede comunicarse fácilmente con el frontend a través de APIs REST utilizando Node.js

Laravel

Laravel es un framework de PHP de aplicaciones web con sintaxis expresiva y elegante. Es un marco web que proporciona una estructura y un punto de partida para crear aplicaciones con excelente experiencia de desarrollo, proporciona características poderosas como una inyección de dependencias completa, una capa de abstracción de base de datos expresiva, colas y trabajos programados, pruebas unitarias y de integración, y más. [51] [52]

Características:

- **Arquitectura MVC (Modelo-Vista-Controlador):** permite relacionar de manera clara y sencilla todas las partes de una aplicación, instanciando clases y métodos sin la necesidad repetir código.
- **Eloquent ORM**: es muy intuitivo para escribir consultas en PHP sobre objetos.
- **Seguridad**: ofrece un nivel bastante fuerte con mecanismos de hash y salt para encriptar por medio de librerías como BCrypt
- **Librerías y modularidad:** Laravel aparte de sus propias librerías cuenta con ayuda de Symfony en otras
- **Base de datos y migraciones**: Permite actualizar y migrar la base de datos de forma fácil. [52]

Spring

Spring es un marco ligero y de código abierto para crear aplicaciones empresariales modernas basadas en Java, proporciona soporte de infraestructura que se centra en la lógica empresarial a nivel de aplicación, sus módulos y marcos están optimizados para ofrecer el rendimiento deseado, la experiencia de desarrollo optimizada y la seguridad. [53]

Características:

- Modular
- Acceso a datos: soporte DAO, JDBC, ORM, Marshalling XML.
- Gestión de transacciones.
- Integración: comunicación remota, JMS, JCA, JMX, correo electrónico, tareas, programación, caché.

- Programación orientada a aspectos (AOP): permite la implementación de rutinas transversales.
- Facilita en gran medida la programación basada en MVC y una implementación rápida basada en Inyección de Dependencias
- Permite el procesamiento de datos por lotes

[54]

Tabla 5 Comparación de tecnologías para desarrollar una API [55] [56] [57] [53]

Nombre	Lenguaje	Ventajas	Desventajas
Flask	Python	• Es muy sencillo de utilizar, minimalista • Incluye servidor web propio para pruebas • Se integra con otras herramientas para incrementar sus funciones • Su velocidad es mejor a comparación de Django	• Genera dificultades a la hora de realizar migraciones o pruebas unitarias • El sistema de autenticación de usuarios de Flask es muy básico. • Pocas funcionalidades incluidas • Requiere de más trabajo para construir estructuras que en otros frameworks
Django	Python	• Contiene muchas funcionalidades ya incluidas • Framework maduro, es compatible con el diseño de la plantilla del modelo MTV • Incorpora una amplia variedad de paquetes de librerías • Proporciona una estructura de código autogenerado • Administración sencilla de bases de datos, soporta ORM. • Para proyectos grandes y robustos. • Fácil de mantener.	• Medianamente complejo • Velocidad reducida en comparación con otros frameworks • Tiene rasgos estrictos y, por lo tanto, ofrece una flexibilidad limitada.
Node.js	JavaScript	• Sigue la arquitectura MVC • Permite procesar varias solicitudes simultáneamente, lo que resulta en una mayor escalabilidad y un rendimiento más rápido. • Flexible y mínima para desarrollar API • Diseñado para admitir la captura de errores en códigos sincrónicos y asincrónicos. • Es beneficioso para desarrollar aplicaciones web rápidamente	• Falta de estandarización, hay muchos caminos para realizar las cosas. • Dado que muchas funciones deben escribirse desde cero, puede experimentar una disminución en la productividad. • Algunas herramientas de código abierto carecen de calidad. • No es adecuado para procesamiento pesado, uso intensivo de CPU

		• Gran cantidad de plugins, librerías • Trabaja con Middleware de todo tipo • Buena compatibilidad con MongoDB • Modelo de programación asíncrona y no bloqueante	
Laravel	PHP	• Fácil de usar y aprender • Aplicaciones web de calidad • Crear aplicaciones con arquitecturas predefinidas • Extensa documentación y ejemplos • Facilidad de mantenimiento y escalabilidad • Facilita el trabajo en equipo y promueve buenas prácticas • Ofrece sus propias bibliotecas modulares de manera preinstalada • Para proyecto pequeños a grandes	• Fuerte orientación a la programación estática. • Velocidad baja en comparación con otros frameworks. • Muchas librerías de terceros hacen que seleccionar el adecuado sea más dificultoso.
Spring	Java	• Reduce significativamente el tiempo de desarrollo • Ofrece modularidad a los desarrolladores al permitirles elegir paquetes y clases adecuados según las necesidades del proyecto. • Programación Orientada a Aspectos (AOP) • Aplicaciones escalables, es flexible y tiene capacidades de procesamiento paralelo	• Requiere experiencia en desarrollo de alto nivel • Exige experiencia en XML • Carece de directrices relacionadas con la seguridad para la falsificación de solicitudes entre sitios o XSS • Consume una gran cantidad de memoria en comparación con otros frameworks

1.2.2.13 Protocolo HTTP

HTTP (Hypertext Transfer Protocol) nos permite realizar una petición de datos y recursos, es un protocolo basado en el principio de cliente-servidor. Usualmente es un navegador Web, pero podría ser cualquier otro programa que maneje este protocolo. [58]

HTTP ha ido evolucionando con el tiempo. Es un protocolo de la capa de aplicación, y se transmite sobre el protocolo TCP, o TLS (protocolo encriptado).

Mensajes HTTP

HTTP/1.1 y versiones anteriores los mensajes eran de formato texto, en HTTP/2, los mensajes están estructurados en un nuevo formato binario y las tramas permiten la compresión de las cabeceras y su multiplexación, la semántica de cada mensaje es la misma. [58]

Existen dos tipos de mensajes HTTP: peticiones y respuestas, cada uno sigue su propio formato.

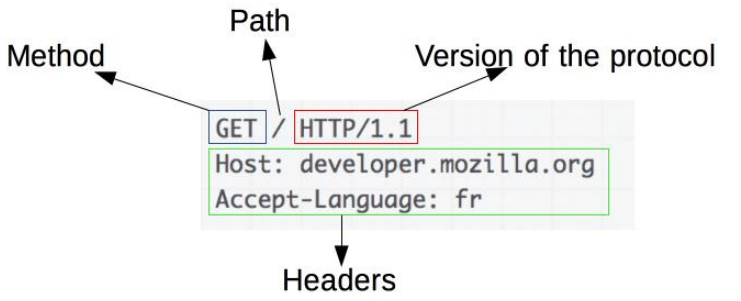


Figura 15 Ejemplo de petición HTTP [58]

Una petición de HTTP, está formado por los siguientes campos:

- Un método de petición HTTP, indica la acción que el cliente desea realizar para un recurso determinado. Se utilizan los verbos HTTP GET, POST, PUT y DELETE para el acceso, creación, actualización y borrado de recursos.

Además, cualquier dispositivo que sepa cómo utilizar HTTP será capaz de consumir una API REST. [59] [44]

Tabla 6 Peticiones HTTP [59]

Peticiones HTTP	Descripción
GET	Solicita datos de un recurso en específico
POST	Se utiliza para enviar una entidad a un recurso en específico
PUT	Reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
DELETE	Borra un recurso en específico.
PATCH	Es utilizado para aplicar modificaciones parciales a un recurso.

- La dirección del recurso pedido; la URL del recurso, sin los elementos obvios por el contexto, como pueden ser: sin el protocolo (http://), el dominio o el puerto TCP.
- La versión del protocolo HTTP.
- Cabeceras HTTP opcionales, que pueden aportar información adicional a los servidores.
- Un cuerpo de mensaje. [58]

Un ejemplo de repuesta:

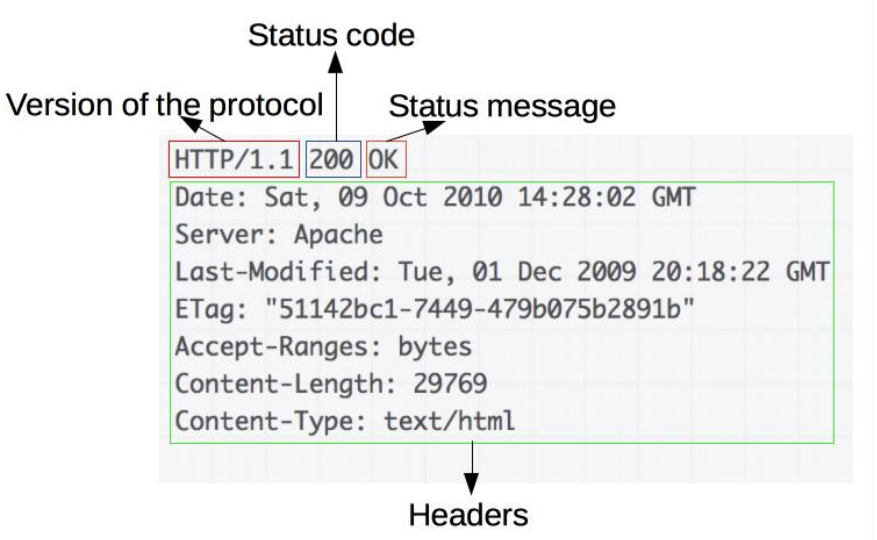


Figura 16 Ejemplo de respuesta HTTP [58]

Las respuestas están formadas por los siguientes campos:

- La versión del protocolo HTTP.
- Un código de estado, indicando si la petición ha sido exitosa, o no, y debido a que.
- Un mensaje de estado con la descripción del código de estado.
- Cabeceras HTTP.
- El recurso que se ha pedido.

Códigos de Estado de respuesta HTTP

Los códigos de estado de respuesta HTTP son mensajes que devuelve el servidor cada vez que el cliente realiza una petición al servidor, indican si se ha completado satisfactoriamente una solicitud HTTP específica. Se agrupan en:

- Respuestas informativas (100–199),
- Respuestas satisfactorias (200–299),
- Redirecciones (300–399),
- Errores de los clientes (400–499),

- y errores de los servidores (500–599).

Los códigos HTTP están estandarizados y se recogen en el registro de códigos de estado HTTP de la IANA (Internet Assigned Numbers Authority).

Cabeceras HTTP

Las especificaciones HTTP tienen un conjunto de encabezados estándar (en inglés headers), a través del cual un cliente puede obtener información sobre un recurso solicitado y llevar los mensajes que indican sus representaciones y pueden servir como directivas para controlar cachés intermediarios, permiten al cliente y al servidor enviar información adicional junto a una petición o respuesta. [45]

1.2.2.14 Bases de datos

Una base de datos es una recopilación organizada de información o datos estructurados, que normalmente se almacena de forma electrónica en un sistema informático.

Los datos estructurados son almacenados en sistemas de administración de base de datos relacionales. Estos sistemas presentan un lenguaje de programación utilizado para la gestión y extracción de información llamado Structured Query Language (SQL).

Existen muchos tipos diferentes de bases de datos y depende de cómo y con que fin se deseen utilizar los datos.

- **Bases de datos relacionales:** los elementos se organizan en tablas con columnas y filas, proporciona la forma más eficiente y flexible de acceder a información estructurada.
- **Bases de datos orientadas a objetos:** la información se representa en forma de objetos
- **Bases de datos distribuidas**: consta de dos o más archivos que se encuentran en sitios diferentes. La base de datos puede almacenarse en varios ordenadores, ubicarse en la misma ubicación física o repartirse en diferentes redes.

- **Bases de datos NoSQL:** permite almacenar y manipular datos no estructurados y semiestructurados (a diferencia de una base de datos relacional, que define cómo se deben componer todos los datos insertados en la base de datos). Su popularidad se debe a que las aplicaciones web se volvían más comunes y complejas.
- **Bases de datos orientadas a grafos:** almacena datos relacionados con entidades y las relaciones entre entidades.
  [60]

Tabla 7 Comparación entre SQL y NoSQL

Elaborado por: El investigador

SQL	NoSQL
SQL permite combinar de forma eficiente diferentes tablas para extraer información relacionada	Bases de datos mucho más abiertas y flexibles
Llevan más tiempo en el mercado, tienen mayor soporte, experiencia y madurez	Gran uso en la última década para aplicaciones modernas
Bases de datos robustas	NoSQL permite un escalado horizontal por su capacidad de distribución
Los datos deben cumplir requisitos de integridad tanto en tipo de dato como en compatibilidad.	No todas las bases de datos NoSQL contemplan la atomicidad de las instrucciones y la integridad de los datos.
Lenguaje SQL para consultas de datos (escritura simple) y relaciones, atomicidad	Optimización de consultas en base de datos para grandes cantidades de datos.
Estándares bien definidos, Todos los procesos deben estar bajo los estándares que plantea el SQL	Falta de estandarización. Hay muchas bases de datos NoSQL y aún no hay un estándar como sí lo hay en las bases de datos relacionales.

Los datos semi-estructurados y no estructurado son almacenados en sistemas de base de datos conocidos como NoSQL. [61]

MongoDB

MongoDB es un sistema de base de datos distribuida de código abierto, NoSQL, basado en documentos, de uso general, sirve para aplicaciones modernas y para la era de la nube.

MongoDB no guarda los datos en tablas como las bases de datos relacionales, los guarda en estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

En el Internet de las cosas requiere la recolección de datos de los dispositivos, el almacenamiento y procesamiento de datos, para esto requiere significativamente más flexibilidad, agilidad y escalabilidad.

Las ventajas de crear una aplicación de Internet de las cosas con MongoDB:

- Modelo de datos del documento. Con MongoDB, puede administrar e incorporar datos en cualquier estructura. Esto le permite iniciar e iterar en su aplicación sin tener que comenzar desde cero para cumplir con los requisitos en evolución.
- Escala a bajo costo. Las aplicaciones de IoT procesan grandes volúmenes de datos a través de sensores, por lo que el sistema tendrá que escalar de forma rápida y económica. Una de las ventajas de MongoDB es la capacidad de escalar horizontalmente en hardware básico de bajo costo en su centro de datos o en la nube.
- Analiza cualquier tipo de datos. El análisis en tiempo real dentro de la base de datos significa que no obtiene el retraso de tiempo que normalmente procesaría los datos a través de un costoso sistema de almacenamiento de datos. [62]

MongoDB es flexible con modelos de objetos con estructuras más complejas o que puedan cambiar con frecuencia como es el caso de estructuras de modelos de sensores IoT ya que estos pueden cambiar mucho de acuerdo a cada usuario o programador de los microcontroladores y las redes de sensores. [63]

Ya que el sistema será usado por varios usuarios, cada uno podrá manejar la estructura de datos como mejor crea, ya que si se maneja con estructuras SQL se tiene impedimentos para agregar más campos o menos debido a una estructura fija, además que existe inconvenientes para migraciones de datos.

1.2 Objetivos

1.2.1 Objetivo General

Desarrollar una API REST para la transmisión de información y control de redes de sensores IoT.

1.2.2 Objetivos Específicos

- Analizar las tecnologías usadas para el desarrollo de una API REST y protocolos de comunicación para redes de sensores IoT.
- Implementar el protocolo MQTT para la conectividad de las redes de sensores IoT con la aplicación.
- Desplegar una API REST para la transmisión de información y control de las redes de sensores IoT a cualquier cliente HTTP, alojado en un servicio en la nube.

CAPÍTULO II

METODOLOGÍA

2.1 Materiales

Para la elaboración del presente proyecto de investigación se utilizó información de artículos, tesis, libros, revistas, documentación web sobre protocolos de comunicación usados en redes de sensores IoT, el diseño, programación de API REST, librerías necesarias, bases de datos, repositorio del código, además de la documentación oficial de los servicios de nube usados.

2.2 Métodos

2.2.1 Modalidad de investigación

El presente proyecto es una investigación aplicada, porque se empleó los conocimientos ya existentes para la creación de una API REST mediante tecnologías que se usan en la actualidad, se dio solución a usuarios que necesiten transmitir información de sus dispositivos, sensores IoT hacia una API alojada en la nube en la cual pudieron conectar sus redes de sensores y realizar la consulta de la información y control de los dispositivos.

Investigación bibliográfica, ya que se basó en consultas de revistas técnicas, documentación de librerías y publicaciones en internet y en proyectos de tesis similares referente a temas de API REST, redes de sensores, protocolos de comunicación en IoT, servicios en la nube.

Investigación Experimental porque se realizó una serie de pruebas con los dispositivos IoT y sus respectivos sensores para comunicarse hacia la aplicación mediante MQTT y subir la información hacia la API REST, y gestionar esta información en un cliente HTTP.

2.2.2 Recolección de información

Para lograr la recolección de información se empleó proyectos desarrollados, documentación de librerías de programación online, así como guías prácticas, se tomó en cuenta base de datos confiables que permitieron el desarrollo del proyecto.

No se recolectó información de fuentes como entrevistas, observaciones, cuestionarios.

2.2.3 Procesamiento y análisis de datos

Para el procesamiento y análisis de datos se realizaron los siguientes pasos:

- Revisión de la información recopilada.
- Estudio de las tecnologías usadas para el desarrollo de una API REST
- Estudio de Protocolo MQTT, API REST, y demás para la comunicación de redes de sensores, bróker MQTT, servidor backend, base de datos y formas presentar la información al cliente
- Interpretación de la información relevante que contribuya al desarrollo de la propuesta de solución.

2.2.4 Desarrollo del proyecto

La presente investigación se desarrolló con base en las siguientes actividades:

- Obtención de información sobre el desarrollo de API REST, que tecnologías actuales se usan, como funcionan, que beneficios tienen y una breve comparación entre ellas.
- Investigación de cómo se monitorea las redes de sensores IoT.
- Determinación de las tecnologías a implementar para el desarrollo de la API REST.
- Investigación de un broker MQTT adecuado de código libre que tenga opciones de configuración para comunicación con la aplicación y las redes de sensores.
- Inclusión del bróker MQTT seleccionado a la aplicación, instalación de librerías y frameworks necesarios.

- Programación de la API REST.
- Agregación de seguridad de los usuarios y dispositivos mediante tokens.
- Selección de una base de datos adecuada para almacenar la información de las redes de sensores IoT.
- Despliegue de la API REST en un servicio en la nube que permita la publicación del proyecto.
- Programación de microcontroladores y placa de ordenador reducida (Raspberry PI) adecuados para la conexión MQTT, y API REST con el servidor.
- Realización de pruebas de conectividad de microcontroladores y placa de ordenador reducida con la aplicación.
- Visualización de la información de los sensores IoT en tres clientes HTTP diferentes.
- Gestión de dispositivos IoT que permitan esta función mediante el protocolo MQTT y API REST.
- Corrección de errores.

CAPITULO III

RESULTADOS Y DISCUSIÓN

3.1 Análisis y discusión de los resultados

El desarrollo de una API REST para la transmisión de información y control de redes de sensores IoT permite a las redes de sensores con un dispositivo o modulo wifi adecuado conectarse a la dirección url de la aplicación y mediante el protocolo MQTT puede publicar información con un formato establecido, y suscribirse al bróker de la plataforma para recibir información o eventos que llaman a las acciones. La aplicación esta alojada en un servicio en la nube donde se puede visualizar información de los sensores siguiendo cierto formato de presentación, la información de los sensores puede ser consumida y usada en cualquier cliente HTTP. Mediante el servicio de API REST que tiene el servidor se puede gestionar la información de las redes de sensores según su dirección URL.

3.1.1 Desarrollo de la propuesta

En este proyecto se desarrolla y despliega en la nube una API REST con los recursos: usuarios, nodos, sensores y datos, esta información se guarda en una base de datos adecuada.

Los usuarios autenticados pueden interactuar con la API mediante métodos de petición HTTP con el formato JSON para crear, leer, actualizar y eliminar información de los recursos.

Las redes de sensores IoT publican información de manera segura hacia el bróker MQTT en la nube y esta información una vez validada se guarda en la base de datos.

Los usuarios pueden configurar los parámetros de los sensores y publicarlos en la API REST en el recurso de los nodos o sensores, estos cambios son publicados hacia los dispositivos IoT y así se pueden controlar a distancia.

En la figura 17 se muestra la estructura general del proyecto.

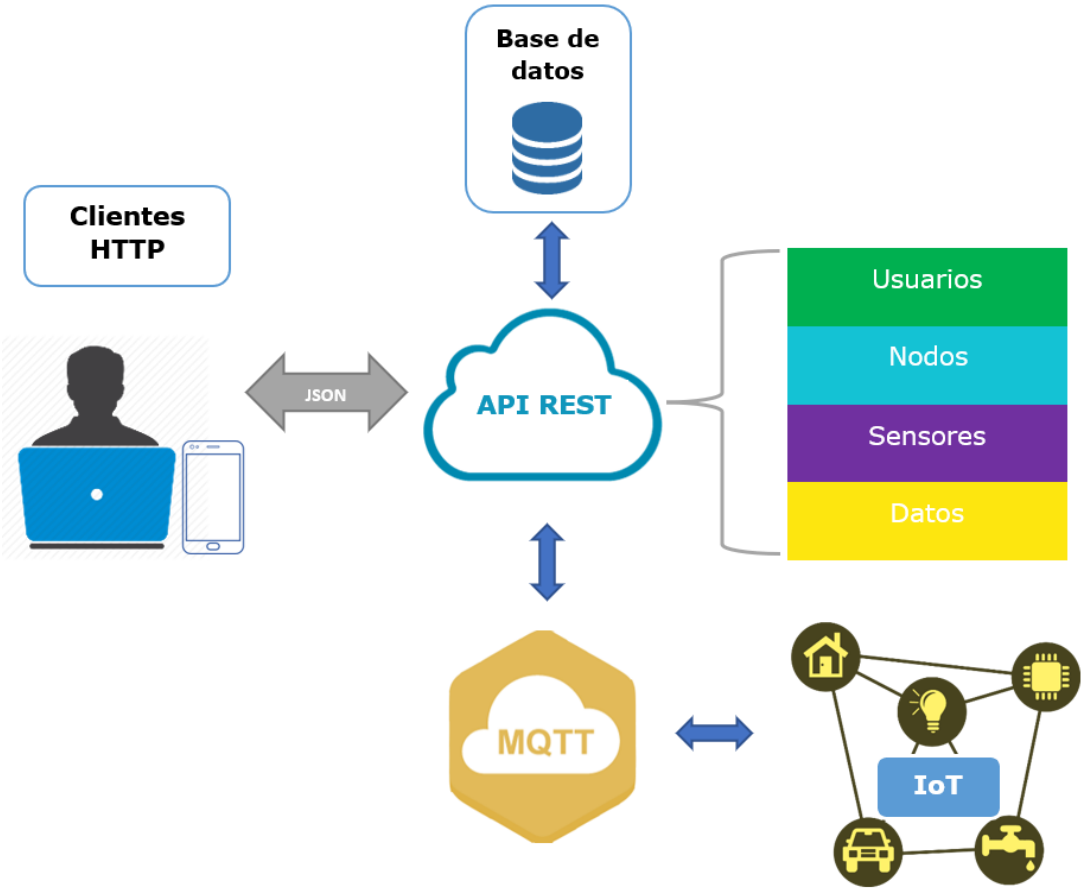


Figura 17 Estructura general del proyecto

Elaborado por el investigador

Selección de software

Entorno para desarrollar la API REST

El entorno de trabajo elegido para la programación de la API REST es Node.js por sus características y ventajas (Ver sección 1.2.2.12) adecuados para aplicaciones del IoT en las que se necesita velocidad, rendimiento, solicitudes simultáneas, peticiones asíncronas y no bloqueantes, además permite crear APIs flexibles desde cero sin una estructura fija.

Node.js se basa en JavaScript por lo cual facilita la programación de aplicaciones de clientes para usar la API REST.

Broker MQTT para el proyecto

Varios proyectos de MQTT ofrecen un bróker MQTT público y en línea para el aprendizaje de MQTT, pruebas, creación de prototipos, estos son de libre acceso, pero tienen retraso en la transmisión de mensajes debido a sus ubicaciones, cargas de trabajo, y demás

Se optó por el uso de un bróker de código libre: EMQ X por sus características (Tabla 3), su documentación es muy completa y tiene versiones para Linux, Windows y Docker. Tiene muchas opciones de interés para el proyecto como seguridad y rendimiento.

Base de datos

Las bases de datos son fundamentales para la persistencia de los datos, para proyectos de IoT en los últimos años se usan bases de datos NoSQL por su alto rendimiento para miles de datos que genera el IoT.

En este proyecto se seleccionó MongoDB por sus ventajas con proyectos de IoT (Sección 1.2.2.14), además que permite la creación de modelos sin estructura fija lo cual es ideal para usuarios que deseen crear nodos, sensores con parámetros distintos y con una variedad de datos que están cambiando continuamente.

Alojamiento en la Nube

En este proyecto se hará uso del modelo serverless para no manejar directamente la infraestructura y obtener todos los beneficios que ofrece este modelo.

Se usó los servicios de la nube de Microsoft Azure con la suscripción de Azure Education con la Universidad Técnica de Ambato para el despliegue de contenedor EMQ X y la aplicación API REST.

Azure Container Instances

Azure Container Instances es una solución para los escenarios que funcionan con contenedores aislados, incluidas las aplicaciones simples, la automatización de tareas y los trabajos de compilación. Se usa este servicio para desplegar el contenedor de bróker MQTT con sus configuraciones.

Azure App Service

Azure App Service es un servicio basado en HTTP para hospedar aplicaciones web, API REST y back-ends, las aplicaciones se ejecutan y escalan fácilmente, provee seguridad, equilibrio de carga y la administración automatizada.

Se uso este servicio para la implementación continua de la API REST

MongoDB Atlas

MongoDB Atlas es un servicio global de base de datos de documentos en la nube para las aplicaciones modernas.

Para almacenar los datos en un servicio en la nube se eligió MongoDB Atlas**,** por su capa gratuita que ofrece los recursos suficientes para guardar la información del proyecto.

A continuación, se puede observar en la figura 18 las tecnologías usadas que contienen los servicios en la nube mencionados

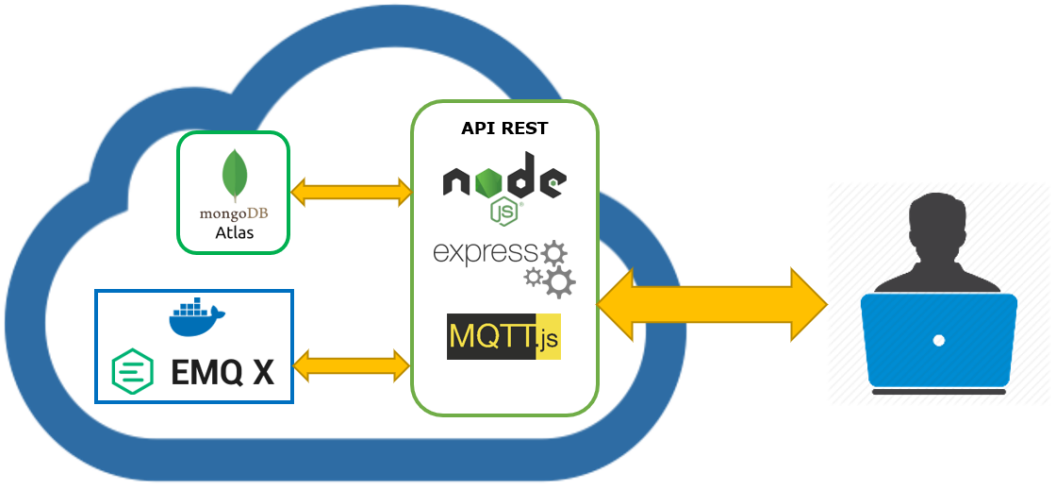


Figura 18 Diagrama general tecnologías usadas

Elaborado por el investigador

Diseño de la API REST

Relaciones de los recursos

Cada recurso (usuarios, nodos, sensores, datos) están relacionados entre sí (figura 19), un usuario puede crear varios nodos, un nodo puede tener varios sensores, un sensor puede tener varios datos. Estas relaciones se conocen como uno a muchos en las bases de datos y se aplica a este contexto. Para crear un nodo se necesita el id del usuario, para crear un sensor se necesita el id del nodo y para publicar datos deben relacionarse a un id de sensor.

En las respuestas JSON se puede observar las relaciones de los recursos.

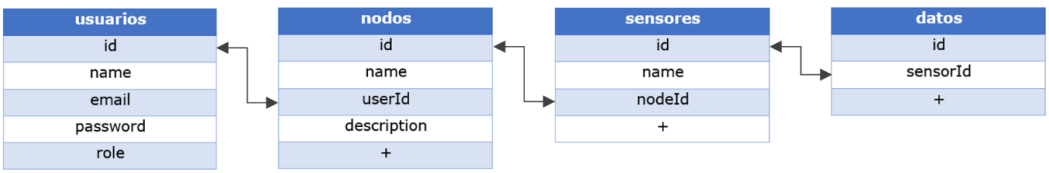


Figura 19 Relaciones de los recursos

Elaborado por el investigador

Ruta de acceso

Esta es la ruta principal a la que se debe acceder primero, el usuario se autentica con su “email” y “password”, la aplicación verifica que el usuario exista en la base de datos y que los campos que envió en la petición POST estén correctos, si es el caso devuelve una respuesta 200 con el usuario y token de acceso para las demás rutas. Ver figura 20.

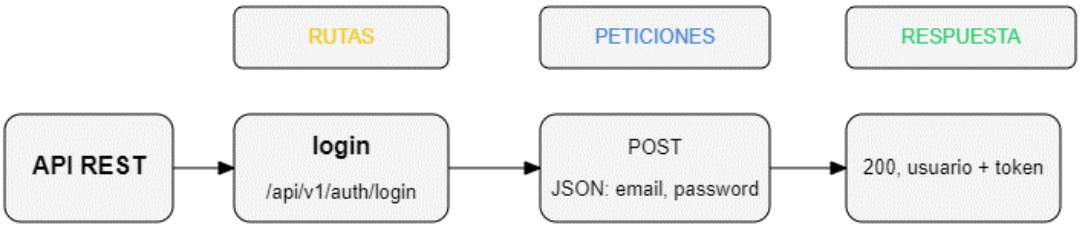


Figura 20 Diseño ruta de acceso

Elaborado por el investigador

Ruta de usuarios

Esta es la ruta de los usuarios, y solo puede ser accedida por el usuario con rol “admin” a excepción de la petición GET {id} que también puede ser solicitada por un usuario con rol “user” para conocer sus propios datos.

Para crear un usuario en la petición POST es necesario agregar en el cuerpo o “body” en formato JSON los campos: “name”, “email”, “password”, “role”.

Para consultar los datos de un usuario en particular se debe especificar su id en la petición GET {id}, como respuesta muestra los datos del usuario sin su contraseña, los nodos, y sensores que ha creado y existen actualmente en la base de datos.

Para la eliminación de un usuario se necesita su Id; los nodos, sensores, datos de sensor que estén asociados a este usuario seguirán vigentes en la base de datos.

En la figura 21 se puede ver el diseño de la ruta usuarios.

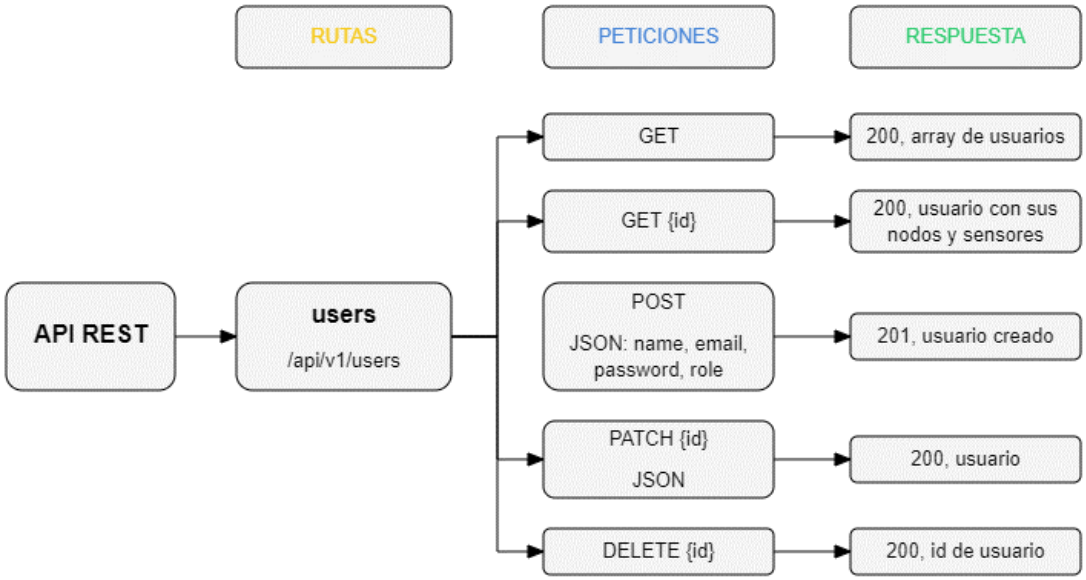


Figura 21 Diseño ruta de usuarios

Elaborado por el investigador

Ruta de nodos

Esta es la ruta de los nodos de las redes de sensores, se puede consultar todos los nodos en la base de datos con la petición GET, esta ruta puede ser accedida solo por el usuario con rol “admin”.

Para consultar un nodo en específico se necesita su id en la petición GET {id}, puede ser accedida por cualquier usuario para ver información de sus nodos e información adicional como los sensores asociados al nodo.

Para crear un nodo en la petición POST es necesario agregar en el cuerpo o “body” en formato JSON los campos: “name”, “userId”, “description”, se pueden agregar más campos.

Se puede actualizar y agregar campos del nodo en la petición PATCH

Para la eliminación de un nodo se necesita su Id; los sensores, datos de sensor que estén relacionados a este nodo seguirán vigentes en la base de datos.

En la figura 22 se puede ver el diseño de la ruta nodos.

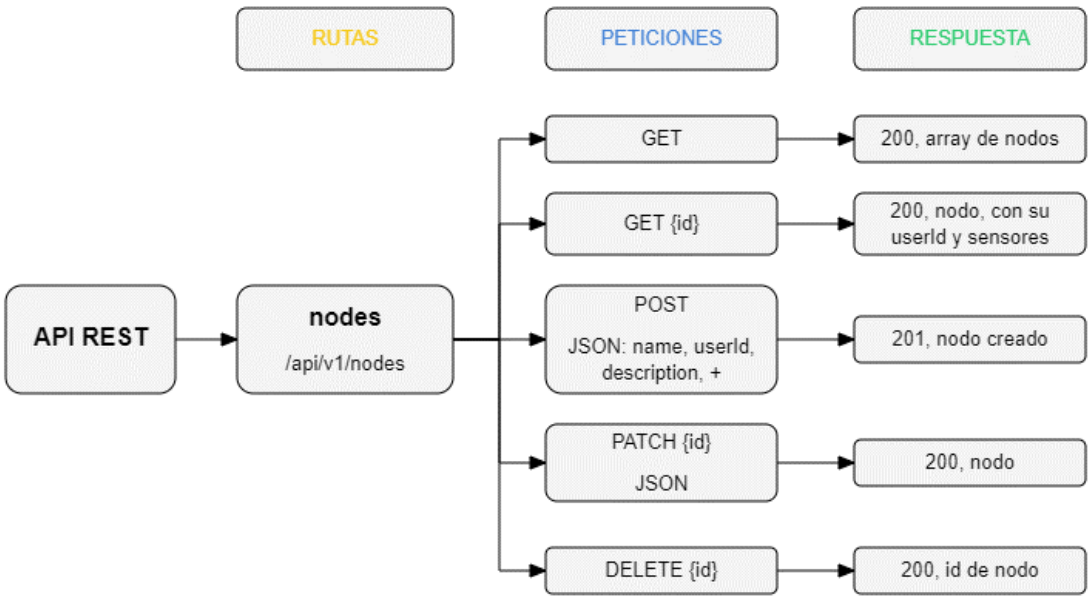


Figura 22 Diseño rutas de nodos

Elaborado por el investigador

Ruta de sensores

En esta ruta podemos obtener todos los sensores de la base de datos y cada uno con más información como el usuario y nodo al que pertenece. Para la creación de un sensor en la petición POST es necesario en formato JSON los campos: nombre y el Id del nodo del cual es parte, además se puede agregar más características y parámetros del sensor sin necesidad de una estructura fija como pueden ser "time_ms", "resolución", etc. Esto facilita la creación de varios sensores de cualquier parámetro según el usuario crea necesario y estos parámetros de configuración se pueden actualizar o agregar mediante la petición PATCH.

Cuando uno o varios parámetros hayan cambiado se dispara un evento en la programación interna para mandar esos cambios mediante una publicación MQTT al tópico **/{sensorId}/config**. El microcontrolador decide si los cambios de un sensor los consigue mediante la suscripción al tópico mencionado o consigue los cambios en una

petición GET al sensor con su Id. Así se logra controlar las redes de sensores a distancia por medio de la API REST y Mqtt.

En este proyecto los cambios de parámetros o configuración de sensor se actualizan en la suscripción al tópico /**{sensorId}/config.** En la figura 23 se puede ver el diseño de la ruta sensores.

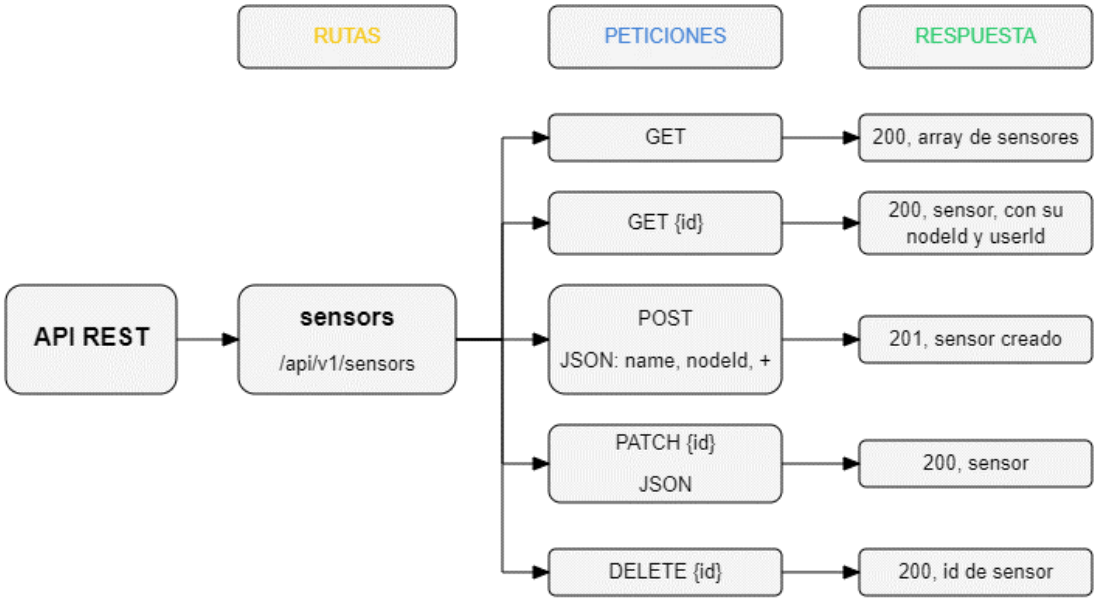


Figura 23 Diseño de ruta de sensores

Elaborado por el investigador

Es posible que se necesite el control de actuadores en las redes de sensores, convencionalmente el inicio o detención de los actuadores corresponden a una programación interna de los dispositivos en las WSN que se desencadenan con ciertas mediciones de los sensores, hay casos en los que el usuario desee controlar los actuadores manualmente mediante parámetros que pueda enviar, en este proyecto también se pueden agregar parámetros de configuración de los actuadores asociados a un sensor, basta con enviar configuraciones en formato JSON en los campos de un sensor o nodo. Los microcontroladores pueden conseguir estos parámetros en el tópico /**{sensorId}/config** o /**{nodeId}/config.**

En este proyecto los cambios de parámetros o configuración de actuadores asociados a un sensor se actualizan en la suscripción al tópico /**{sensorId}/config**

Ruta de datos por sensor

En esta ruta se pueden consultar los datos por id de sensor, y no todos los datos como con los recursos de las rutas anteriores ya que se necesita un orden en los datos y no ver datos al azar de múltiples sensores.

Esta es la ruta más importante del proyecto ya que la finalidad es transmitir los datos de las WSN hacia la aplicación y que se guarden en la base de datos para su posterior uso.

Los datos ordenados por id de sensor son presentados en un array, en las primeras posiciones están los datos más actuales.

Por defecto la consulta de datos con GET tiene un límite de datos a presentar, para no presentar todos los datos que pueden ser miles y sobrecargar el servidor, se puede manejar este límite con la query "limit". Con "offset" se puede señalar la posición desde la que se obtienen los datos, por defecto es cero, y con la query date se elige los datos desde la fecha que se desean obtener los datos hasta la actualidad.

Se pueden publicar datos con el método POST, esta ruta y método sirvió para el desarrollo y pruebas de la API, se pueden publicar los datos con este método, pero en este proyecto las redes de sensores publican los datos al tópico **/{sensorId}/data**, en la programación interna de la API existe un archivo que se suscribe a este tópico, valida los datos y los guarda en la base de datos.

No se consideró usar el método PATCH ya que si existe una medición errónea se la debe eliminar.

Se pueden eliminar varios datos conociendo el id del sensor al que pertenecen, se puede señalar el número de datos a eliminar con "limit" por defecto es uno. Se eliminan los datos más antiguos.

Finalmente existe la ruta de los datos llamada "data" para eliminar datos conociendo su id. En la figura 24 se puede ver el diseño de la ruta de datos por sensor.

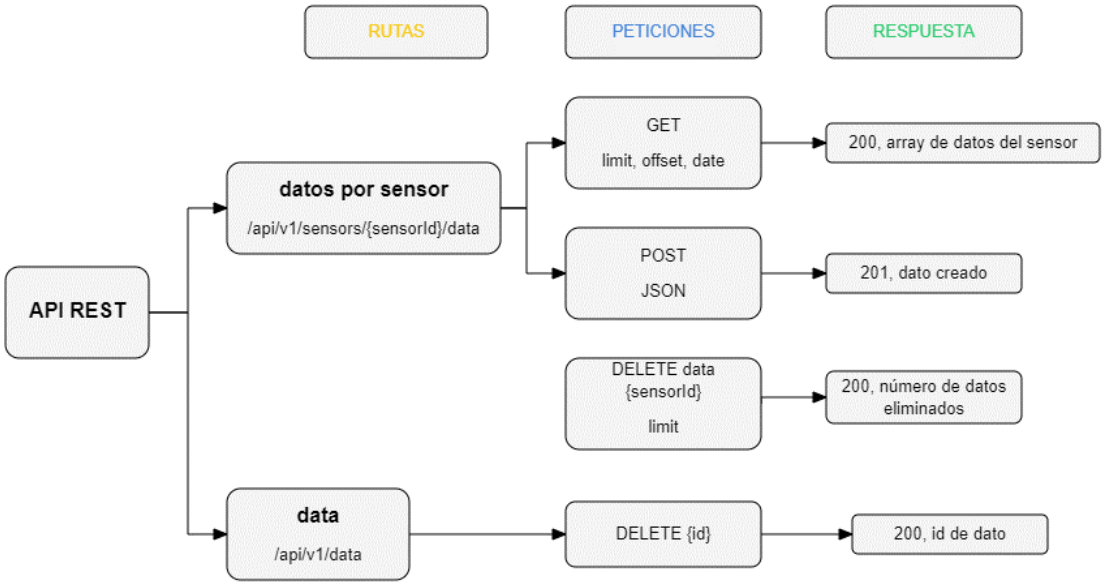


Figura 24 Diseño de ruta de datos por sensor

Elaborado por el investigador

Para todas las rutas si ocurre algún problema, el servidor contesta con un código de estado: 400, 401, 500, según sea el caso.

Programación de la API REST

Para comenzar a programar la API REST en la fase de desarrollo es necesario tener instalado node.js para lo cual se puede revisar el ANEXO A que indica todas las instalaciones de software necesario.

En el directorio del proyecto se ejecutan todos los comandos en la terminal.

Para comenzar el proyecto con node.js se utiliza el siguiente comando:

npm init

Crea un archivo package.json el cual tiene los scripts de inicio y librerías con su versión

Para instalar las librerías se utiliza npm, las librerías se instalan a medida que se necesite en el avance del proyecto, un ejemplo del comando es el siguiente:

npm install express

Al ejecutar por primera vez el comando **npm install** crea una carpeta "node_modules" en el cual se ubican todos los archivos, carpetas y dependencias de las librerías. También crea un archivo package-lock.json el cual no se debe modificar.

Tabla 8 Librerías necesarias para el desarrollo de la API

Librerías	Descripción
bcrypt	Realiza hash de contraseñas, útil para las contraseñas de los usuarios
boom	Objetos de error compatibles con HTTP
cors	Habilita cors, tiene varias opciones
dotenv	Sirve para cargar y gestionar variables de entorno
express	Es el framework más popular de Node.js, permite estructurar una aplicación de una manera ágil, nos proporciona funcionalidades como el enrutamiento, opciones para gestionar sesiones, parámetros url, etc
joi	Sirve para definir esquemas de recursos, validación de datos
jsonwebtoken	Librería de javascript para la creación de JWT, manipulación, validación de tokens
mongoose	Herramienta para el modelado de objetos de MongoDB, trabaja en un entorno asíncrono
morgan	Middleware para el registro de solicitudes HTTP para node.js
mqtt	Cliente para el protocolo MQTT para node.js

nodemon	Es una herramienta que ayuda en el desarrollo al reiniciar automáticamente la aplicación cuando se detectan cambios. Es una dependencia de desarrollo
passport	Middleware de autenticación compatible con Express, autentica solicitudes.
passport-jwt	Estrategia de passport para la autenticación con JWT
passport-local	Estrategia de passport para la autenticación con usuario y contraseña
swagger-jsdoc	Librería para leer el código JSDoc y generar una especificación OpenAPI (Swagger)
swagger-ui-express	Permite servir la documentación API generados con express.

Elaborado por el investigador

Estructura de carpetas y archivos

En la figura 25 se puede observar la estructura de las carpetas y archivos de la API REST, se crea esta estructura para organizar cada parte del código y lograr un código que se entienda y mantenga mejor.

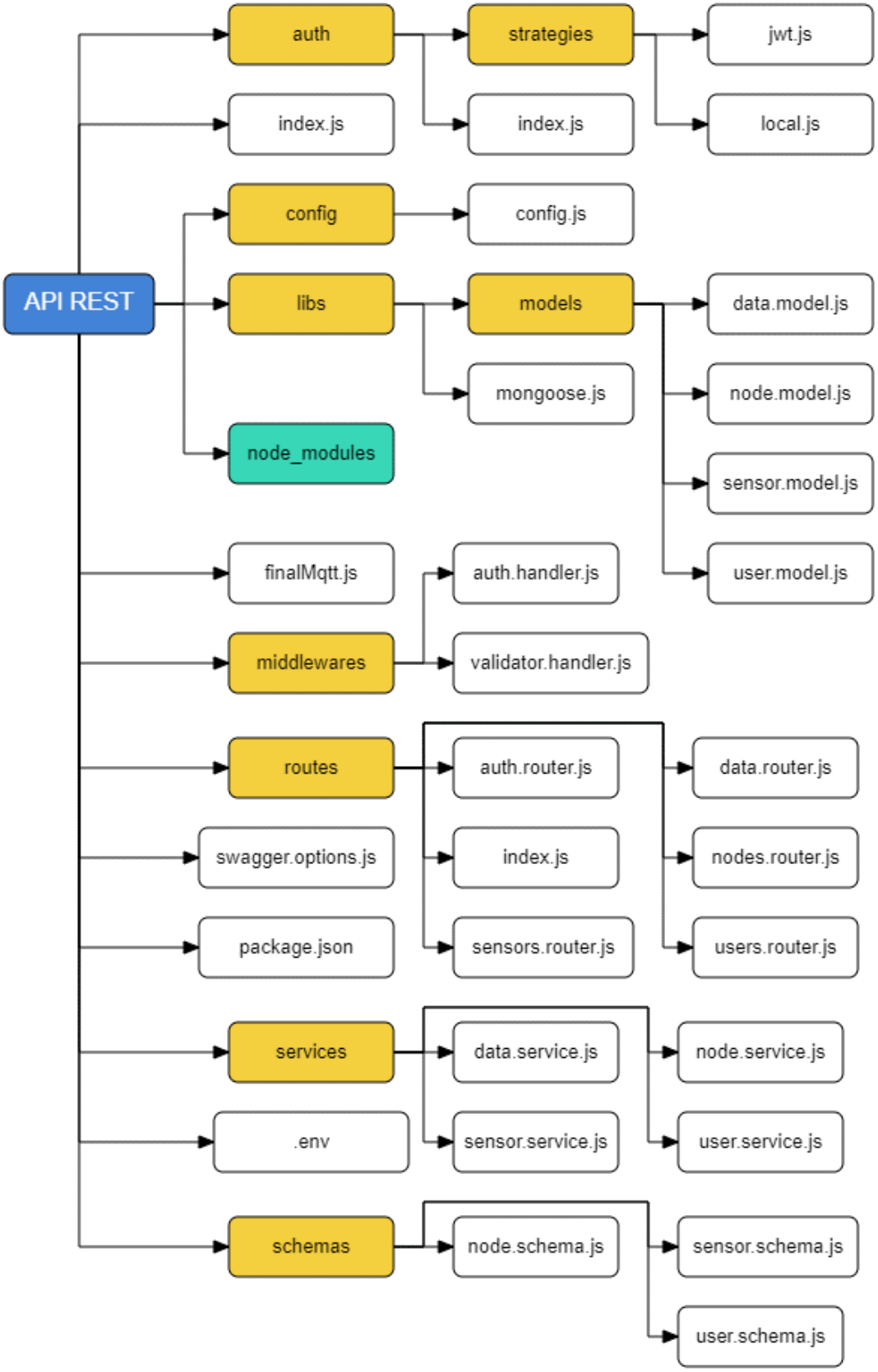


Figura 25 Estructura de los archivos del proyecto

Elaborado por el investigador

Código fuente

index.js

Este archivo es el que se ejecuta para que toda la API REST funcione, mediante el siguiente comando:

node index.js

Para el desarrollo continuo de la aplicación se usó

nodemon index.js

El código de este archivo requiere las siguientes dependencias, ver Tabla 8

```
const express = require('express');
const morgan = require('morgan');
const routerApi = require('./routes');
const cors = require('cors');
```

En routerApi se llama a todas las rutas de la API contenidas en routes/index.js

Para la documentación de la API se utiliza swagger.

Hace un llamado al archivo finaMqtt

```
//swagger documentación
const swaggerUi = require('swagger-ui-express');
const swaggerJsDoc = require('swagger-jsdoc');
const { options } = require('./swagger.options');

require('./finalMqtt');
```

Se crea la aplicación con el método **express()**

El puerto en el que funciona la aplicación en la fase de desarrollo es el 3000, si no está declarado un puerto en las variables de entorno.

```
const app = express();
const port = process.env.PORT || 3000;
```

Se usa el middleware de morgan para obtener detalles de las solicitudes

express.json() es una función de middleware integrada en express, para las solicitudes con cargas útiles JSON.

El Intercambio de Recursos de Origen Cruzado (CORS) es un mecanismo que utiliza cabeceras HTTP adicionales para permitir acceder a los recursos de un servidor de un origen distinto, se utiliza la librería cors para permitir solicitudes de otros orígenes

```
//middlewares
app.use(morgan(':method -- :res[content-length] - :response-time ms'))
app.use(express.json());

app.use(cors());
require('./auth'); //para que passport funcione

const specs = swaggerJsDoc(options);
```

La primera ruta con el método GET en el directorio raíz tiene esta respuesta.

```
app.get('/' , (req, res) =>{
  res.send("Hola mi servidor en Express, autor: Giancarlo Culcay");
});

routerApi(app);
app.use('/docs', swaggerUi.serve, swaggerUi.setup(specs));
```

La aplicación escucha en el puerto ya definido.

```
app.listen(port, () => {
  console.log("My port: " + port)
});
```

Carpeta routes

En esta carpeta se definen las rutas que tiene la API, para acceder a todas las rutas es necesario tener un token de acceso que es otorgado cuando un usuario se autentica con su “email” y “password”, existen usuarios con dos tipos de rol: “admin” y “user”.

El usuario con rol “admin” es capaz de acceder a todas las rutas, puede realizar todas las operaciones CRUD (Create, Read, Update, Delete), en todas las rutas. Puede acceder a la información de otros usuarios si posee el id.

El usuario con rol “user” solo puede acceder a su información en la ruta user, en las demás rutas de nodos, sensores y datos puede realizar todas las operaciones CRUD.

El archivo que contiene todas las rutas es el routes/index.js, todas las rutas empiezan con “/api/v1/” y usan el router de express.

/routes/index.js

```
const express = require('express');

const usersRouter = require('./users.router');
const nodesRouter = require('./nodes.router');
const sensorsRouter = require('./sensors.router');
const dataRouter = require('./data.router');
const authRouter = require('./auth.router');

function routerApi(app){
  const router = express.Router()

  app.use('/api/v1', router);
  router.use('/users', usersRouter);
  router.use('/nodes', nodesRouter);
  router.use('/sensors', sensorsRouter);
  router.use('/data', dataRouter);
  router.use('/auth', authRouter);
}

module.exports = routerApi;
```

auth.router.js

Este archivo se encarga de la ruta de autenticación de los usuarios, usa una estrategia local para el acceso de los usuarios, si el usuario llena los campos “email” y “password” con los datos correctos se le otorga un token de acceso para las demás rutas que expira en 7 días.

```
router.post('/login',
  passport.authenticate('local', { session: false }),
  async (req, res, next) => {
    try {
      const user = req.user;
      const payload = {
        role: user.role,
        sub: user._id
      }
      const token = jwt.sign(payload, config.jwtSecret, {expiresIn: '7d'});
      res.json({
        user,
        token
      });
    } catch (error) {
      next(error);
    }
});

module.exports = router;
```

users.router.js

Este archivo contiene las peticiones que se pueden realizar a la ruta **“**/api/v1/users”

Para users.router.js y las demás rutas se utiliza la autenticación de JWT (Json Web Token) y no sesiones locales, una vez que se autentica el usuario se verifica que rol tiene, si tiene el rol “admin” puede seguir con el método de petición GET para obtener todos los usuarios. Sino se especifica una respuesta de estado, contesta con 200 por defecto.

```
router.get('/',
  passport.authenticate('jwt', { session: false }),
  checkRoles('admin'),
  async (req, res, next) => {
    try {
      const users = await service.find();
      res.json(users);
    } catch (error) {
      next(error);
    }
  }
);
```

Para la petición POST que corresponde a la creación de un usuario, es necesario el cuerpo o "body" de la petición que tiene como campos obligatorios: "name", "email", "password", "role", estos en formato JSON, se valida mediante la función **validatorHandler** que tiene como parámetros el esquema y las propiedades.

Si los campos tienen la estructura correcta se procede a crear un usuario con los datos del cuerpo, se utiliza la función asíncrona **create** que es parte del objeto **UserService.**

Si el usuario es creado, el servidor contesta con un código de estado 201 que significa que la solicitud ha tenido éxito y se ha creado un nuevo recurso, además responde con un JSON del usuario creado con sus campos sin incluir la contraseña.

```
router.post('/',
  passport.authenticate('jwt', { session: false }),
  checkRoles('admin'),
  validatorHandler(createUserSchema, 'body'),
  async (req, res, next) => {
    try {
      const body = req.body;
      const newUser = await service.create(body);
      res.status(201).json(newUser);
    } catch (error) {
      next(error);
    }
  }
);
```

Se sigue las mismas prácticas y funciones para las peticiones PATCH y DELETE, en este proyecto no se realiza la petición PUT ya que se considera que PATCH es más adecuado ya que no elimina datos, solo los actualiza.

Para las peticiones PATCH y DELETE se requiere el id del usuario que es un requisito en los parámetros de la petición.

nodes.router.js

Este archivo contiene las peticiones que se pueden realizar a la ruta **“**/api/v1/nodes”.

Solo el usuario con role “admin” puede realizar la petición GET, para obtener todos los nodos creados.

```
router.get('/',
  passport.authenticate('jwt', { session: false }),
  checkRoles('admin'),
  async (req, res, next) => {
    try {
      const nodes = await service.find();
      res.json(nodes);
    } catch (error) {
      next(error);
    }
  }
);
```

Para obtener un nodo en específico se debe indicar su id en los parámetros de la petición,

```
router.get('/:id',
  passport.authenticate('jwt', { session: false }),
  checkRoles('admin','user'),
  validatorHandler(getNodeSchema, 'params'),
  async (req, res, next) => {
    try {
      const { id } = req.params;
      const node = await service.findById(id);
      res.json(node);
    } catch (error) {
      next(error);
    }
  }
);
```

Todas las demás peticiones para la ruta de /api/v1/nodes requieren un usuario con role “admin” o “user” y siguen las mismas prácticas y usan funciones similares.

sensors.router.js

Se utiliza la misma lógica para los métodos GET, POST, PATCH, DELETE en la ruta de los sensores

Para obtener los datos por sensor se necesita el id del sensor, en la ruta **/api/v1/sensors/{sensorId}/data**

Para la petición GET se valida el token, el rol del usuario, el id del sensor. En esta ruta se puede poner, en la parte de la dirección url parámetros de consulta como **limit, offset** y **date**, corresponden a la “query”. Limit y offset tienen valores por defecto si no se les indica en la query, de 10 y 0 respectivamente.

Estos valores se los transforma a enteros y se envía a la función **getDataBySensorId,** el campo date puede venir o no en la query y no tienen un valor por defecto.

```
router.get('/:sensorId/data',
  passport.authenticate('jwt', { session: false }),
  checkRoles('admin','user'),
  validatorHandler(getSensorSchema, 'params'),
  async (req, res, next) => {
    try {
      const { sensorId } = req.params;
      const limit = parseInt(req.query.limit) || 10;
      const offset = parseInt(req.query.offset) || 0;
      const { date } = req.query;

      const data = await service2.getDataBySensorId(sensorId, limit, offset, date);
      res.json(data);
    } catch (error) {
      next(error);
    }
  }
);
```

Para el método de petición POST se puede crear datos a partir del id del sensor, este método se creó con fines de desarrollo los datos de los microcontroladores se publican con el protocolo MQTT y la aplicación guarda los datos en la base de datos de MongoDB.

```
router.post('/:sensorId/data',
  passport.authenticate('jwt', { session: false }),
  checkRoles('admin','user'),
  validatorHandler(getSensorSchema, 'params'),
  async (req, res, next) => {
    try {
      const { sensorId } = req.params; //sensor id
      const body = req.body;
      const newData = await service2.createDataBySensorId(sensorId, body);
      res.status(201).json(newData);
    } catch (error) {
      next(error);
    }
  }
);
```

Para eliminar datos de un sensor sirve el método DELETE en la ruta **“api/v1/sensors/{sensorId}/data”,** por defecto se borra un dato y se puede especificar en el campo **limit** cuantos datos serán borrados.

```
router.delete('/:sensorId/data',
  passport.authenticate('jwt', { session: false }),
  checkRoles('admin','user'),
  validatorHandler(getSensorSchema, 'params'),
  async (req, res, next) => {
    try {
      const { sensorId } = req.params;
      const limit = parseInt(req.query.limit) || 1; //por defecto borra 1 dato
      const data = await service2.deleteManyBySensorId(sensorId, limit);
      res.status(201).json(data);
    } catch (error) {
      next(error);
    }
  }
);
```

data.router.js

En esta ruta solo se agregó el método DELETE para borrar un dato en específico conociendo su id.

```
router.delete('/:id',
  passport.authenticate('jwt', { session: false }),
  checkRoles('admin'),

  async (req, res, next) => {
    try {
      const { id } = req.params;
      await service.deleteDataById(id);
      res.status(200).json({id});
    } catch (error) {
      next(error);
    }
  }
);
```

Carpeta libs

En esta carpeta están los archivos para la conexión con la base de datos, y la carpeta models

mongoose.js

```
//conexion:
const mongoose = require('mongoose');
const { config } = require('../config/config');

const URI = config.URI

mongoose.connect( URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const db = mongoose.connection;

db.once('open', function () {
  console.log('[db] Conectada con éxito');
});

db.on('error', (err) => {
  console.log('[db] Error de conexión', err);
});

module.exports = db;
```

Para la conexión con la base de datos de MongoDB se utiliza una función de la librería mongoose, el URI corresponde a la cadena de string de conexión que provee mongo, en el desarrollo no se especificó usuario y contraseña (ANEXO C), pero si existen estos campos para la conexión con MongoDB Atlas, y este ya ofrece el string de conexión. (ANEXO E y F)

En la consola se puede ver si la conexión se realizó correctamente o no figura 26

Figura 26 Respuesta de consola de conexión exitosa con la base de datos

Carpeta models

En esta carpeta están los modelos de los recursos (usuarios, nodos, sensores, datos)

user.model.js

```
const UserSchema = new Schema(
  {
    name: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      unique: true,
      required: true,
    },
    password: {
      type: String,
      required: true,
    },
    enabled: {
      type: Boolean,
      default: true,
    },
    role: {
      type: String,
      default: "user",
    },
  },
  {
    versionKey: false,
    timestamps: true,
    id: false
  }
);
```

Mediante los Schema que provee mongoose se crea el esquema para los usuarios con los campos “name”, “email”, “password”, “enabled”, “role”. El campo “enabled” por defecto es verdadero ya que al crear un usuario por defecto este ya este habilitado para crear los demás recursos y usar la aplicación.

- **versionKey:** este campo lo omitimos ya que no sirve en esta versión del proyecto, tiene una salida: “__v”.
- **timestamps:** llena por defecto campos “createdAt” y “updatedAt”.
- **id:** este campo está en falso para que no se imprima en el JSON dos veces el id en el campo virtuales.

```
UserSchema.virtual('nodes', {
  ref: 'Node',
  localField: '_id',
  foreignField: 'userId',
})
```

Se utiliza la función de mongoose “virtual” para las propiedades calculadas de los documentos.

Para los modelos: node.model.js, sensor.model.js, se utilizan esquemas similares al anterior para cada recurso, corresponden a los campos ya demostrados en la sección Diseño de la API REST, en estos modelos existe un campo particular:

```
{
  versionKey: false,
  timestamps: true,
  strict: false
}
```

- **strict: false** esta opción está habilitada por defecto, con false permite que se puedan agregar varios valores al constructor de modelos que no se especificaron en el esquema y se guarden en la base de datos, así se pueden agregar más valores a un nodo, sensor y datos.

Carpeta schemas

En esta carpeta están los archivos de los esquemas de los recursos

user.schema.js

Hace uso de la librería joi, ver Tabla 8, cada valor del recurso usuario como id se le da un formato de texto, tipo, longitud de caracteres. Todo esto con el fin de validar los datos que llegan en las solicitudes.

```
const Joi = require('joi');

const id =   Joi.string().alphanum().length(24);
const name = Joi.string().min(3);
const email = Joi.string().email();
const password = Joi.string().min(5);
const role = Joi.string().min(4).max(5);
```

Para obtener un usuario es necesario su id

```
const getUserSchema = Joi.object({
  id: id.required()
});
```

Para la creación de un usuario se necesita los siguientes valores

```
const createUserSchema = Joi.object({
  name: name.required(),
  email: email.required(),
  password: password.required(),
  role: role.required()
});
```

Se utiliza la misma lógica para los demás recursos y se agrega la opción:

- options({ allowUnknown: true }) : Permite agregar más valores que no pertenezcan al esquema

El recurso de datos no tiene esquema ya que se valida solo el id de sensor y esto se realiza en node.schema.js.

Carpeta services

En esta carpeta están los archivos con objetos y métodos para realizar las acciones o lógica de las rutas.

Se hace uso de la programación orientada a objetos y de los métodos asíncronos.

Al utilizar la base de datos de MongoDB y mongoose se hace uso de los métodos de los modelos y documentos que provee mongoose ver Tabla 9.

Tabla 9 Funciones de mongoose utilizadas

Funciones	Descripción
Model.find()	Busca documentos
Model.findById()	Busca un único documento por su campo de _id
Model.findByIdAndDelete()	Busca un documento por su campo de _id y lo elimina
Model.findByIdAndUpdate()	Busca un documento por su campo de _id y lo actualiza

Model.findOne()	Busca un solo documento
Model.populate()	Rellena las referencias del documento.
Model.prototype.save()	Guarda el documento insertando un nuevo documento en la base de datos
Model()	Genera un nuevo documento de tipo objeto con los valores del esquema

Elaborado por el investigador

user.service.js

En este archivo se encuentra la clase **UserService** que tiene varios métodos asíncronos

```
class UserService {
```

- find(): esta función usa al modelo User para buscar todos los documentos y en la respuesta no traer el valor "password", después esta función retorna el objeto "users".

```
async find() {
  const users = await User.find().select("-password");
  return users;
}
```

- create(data): Esta función usa la librería bcrypt para realizar hash de la contraseña y guardar el documento junto a los demás datos (name, email, role) que están en el parámetro data.

  Se guarda el nuevo usuario y para la respuesta JSON que envía el servidor se cambia el valor de la contraseña.

```
async create(data) {
  const hash = await brcypt.hashSync(data.password, 10);
  const newUser = await User({
    ...data,
    password: hash
  });
  await newUser.save();
  newUser.password = undefined;
  return newUser;
}
```

- findById(data): Esta función usa el parámetro id para buscar el usuario con este id, en la respuesta JSON no incluye la contraseña y rellena los valores de que nodos y que sensores tiene actualmente. En caso que no se encuentro el usuario con el id lanza un error no bloqueante con su mensaje.

```
async findById(id) {
  const user = await User.findById(id)
  .select('-password')
  .populate({
    path: 'nodes',
    select: 'name -userId',
    populate: { path: 'sensors', select: 'name -nodeId' }
  })
  if(!user){
    throw new Error('user not found');
  }
  return user;
}
```

- update(id, changes): esta función sirve para actualizar los valores de un usuario, primero compara la contraseña en el objeto changes, y si coincide con la contraseña encriptada en la base de datos, permite realizar las actualizaciones.
- delete(id): permite eliminar el usuario con su id, hace uso de la función **findByIdAndDelete**

node.service.js

En este archivo se encuentra la clase **NodeService** que tiene al igual que **UserService** tiene los mismos métodos, pero con el modelo "Node" y sin el campo contraseña. (ver Ruta de nodos)

sensor.service.js

En este archivo se encuentra la clase **SensorService** que tiene los mismos métodos que NodeService, con el modelo "Sensor", en el método update() valida los cambios y los publica en el tópico **/{sensorId}/config** (ver Ruta de sensores)

data.service.js

En este archivo se encuentra la clase **DataService** que tiene varios métodos asíncronos

(ver Ruta de datos por sensor)

- getDataBySensorId (sensorId, limit, offset, date)
- createDataBySensorId (sensorId, data)
- deleteManyBySensorId(sensorId, limit)
- deleteDataById(id)

Carpeta middlewares

En esta carpeta se encuentran los middlewares para la validación del role y el esquema de los recursos.

auth.handler.js

Este archivo se encarga de verificar el rol del usuario dado en la solicitud con el array "roles", si lo contiene permite continuar y si no lanza un error de desautorizado.

Hace uso de la función flecha checkRoles()

```
checkRoles = (...roles) => {
  return (req, res, next) => {
    const user = req.user;
    roles.includes(user.role) ? next() : next(boom.unauthorized());
  }
}
```

validator.handler.js

Este archivo valida si los datos ingresados corresponden al esquema del recurso, tiene una lógica similar al anterior archivo

Carpeta auth

En esta carpeta se ubican los archivos para la autenticación de los usuarios

auth/index.js

```
const passport = require("passport");

const LocalStrategy = require("./strategies/local");
const JwtStrategy = require("./strategies/jwt");

passport.use(LocalStrategy);
passport.use(JwtStrategy);
```

Este archivo hace uso de la librería Passport para autenticar las solicitudes y usa dos formas la local y por JWT

Carpeta strategies

En esta carpeta se ubican las estrategias de passport a utilizar

- **jwt.js:** se define el token del usuario en la cabecera y el secreto del token en las variables de entorno por seguridad.
- **local.js:** se define la autenticación local con el email y password

Carpeta config

En esta carpeta se encuentra el archivo config.js

config.js

En este archivo se encuentran las configuraciones internas de la aplicación para las variables de entorno.

.env

En este archivo se encuentran todas las variables de entorno para el usuario, contraseña, host de la base de datos, el secreto de JWT, puerto de la aplicación. Este archivo al contener estos datos sensibles sirve para la fase de desarrollo y no se sube al repositorio de git.

finalMqtt.js

Este archivo usa la librería mqtt.js para conectarse al bróker EMQ X del contenedor requiere un usuario y contraseña según configuración del contenedor. Además, se conecta a la base de datos de MongoDB y usa el modelo de Data

```
const mqtt = require('mqtt');
require('./libs/mongoose'); //conexion
const Data = require('./libs/models/data.model');

const mqttClient = mqtt.connect(`mqtt://${process.env.MQTT_HOST}:$
{process.env.MQTT_PORT}`, {username: process.env.MQTT_USER, password:
process.env.MQTT_PASSWORD});
```

El cliente de MQTT se suscribe a los datos y configuración de los sensores, posteriormente se valida si es un tópico válido.

```
mqttClient.subscribe(['/+/data', '/+/config']);
```

El mensaje que recibe de la suscripción datos se transforma a un string, para guardarse en la base de datos haciendo uso del modelo Data en la función createData()

Si ha ocurrido un error en este proceso se publica en el tópico “data/error”

```
try {
  const m = JSON.parse(message.toString());

  async function createData(payload) {
    payload.sensorId = sensorId;
    const newData = await Data(payload);
    return await newData.save();
  }
  createData(m).then(console.log)
}catch(e){
  mqttClient.publish(`/data/error`,`{"error": "${topic}"`);
  console.log(`error: ${topic}`);
}
```

swagger.options.js

Este archivo provee la configuración de la documentación de swagger

```
const options = {
    definition: {
        openapi: '3.0.0',
        info: {
            title: 'API REST WSN IoT',
            version: '1.0.0',
            description: 'API REST PARA LA TRANSMISIÓN DE INFORMACIÓN Y CONTROL DE REDES DE SENSORES IOT',
            contact: {
                name: 'Giancarlo Culcay',
                email: '[redacted]',
            }
        },
        servers: [
            {
                url: `${URL}/api/v1`,
            }
        ],
    },
    apis: ["./routes/*.js"]
}
```

Lo que nos da como resultado en la ruta "/docs" se puede visualizar en la figura 27.

Figura 27 Portada documentación con swagger

En todos los archivos de las rutas se comentó cada método siguiendo la sintaxis de la documentación de swagger como:

```
/**
 * @swagger
 * tags:
 *  name: Users
 *  description: Users endpoint
 */
```

En el Anexo G se muestra los comentarios con swagger de la ruta usuarios con el método GET.

La documentación de swagger terminada en "/docs" para cada endpoint se puede visualizar en las siguientes figuras.

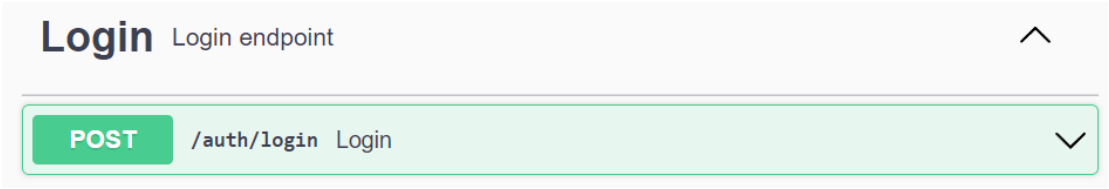


Figura 28 Login endpoint swagger

Datos

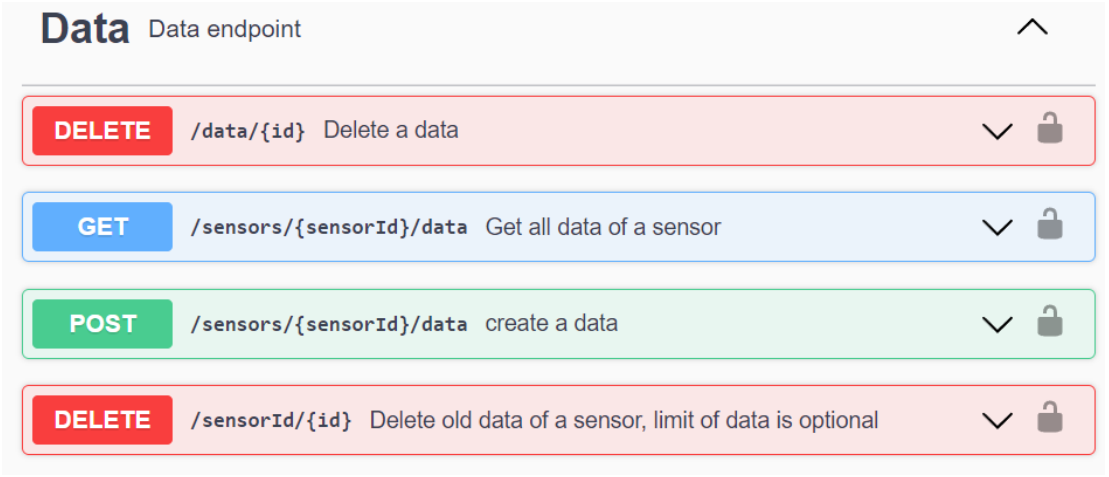


Figura 29 Data endpoint swagger

Nodos

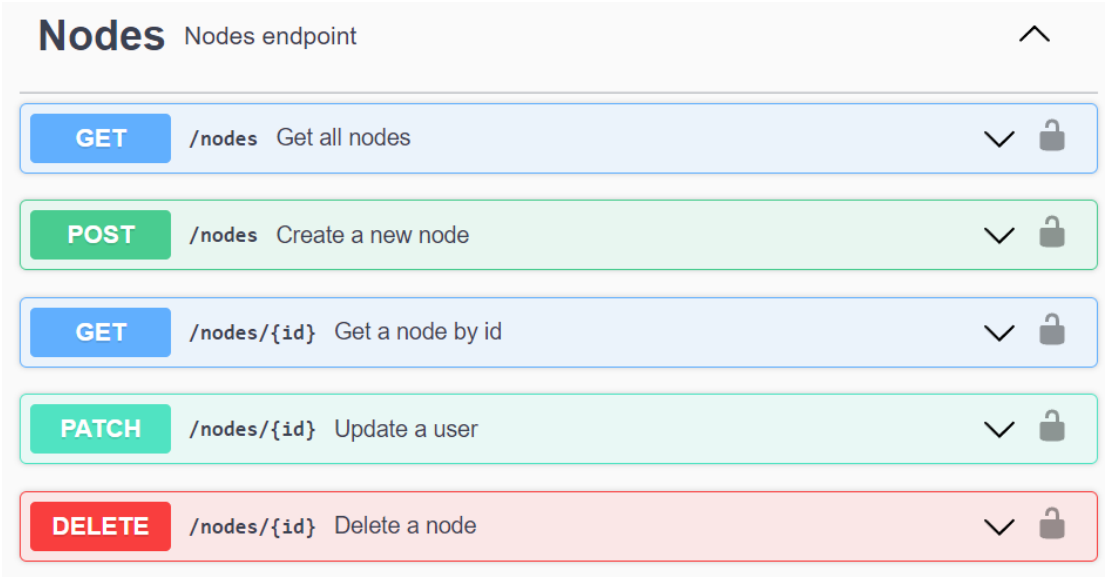


Figura 30 Nodes endpoint swagger

Sensores

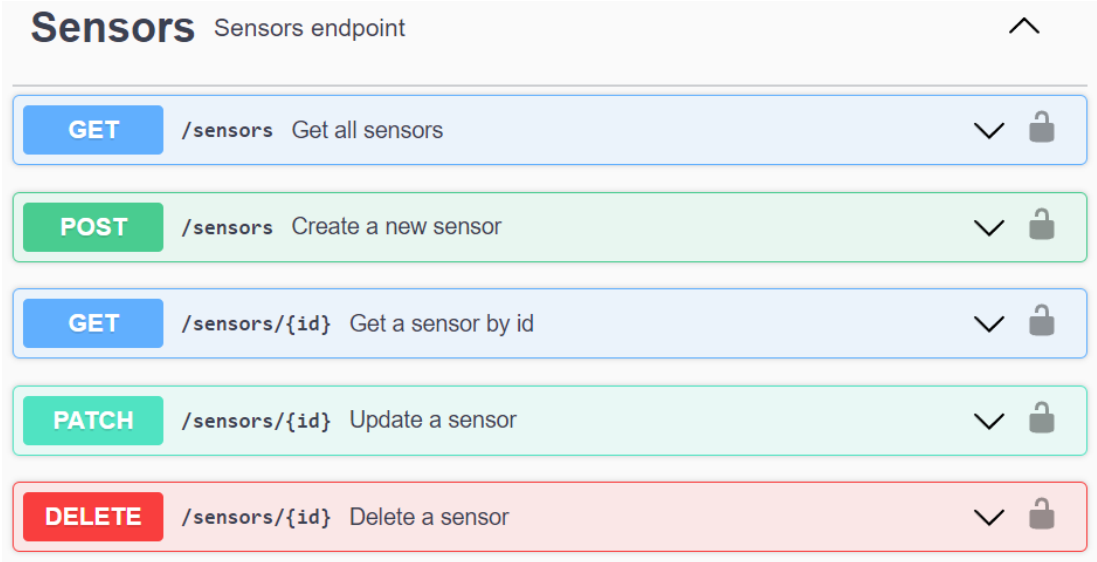


Figura 31 Sensors endpoint swagger

Usuarios

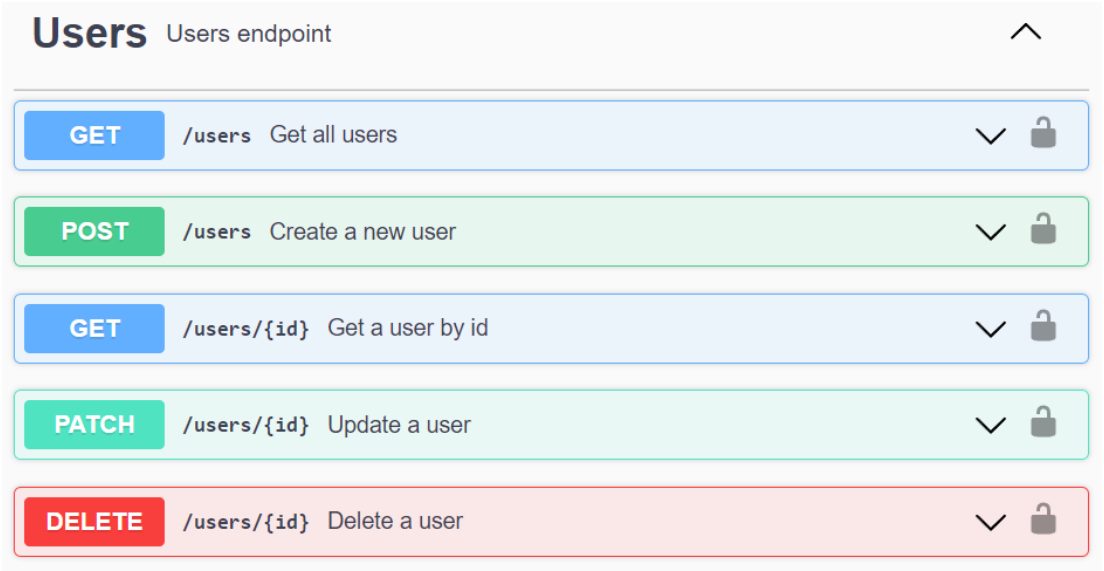


Figura 32 Users endpoint swagger

Conexiones y configuraciones al broker MQTT local

Este proyecto utiliza el broker EMQ X por medio de una imagen de contenedor de docker (ANEXO B).

En la fase de desarrollo se prueba la conectividad local (ANEXO H).

Conexión a la base de datos local

En la fase de desarrollo los datos de las redes de sensores se almacenan en el contenedor de la base de datos de MongoDB para lo cual se levanta este contenedor con volumen compartido (ANEXO B).

Para la conexión se puede revisar el ANEXO C.

Una vez que se ha logrado las conexiones locales entre la base de datos, bróker mqtt y se ha corregido todos los errores que surgieron en la fase de desarrollo, se continua con la fase de producción es decir cuando los servicios están en la nube.

Uso de servicios en la nube de Microsoft Azure

Para usar los servicios de Microsoft Azure es necesario tener una cuenta y una suscripción. Figura 33.

Se usó la suscripción de “Azure for students” que provee la Universidad Técnica de Ambato para los estudiantes, cuenta con todos los servicios de Azure y un crédito de $100 para 12 meses. Lo cual es suficiente para llevar a cabo el proyecto.

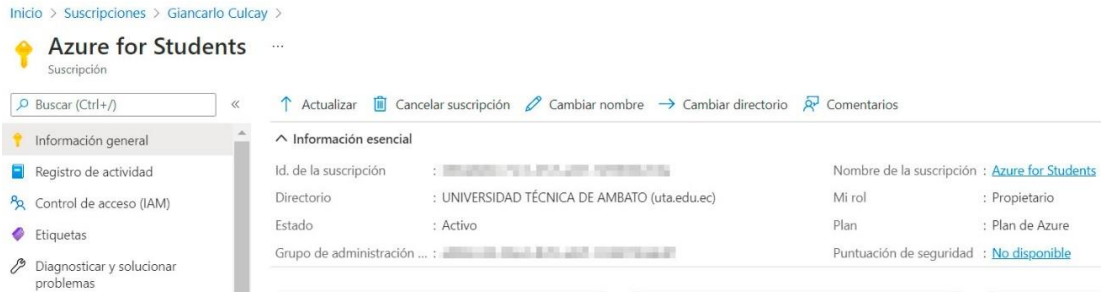


Figura 33 Suscripción de Azure

Es necesario crear un grupo de recursos para implementar y administrar los recursos de Azure, ver figura 34

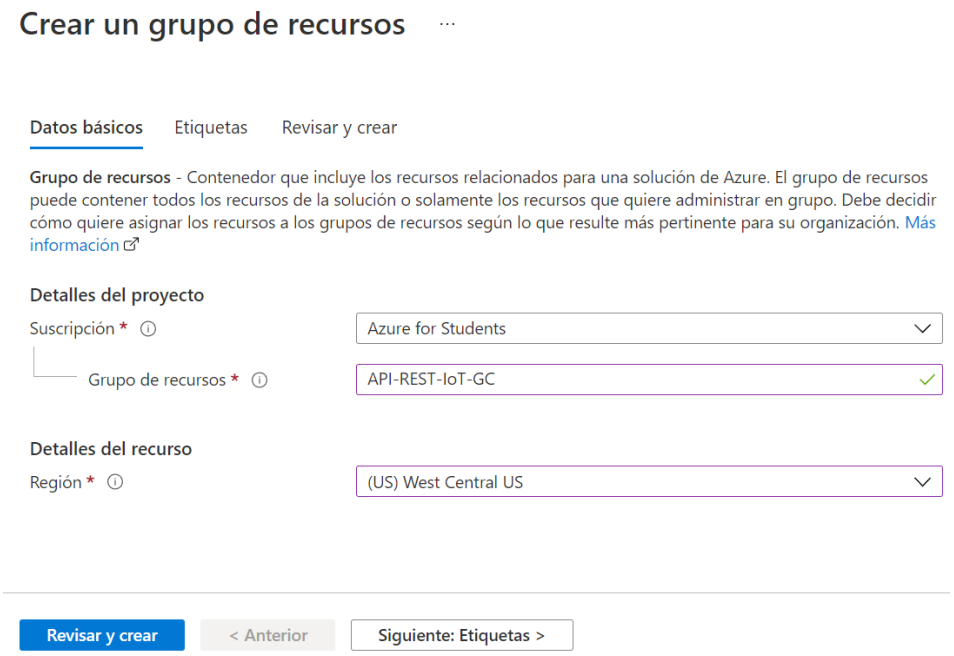


Figura 34 Creación de un grupo de recursos

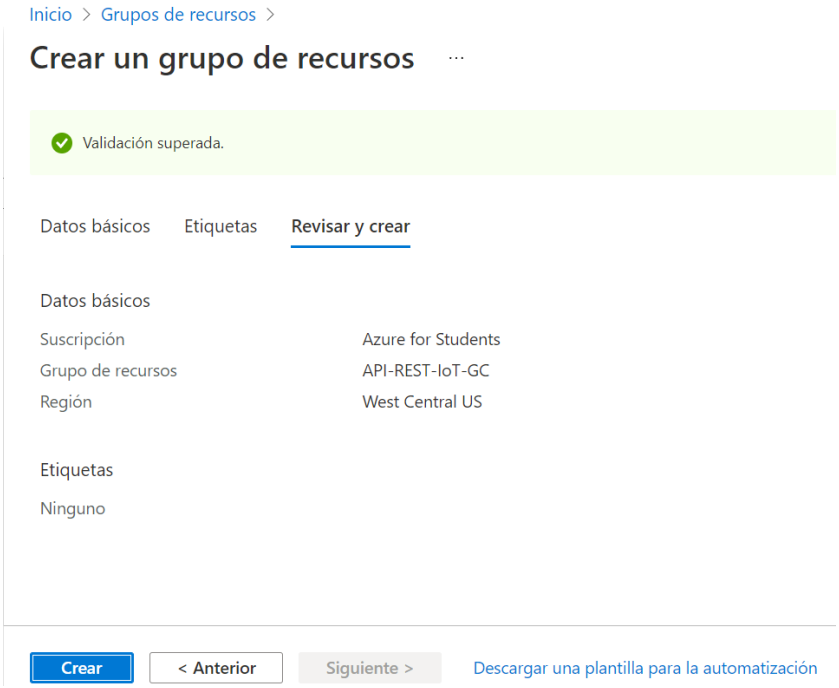


Figura 35 Resumen creación de un grupo de recursos

Una vez creado el grupo de recursos se pueden crear recursos de todo tipo.

Despliegue de contenedor MQTT en la nube

Para desplegar el contenedor de bróker MQTT en a la nube se usó Azure Container Instances, para asignar parámetros del contenedor como puertos, nombre, dns, variables de entorno se usó Azure CLI.

Se debe instalar Azure CLI (ver ANEXO A) e iniciar sesión con el comando:

az login

Se abre una página web para iniciar sesión en la cuenta.

Se despliega un contenedor de Docker con la imagen de bróker MQTT: emqx en la versión 4.3.11 con las siguientes configuraciones y variables de entorno en Azure Container Instances, ver figura 36.

Figura 36 Creación de contenedor mqtt con Azure CLI

Elaborado por el investigador

En este contenedor se cambia la contraseña por defecto y no se permiten las conexiones anónimas para la seguridad de los usuarios, integridad de los datos y garantizar la autenticación mediante JWT.

Los valores por defecto de número de núcleos de cpu del contenedor es: 1 y la memoria es: 1.5GB

Se puede revisar los registros de contenedor, FQDN y visualización del dashboard de EMQ X en el ANEXO I

Despliegue de aplicación API REST

Para desplegar la aplicación desarrollada se usó el servicio de App Services de Azure con los siguientes detalles figura 37

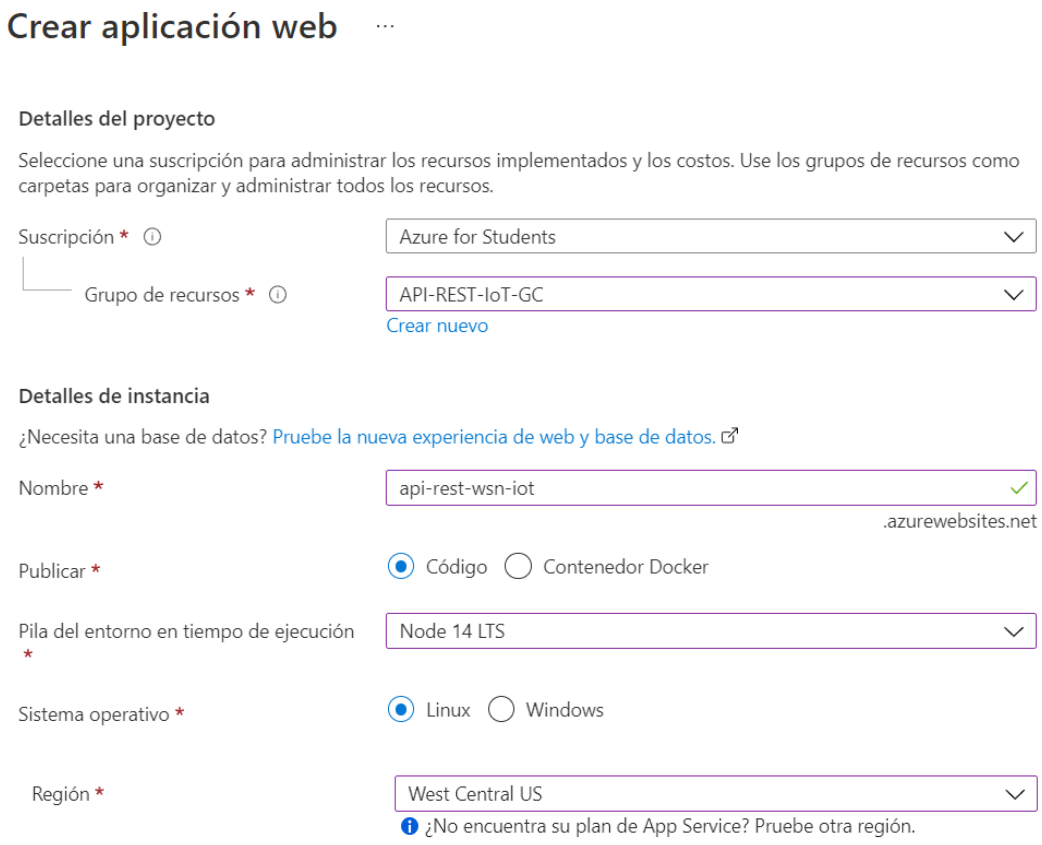


Figura 37 Creación de instancia en App Services

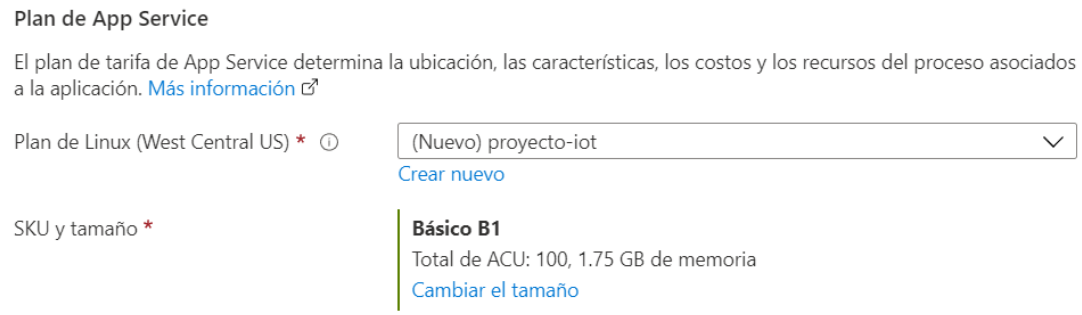


Figura 38 Plan de App Services

El plan de App Services elegido es el Básico B1 para pruebas de desarrollo con un total de ACU:100, 1,75 GB de memoria equivalente de proceso de serie A, existen planes más adecuados para producción cuyo costo es más elevado por mes y supera los créditos restantes de esta suscripción.

Implementación

El proyecto puede presentar cambios por lo que se necesita de una implementación continua (figura 39) y se eligió el método con GitHub Actions, el código se subió al repositorio personal del investigador al proyecto privado API-REST-WSN-IOT (ANEXO J)

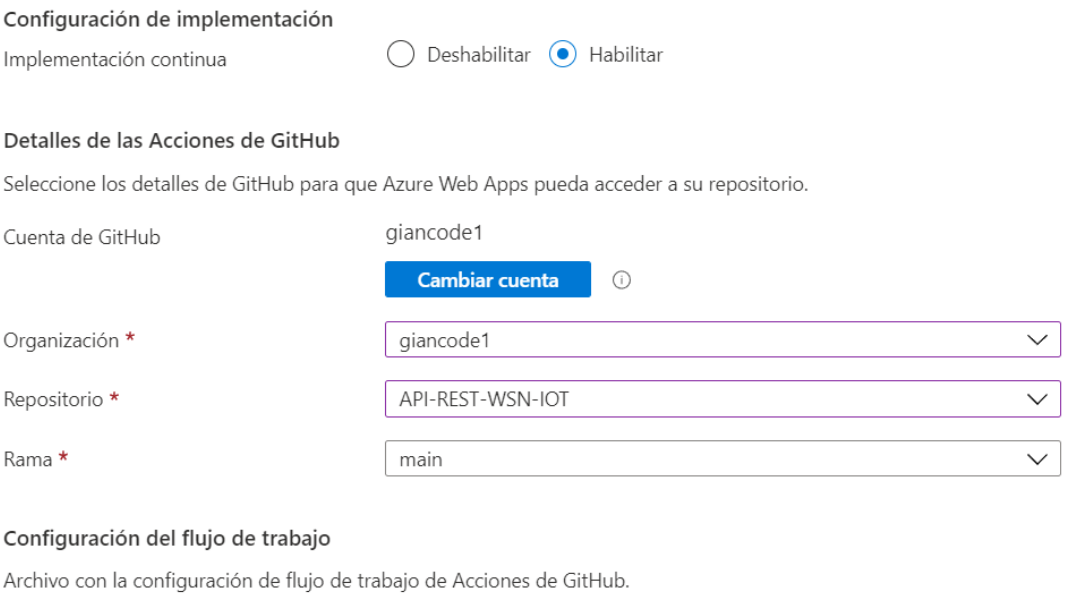


Figura 39 Configuración de la Implementación de la API REST

Cuando se ha terminado las configuraciones y Github Actions ha terminado de desplegar el código figura 40.

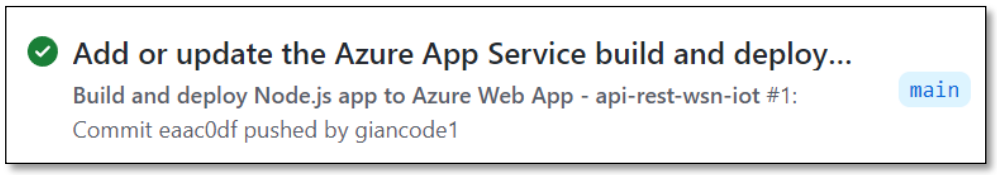


Figura 40 Construcción y despliegue de la aplicación con github Actions

Los detalles de la aplicación en App Service brinda la URL del proyecto figura 41

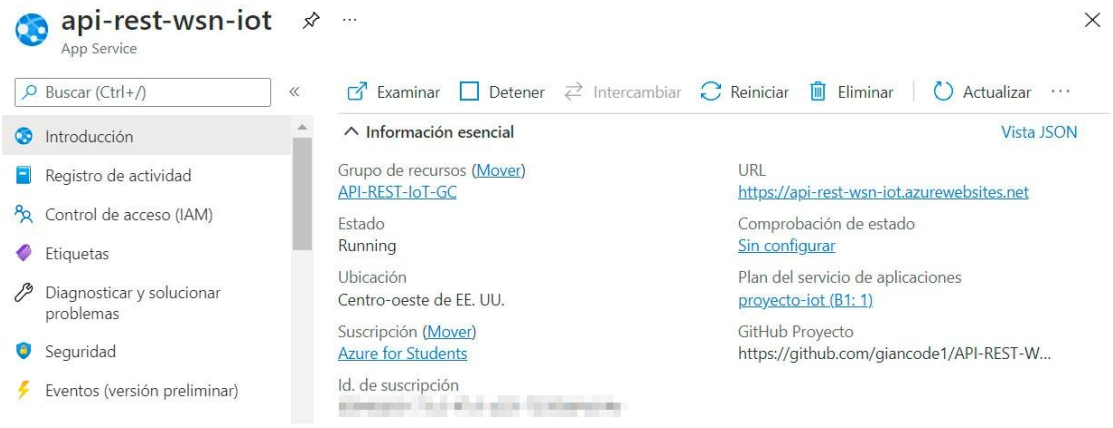


Figura 41 Aplicación en App Service

Los detalles de configuración de la aplicación se encuentran en el ANEXO K

En la ruta raíz "/" se puede visualizar la figura 42, tal y como se realizó en la etapa de desarrollo.

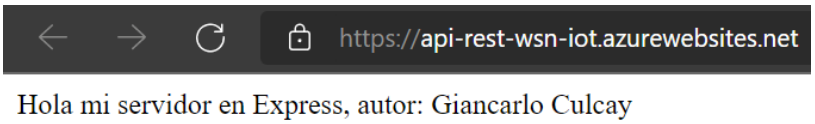


Figura 42 Url de la aplicación

En la figura 43 se puede observar todas las rutas disponibles con la documentación de swagger y la dirección url que provee App Service.

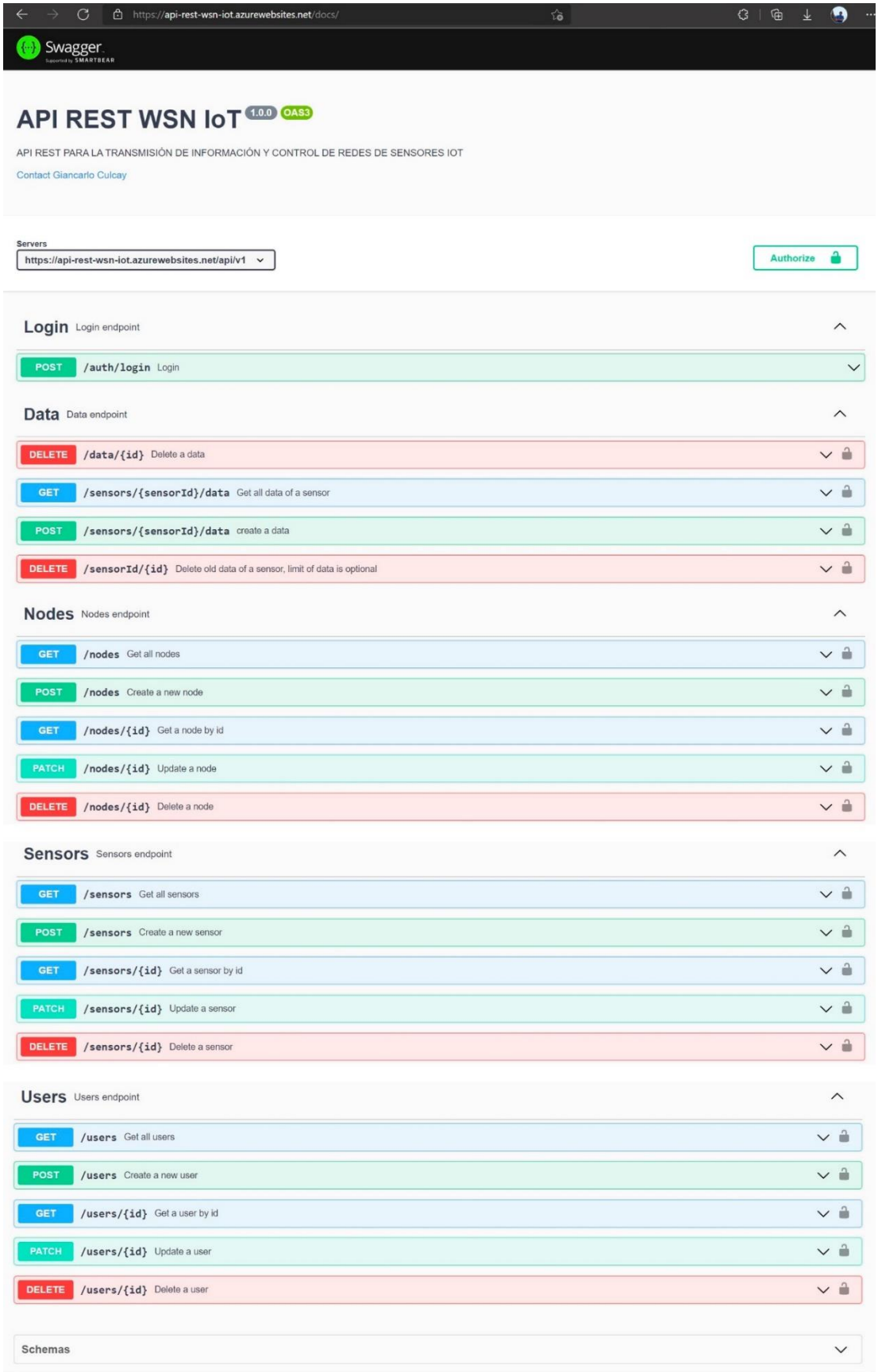


Figura 43 Documentación swagger de la aplicación en la nube

Se procede a la creación del primer usuario con role “admin” a la API REST en la nube el cual es el primer paso para crear más usuarios, nodos, sensores y datos.

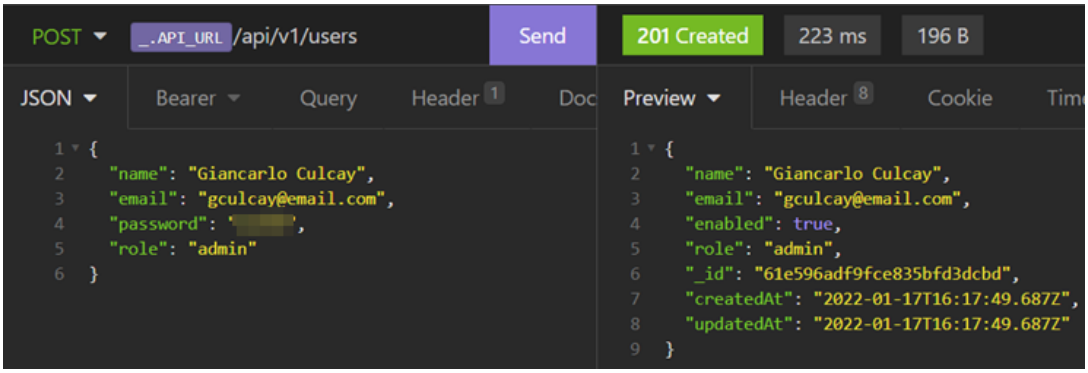


Figura 44 Creación de usuario administrador

Banco de Pruebas

Para demostrar el funcionamiento de la API REST con las redes de sensores se ha desarrollado 3 nodos sensores los cuales se conectan mediante el protocolo MQTT al bróker EMQ X alojado en la nube en puerto 1883. Los nodos sensores están compuestos por (figura 45).

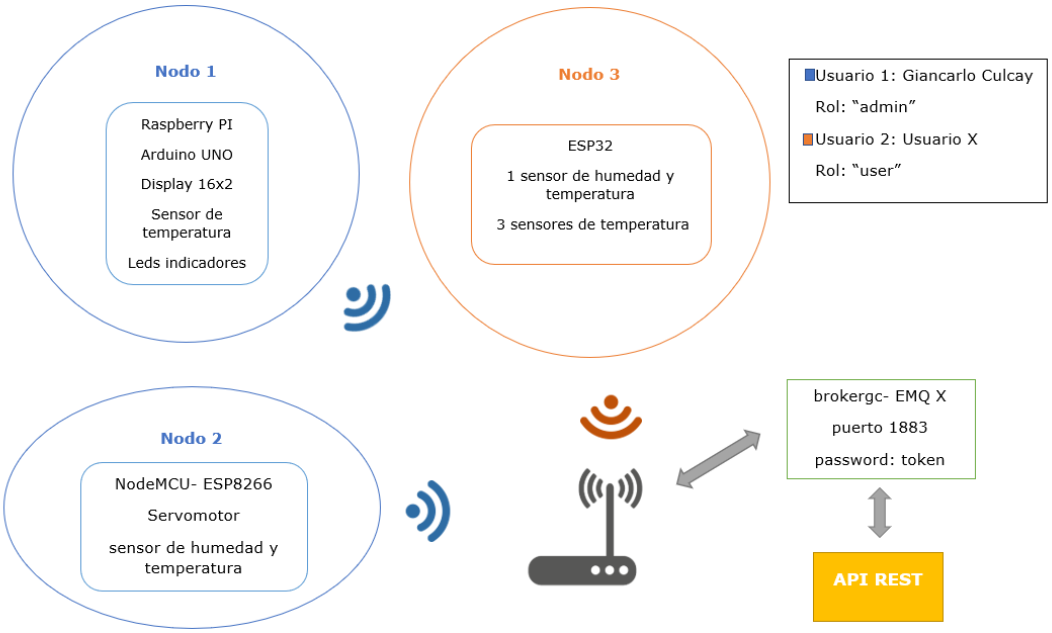


Figura 45 Red de sensores para las pruebas del broker y API REST

Elaborado por el investigador

Nodo 1

El nodo 1 está compuesto por la placa Arduino UNO un sensor de temperatura, una pantalla LCD 16X2, un sensor de temperatura LM35, un led rojo, un led verde y la Raspberry PI 4.

Para realizar las consultas a la API REST se usa el software Insomnia (ANEXO A) y su uso (ANEXO M).

Para crear un nodo, sensor, en la API primero el usuario debe autenticarse (ANEXO L) y conseguir el token, luego puede usar los demás endpoints.

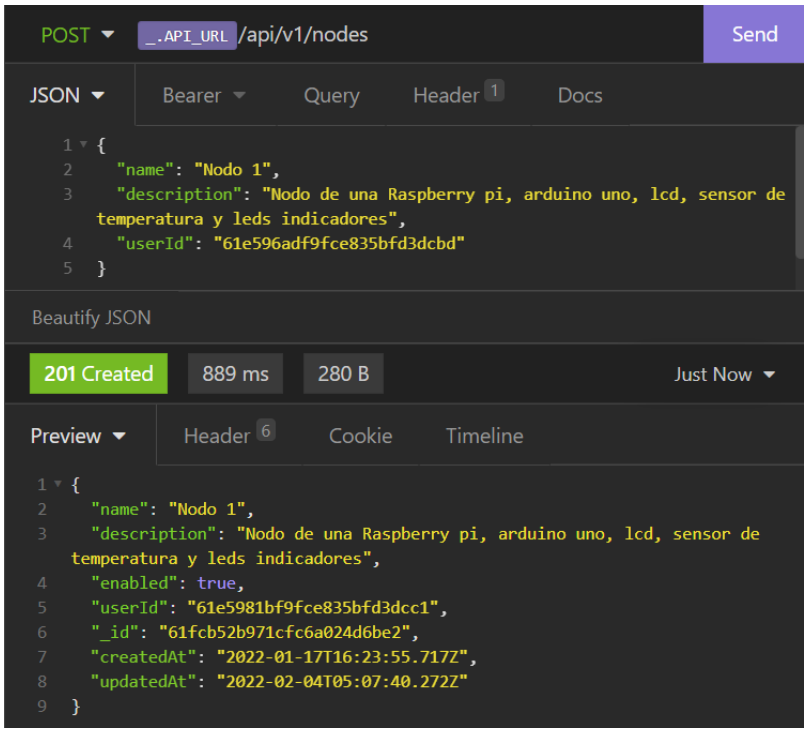


Figura 46 Creación Nodo 1

Una vez creado el nodo se pueden crear sensores relacionados a este nodo

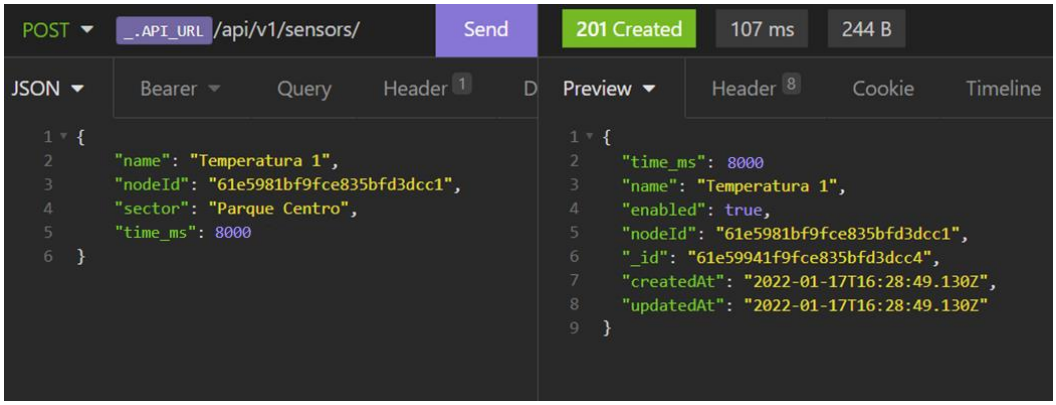


Figura 47 Creación sensor nodo 1

Raspberry pi

Sirve como un concentrador de datos para enviar los datos por wifi hacia el bróker.

Se crea el archivo datos_pi.py que se encarga de enviar los datos del Arduino UNO hacia el bróker y recibe datos para configurar el tiempo de muestreo “time_ms”, estado de los leds “led” y un mensaje del usuario “mensaje”.

El código de datos_pi.py se muestra en el ANEXO O

Arduino UNO

Los datos que le llegan por la conexión serial con la Raspberry PI tienen formato JSON por lo cual se usa la librería ArduinoJson.h para deserializar los datos:

```
String payload = Serial.readString();
StaticJsonDocument<256> doc;
DeserializationError err = deserializeJson(doc, payload);
if (err){
  Serial.println("ocurrio un error");
  Serial.println(err.c_str());
  return;
}
```

Las variables tienen el siguiente formato

```
bool led = doc["led"];
time_ms = doc["time_ms"] | time_ms;
String mensaje = doc["mensaje"] | "";
```

El código se muestra en el ANEXO P

Resultados:

El funcionamiento normal del circuito es mostrar la temperatura en la pantalla LCD y enviar este dato hacia la Raspberry PI que se conecta al bróker del contenedor de Azure y publicar los datos continuamente en "/61e59941f9fce835bfd3dcc4/data". Figura 48

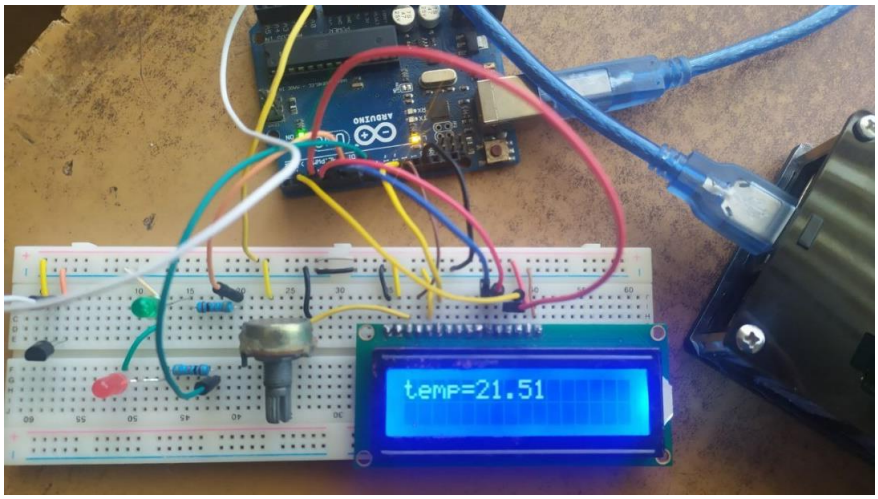


Figura 48 Circuito Nodo 1

Elaborado por el investigador

Cuando el usuario actualiza un valor de: "led", "time_ms" o "mensaje" (figura 49) el cliente mqtt de la API envía estos cambios a "/61e59941f9fce835bfd3dcc4/config". La raspberry pi que esta suscrita a este tópico manda estos datos al Arduino y así logra controlar esta red (figura 50)

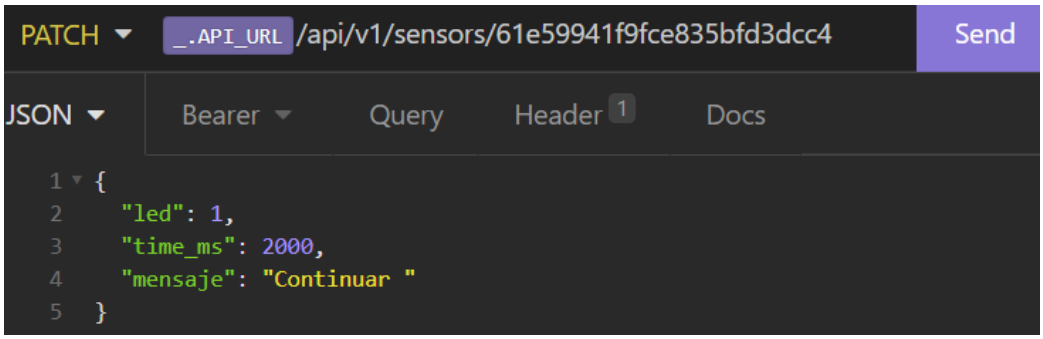


Figura 49 Primer envió de parámetros al Nodo 1

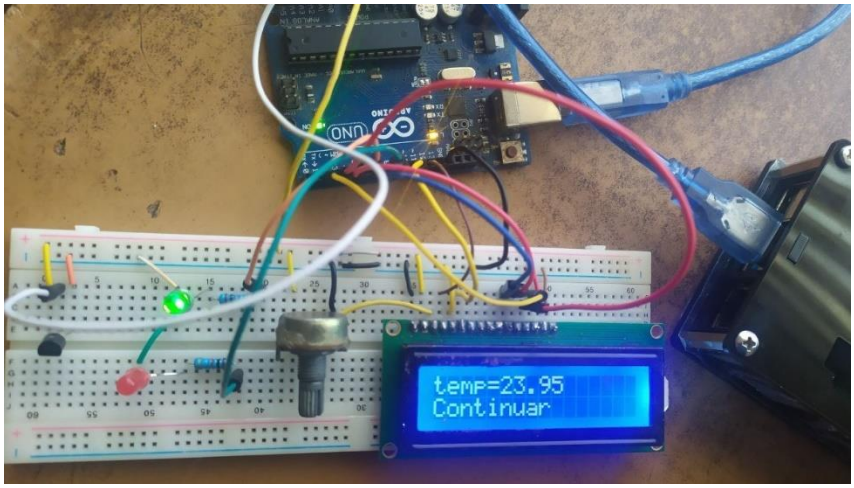


Figura 50 Primera actualización de parámetros del Nodo 1

Elaborado por el investigador

Enviar datos y recibir los parámetros puede realizarse simultáneamente, el usuario puede volver a enviar los parámetros figura 51 y se realizan los cambios como se observa en la figura 52

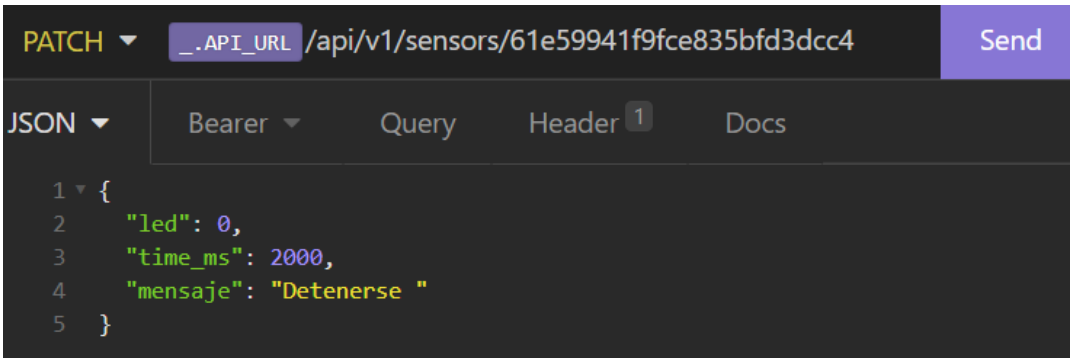


Figura 51 Segundo envió de parámetros al Nodo 1

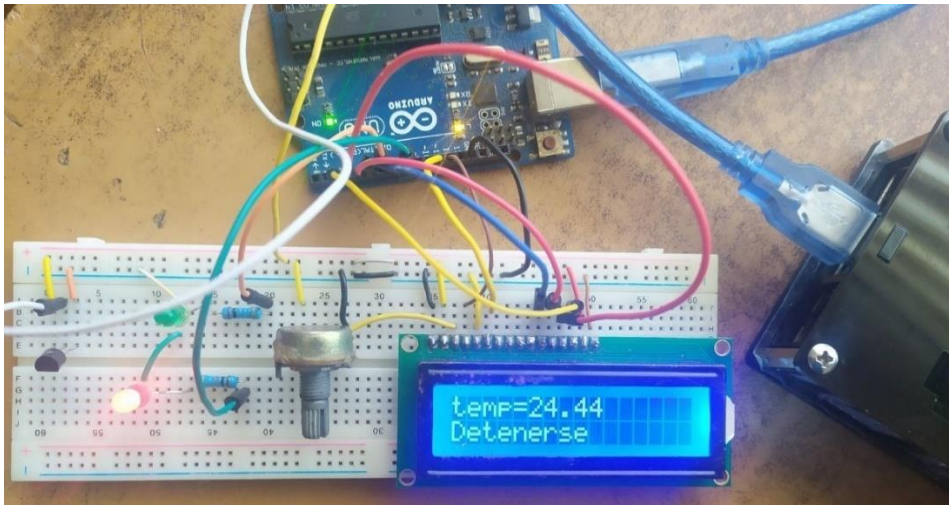


Figura 52 Segunda actualización de parámetros del Nodo 1

Elaborado por el investigador

Para visualizar y usar los datos en la API REST se tienen muchos métodos es por esto que manejar los datos en una API es muy conveniente y tiene sus ventajas. Figura 53 y 54

Figura 53 Datos de Nodo 1 en consola

En el Anexo R, se muestran algunas formas muy útiles y se verifican los datos.

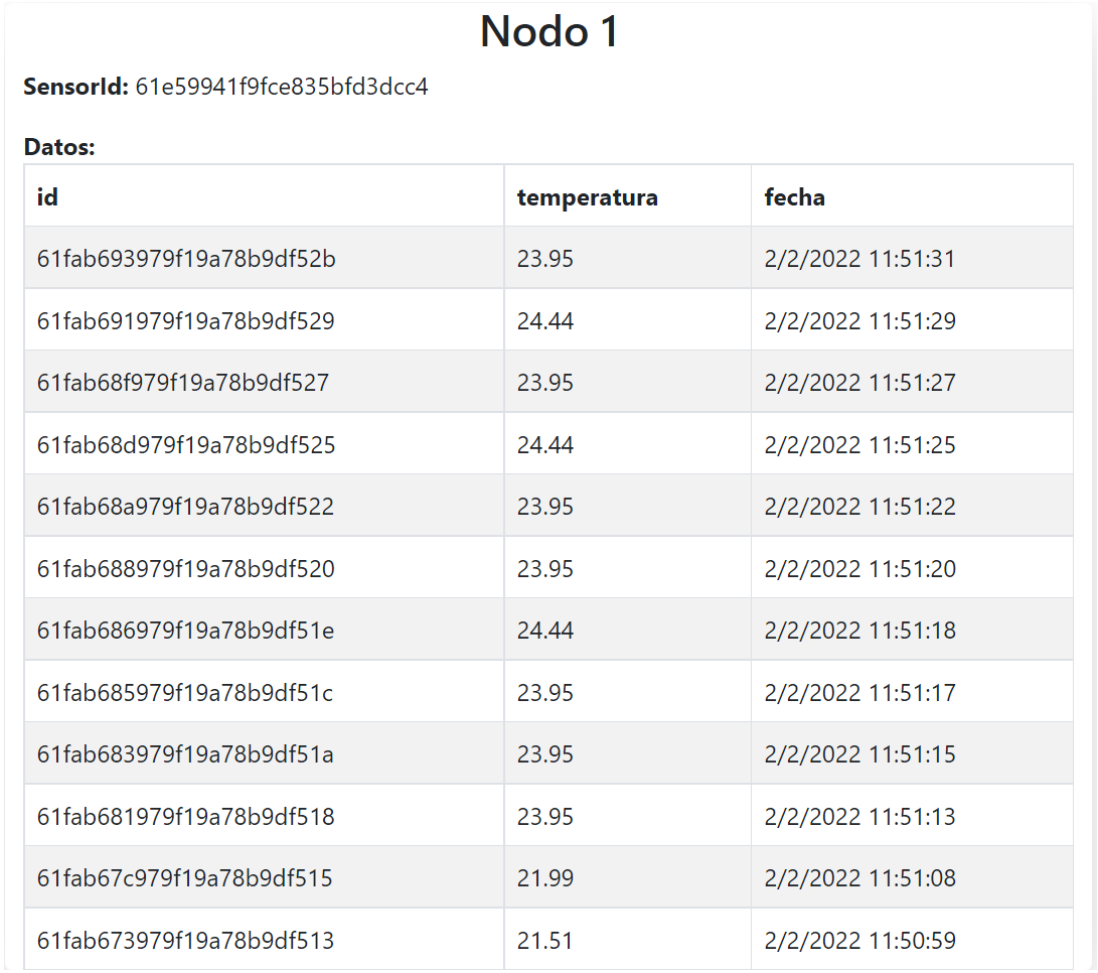


Figura 54 Tabla de datos de Nodo 1

Elaborado por el investigador

Nodo 2

El nodo 2 este compuesto por la placa NodeMCU-ESP8266, un sensor de temperatura y humedad: DHT11 y un servomotor.

Creación de nodo 2 figura 55 y sensor figura 56.

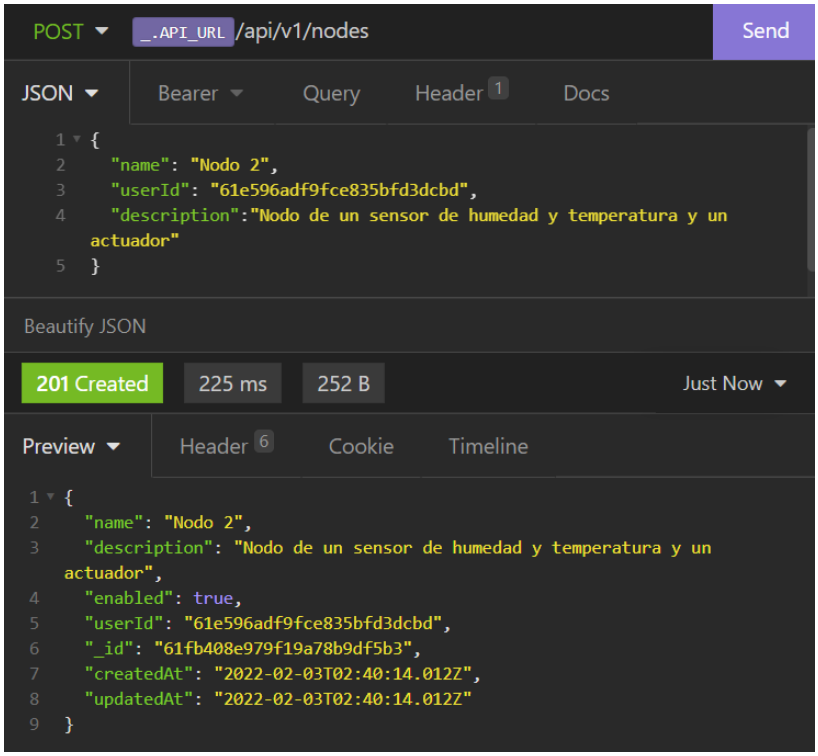


Figura 55 Creación nodo 2

Una vez creado el nodo se crea el sensor

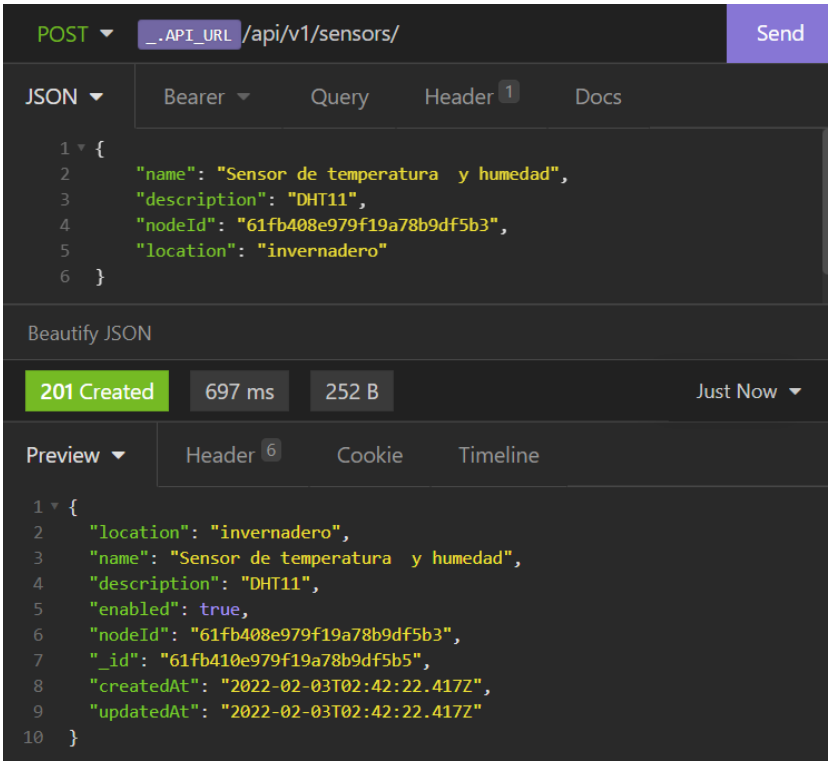


Figura 56 Creación sensor de nodo 2

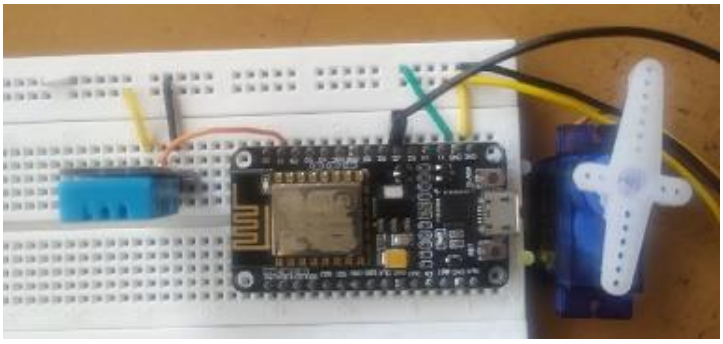


Figura 57 Circuito Nodo 2

Elaborado por el investigador

Para recibir y enviar datos es necesario que tenga formato JSON, a continuación, se muestra como serializar los datos publicar los datos en este formato con ArduinoJson.h. El código se muestra en el ANEXO U

```
DynamicJsonDocument doc(192);
doc["temperatura"] = temperatura;
doc["humedad"] = humedad;

serializeJson(doc, payload);
Serial.print("Publish message: ");
Serial.println(payload);
client.publish("/61fb410e979f19a78b9df5b5/data", payload.c_str());
```

En el monitor serial se puede ver los datos publicados y el mensaje que llega con los datos: “angle” y “time_ms” los cuales son parámetros del ángulo del servo y tiempo de muestreo del sensor DHT11.

Resultados:

El funcionamiento del circuito es enviar los datos se humedad y temperatura al bróker del contenedor de Azure y publicar los datos continuamente en “/61fb410e979f19a78b9df5b5/data”, ver ANEXO S

Se puede observar los datos enviados en la figura 58.

Figura 58 Datos monitor serial, nodo 2

Igual que el anterior nodo se pueden enviar parámetros de configuración del nodo, en este caso el ángulo de movimiento de un servomotor. Figura 59

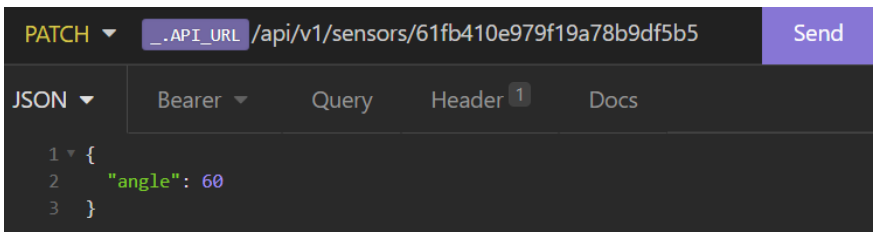


Figura 59 Primer envió de parámetros al Nodo 2

Se puede observar más envió de parámetros y respuestas en el ANEXO S, los datos del nodo 2 se pueden observar en la figura 60.

Nodo 2

SensorId: 61fb410e979f19a78b9df5b5

Datos:

temperatura	humedad	fecha
22.8	46	3/2/2022 12:08:30
22.7	47	3/2/2022 12:06:30
22.7	46	3/2/2022 12:04:30
22.7	46	3/2/2022 12:02:30
22.8	46	3/2/2022 12:00:30
22.7	47	3/2/2022 11:58:30
22.5	48	3/2/2022 11:56:30
22.5	47	3/2/2022 11:54:30
22.6	46	3/2/2022 11:52:30
22.5	46	3/2/2022 11:50:30
22.6	47	3/2/2022 11:48:30
22.7	47	3/2/2022 11:46:30

Figura 60 Tabla de datos de Nodo 2

Elaborado por el investigador

Nodo 3

Este nodo cuenta con un ESP32, un sensor DHT11 para medir la temperatura y humedad además de 3 sensores de temperatura lm35 (figura 63), estos 5 datos se envían al mismo tópico, y servirán para realizar pruebas de rendimiento.

Creación de nodo figura 61 y sensor figura 62

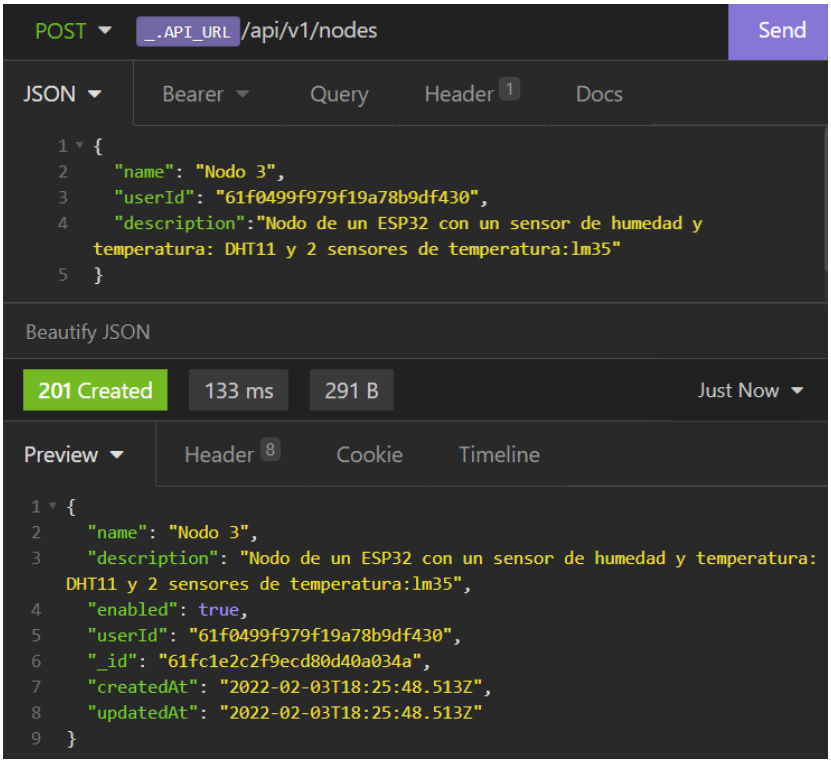


Figura 61 Creación nodo 3

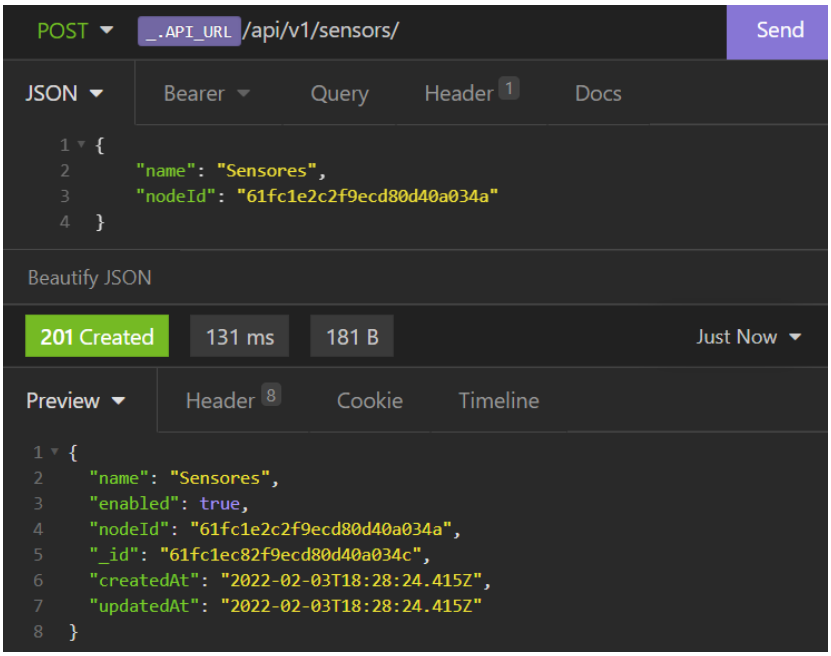


Figura 62 Creación sensor de nodo 3

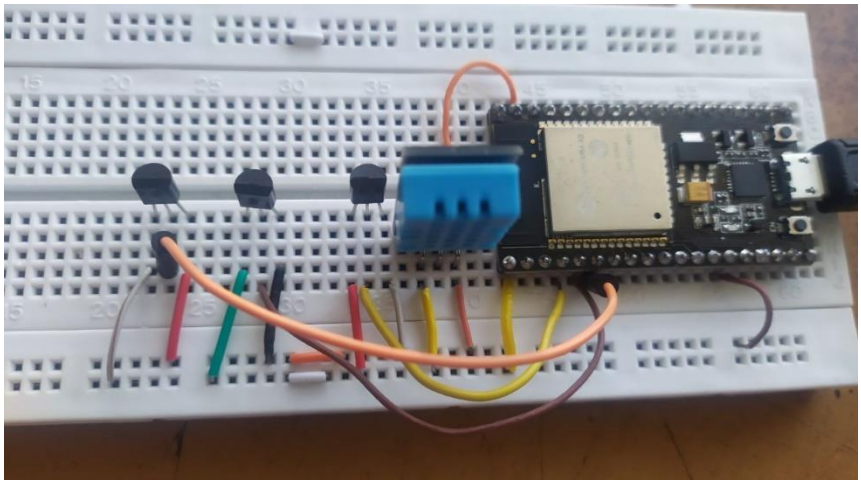


Figura 63 Circuito nodo 3

Elaborado por el investigador

En el Anexo V se puede observar el código fuente de este nodo

Resultados:

Se pueden observar los 5 datos enviados a la vez por la ESP32 cada segundo.

COM4

```
Publish message: {"dht_temperatura":23.1,"dht_humedad":51,"temperatura1":23.87912,"temperatura2":24.48962,"temperatura3":24.48962}
Publish message: {"dht_temperatura":24.3,"dht_humedad":87,"temperatura1":26.44322,"temperatura2":24.48962,"temperatura3":24.12332}
Publish message: {"dht_temperatura":25.7,"dht_humedad":98,"temperatura1":25.58852,"temperatura2":24.00122,"temperatura3":23.87912}
Publish message: {"dht_temperatura":25.9,"dht_humedad":99,"temperatura1":24.12332,"temperatura2":22.53602,"temperatura3":22.29182}
Publish message: {"dht_temperatura":25.7,"dht_humedad":99,"temperatura1":24.48962,"temperatura2":22.90232,"temperatura3":22.90232}
Publish message: {"dht_temperatura":25.7,"dht_humedad":99,"temperatura1":25.58852,"temperatura2":24.12332,"temperatura3":24.12332}
Publish message: {"dht_temperatura":25.6,"dht_humedad":99,"temperatura1":24.12332,"temperatura2":22.65812,"temperatura3":22.90232}
Publish message: {"dht_temperatura":25.4,"dht_humedad":99,"temperatura1":24.00122,"temperatura2":22.29182,"temperatura3":22.90232}
Publish message: {"dht_temperatura":25.3,"dht_humedad":99,"temperatura1":23.63492,"temperatura2":21.68132,"temperatura3":22.41392}
Publish message: {"dht_temperatura":24.6,"dht_humedad":96,"temperatura1":23.39072,"temperatura2":21.19292,"temperatura3":22.29182}
Publish message: {"dht_temperatura":24.3,"dht_humedad":95,"temperatura1":23.51282,"temperatura2":21.43712,"temperatura3":22.04762}
Publish message: {"dht_temperatura":23.9,"dht_humedad":90,"temperatura1":23.14652,"temperatura2":21.43712,"temperatura3":21.80342}
Publish message: {"dht_temperatura":23.6,"dht_humedad":87,"temperatura1":23.63492,"temperatura2":21.68132,"temperatura3":22.29182}
Publish message: {"dht_temperatura":23.5,"dht_humedad":82,"temperatura1":23.75702,"temperatura2":21.92552,"temperatura3":22.65812}
Publish message: {"dht_temperatura":23.6,"dht_humedad":76,"temperatura1":23.39072,"temperatura2":21.92552,"temperatura3":22.53602}
Publish message: {"dht_temperatura":23.6,"dht_humedad":72,"temperatura1":23.26862,"temperatura2":21.92552,"temperatura3":21.92552}
Publish message: {"dht_temperatura":23.6,"dht_humedad":68,"temperatura1":23.39072,"temperatura2":21.80342,"temperatura3":22.29182}
Publish message: {"dht_temperatura":23.6,"dht_humedad":64,"temperatura1":23.75702,"temperatura2":21.80342,"temperatura3":22.41392}
```

Figura 64 Datos monitor serial, nodo 3

Un dato tiene la siguiente estructura con un tamaño de 264B, figura

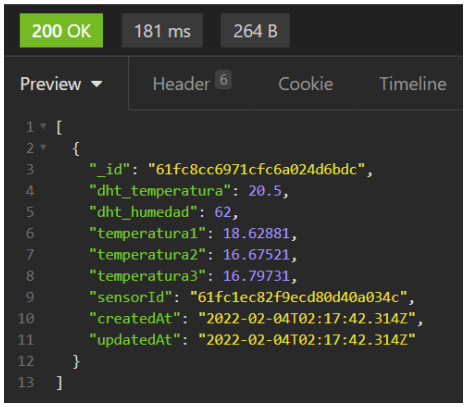


Figura 65 Dato de nodo 3

Nodo 3

SensorId: 61fc1ec82f9ecd80d40a034c

Datos:

dht_temperatura	dht_humedad	temperatura 1	temperatura 2	temperatura 3	fecha
20.5	62	18.62881	16.67521	16.79731	3/2/2022 21:17:42
20.4	62	18.38461	16.55311	16.67521	3/2/2022 21:17:41
20.4	62	18.50672	16.67521	16.91941	3/2/2022 21:17:40
20.5	62	18.38461	16.55311	16.67521	3/2/2022 21:17:39
20.5	62	18.62881	16.67521	16.79731	3/2/2022 21:17:38
20.4	62	18.62881	16.55311	16.67521	3/2/2022 21:17:38
20.4	62	18.62881	16.67521	16.67521	3/2/2022 21:17:38
20.4	62	18.38461	16.67521	16.55311	3/2/2022 21:17:35
20.4	62	18.62881	16.55311	16.91941	3/2/2022 21:17:34
20.5	62	18.50672	16.67521	16.67521	3/2/2022 21:17:33
20.5	62	18.26252	16.79731	16.67521	3/2/2022 21:17:32
20.4	62	18.26252	16.30891	16.67521	3/2/2022 21:17:31
20.4	62	18.62881	16.67521	16.91941	3/2/2022 21:17:30
20.4	62	18.26252	16.43101	16.67521	3/2/2022 21:17:29
20.4	62	18.62881	16.55311	16.67521	3/2/2022 21:17:28
20.4	62	18.01831	16.43101	16.67521	3/2/2022 21:17:27
20.4	62	18.38461	16.43101	16.67521	3/2/2022 21:17:26
20.4	62	18.01831	16.55311	16.43101	3/2/2022 21:17:25
20.4	62	18.62881	16.67521	16.79731	3/2/2022 21:17:24
20.4	62	18.01831	16.55311	16.67521	3/2/2022 21:17:23
20.4	62	18.26252	16.30891	16.67521	3/2/2022 21:17:22
20.4	62	18.26252	16.43101	16.67521	3/2/2022 21:17:21
20.4	62	18.62881	16.67521	16.67521	3/2/2022 21:17:20
20.4	62	18.26252	16.43101	16.67521	3/2/2022 21:17:19
20.4	62	18.50672	16.55311	16.79731	3/2/2022 21:17:18

Figura 66 Tabla de datos de Nodo 3

Elaborado por el investigador

Para este nodo se tomaron más de 10000 datos ver ANEXO T, que servirán para las pruebas de rendimiento de la API

Pruebas de rendimiento de la API REST

Para realizar estas pruebas se utilizó el programa JMETER, principalmente se probó con las peticiones GET para obtener los datos de los sensores, cada consulta por defecto devuelve 10 datos y se puede cambiar a más con el campo "limit" como se vio en la sección Ruta de datos por sensor.

Es importante volver a mencionar las características del hardware que tiene la aplicación: ACU:100, 1,75 GB de memoria, equivalente de proceso de serie A, esta aplicación esta hospedada en Azure App Service y cuenta con las características del plan básico. Además del plan gratuito de MongoDB Atlas

Las solicitudes son de tipo HTTPS y autenticadas con su token con lo cual pierde rendimiento, pero se tiene conexiones seguras.

Al inicio se considera un escenario normal en la que varios usuarios consultan los datos de los sensores y se obtiene los siguientes datos:

- Número de hilos: 100
- Periodo de subida: 1 segundo
- Bucle: 1

Estado de las peticiones: 200, resultados tabla 10

La cantidad de datos son el número de datos que entrega la solicitud

Tabla 10 Resultados primera prueba

Cantidad de datos:	**10**	**500**
Número de muestras	100	100
Media	1338	6100
Desviación	240	2496.48
%Error	0.00%	0.00%
Rendimiento	44.5/sec	9.0/sec
Kb/sec	89.19	802.12

Kb/sec enviado	14.42	2.99
Media de Bytes	2054.0	91665.0
Tiempo de conexión promedio (ms)	390	500

Elaborado por el investigador

- Número de hilos: 100
- Periodo de subida: 1 segundo
- Bucle: 10

Estado de las peticiones: 200, resultados tabla 11

Tabla 11 Resultados segunda prueba

Cantidad de datos:	**10**	**500**
Número de muestras	1000	1000
Media	1566	10065
Desviación	574.51	2580.71
%Error	0.00%	0.00%
Rendimiento	57.3/sec	9.3/sec
Kb/sec	114.86	830.05
Kb/sec enviado	18.56	3.10
Media de Bytes	2054.0	91665.0
Tiempo de conexión promedio (ms)	460	490

Elaborado por el investigador

- Número de hilos: 1000
- Periodo de subida: 1 segundo
- Bucle: 10

Estado de las peticiones: 200, resultados tabla 12.

En esta prueba se comienza a ver los primeros errores y la latencia comienza aumentar mientras más cantidad de datos se solicitan

Tabla 12 Resultados tercera prueba

Cantidad de datos:	**10**	**500**
Número de muestras	10000	10000
Media	10658	97036
Desviación	4977.52	24627.59
%Error	13.14%	4.15%
Rendimiento	77.1/sec	9.7/sec
Kb/sec	163.18	830.79
Kb/sec enviado	21.72	3.10
Media de Bytes	2166.1	87981.5
Tiempo de conexión promedio (ms)	460	1500, existieron picos de 18000

Elaborado por el investigador

- Número de hilos: 10
- Periodo de subida: 1 segundo
- Bucle: 1

Estado de las peticiones: 200, resultados tabla 12.

Tabla 13 Resultados cuarta prueba

Cantidad de datos:	**10000**	**5000**	**10000**
Número de muestras	10	100	50
Media	17463	12925	97036
Desviación	4279.73	5596.29	24627.59
%Error	0.00%	0.00%	4.15%
Rendimiento	25.4/min	4.4/sec	9.7/sec
Kb/sec	1087.84	1138.65	830.79

Kb/sec enviado	0.14	1.48	3.10
Media de Bytes	2629691.0	263232.0	87981.5
Tiempo de conexión promedio (ms)	500	485	1500, existieron picos de 18000

Elaborado por el investigador

El tiempo de respuesta se elevó ya que cada consulta está en el orden de los MB, vemos que mientras mayor sea la carga de la respuesta y la cantidad de solicitudes se empieza a ver altos retrasos.

Mediante varias pruebas se determinó que lo óptimo es menos de 100 usuarios enviando solicitudes con una cantidad de datos máxima de 2000 datos.

La respuesta de datos de la API por defecto son 10 y se puede tener una cantidad de 100 a 300 usuarios consultando datos continuamente sin que el servicio tenga problemas.

A continuación, se muestra un gráfico con los datos de salida realizada en estas pruebas:

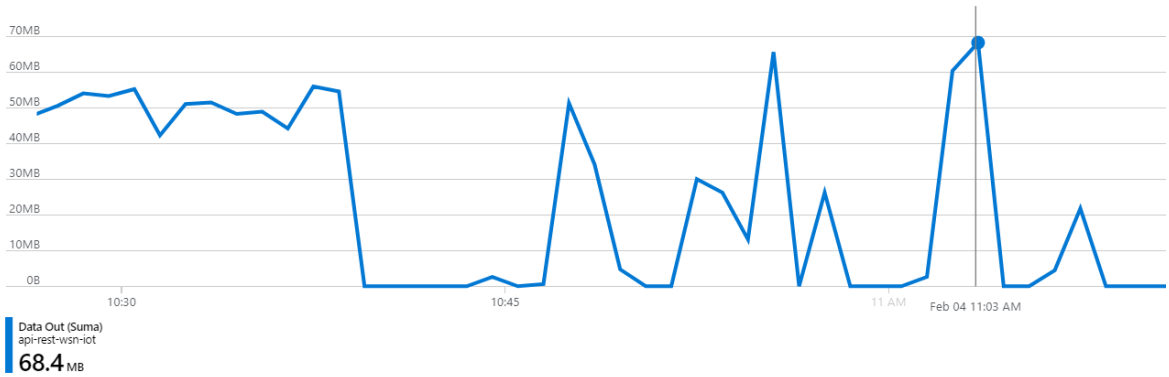


Figura 67 Supervisión de Azure: datos salientes

La cantidad de solicitudes y tiempo de respuesta promedio según la supervisión de Azure:

Figura 68 Solicitudes y tiempo de respuesta promedio

Precios de los servicios

Al utilizar soluciones serverless se paga por el uso de las instancias, esto se descuenta del crédito de la suscripción

Azure Container Instances se factura por "grupos de contenedores", que son asignaciones de recursos de memoria o vCPU que pueden usarse en un solo contenedor o dividirse entre varios contenedores. El precio depende del número de vCPU y de GB de memoria solicitado. Se utilizó 1.5GB y 1 cpu.

SO Linux

Duración del grupo de contenedores

Recursos	Precio por segundo	Precio por hora
Memoria	$0,0000015 por GB	$0,00533 por GB
vCPU	$0,0000135 por vCPU	$0,04860 por vCPU

Figura 69 Precios de Container Instances para recursos en el Centro-oeste de EEUU

Se uso en el tiempo de desarrollo y producción durante 20 días

Uso de memoria: 2.55 USD

Uso de cpu: 23.328 USD

Uso de la aplicación en App Services 13,14 USD que corresponde al plan B1

Para el almacenamiento de datos se usó MongoDB Atlas en su capa gratuita.

CAPITULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

Una vez finalizado el trabajo de investigación se determinaron las siguientes conclusiones.

- Una vez analizado las tecnologías disponibles para el desarrollo de una API REST, Node.js resulto ser el más óptimo para este proyecto debido a sus características de rendimiento, aplicaciones en tiempo real e IoT y junto a Express que resulto muy útil para construir la API desde cero para las redes de sensores.

- Para las conexiones de las redes de sensores IoT se determinó que usar el protocolo MQTT es la mejor opción para este proyecto ya que se realizó múltiples suscripciones a la configuración mediante el cliente de MQTT.js de node.js y múltiples publicaciones de los datos sin tener problemas. El bróker EMQ X fue el más adecuado por su capacidad de conexiones, autenticación con JWT y demás.

- Las redes de sensores autenticadas publican sus datos al bróker y la aplicación API REST guarda los datos, todo esto ocurre en la nube y en tiempo real, cualquier cliente HTTP puede consultar los datos en formato JSON más recientes de los sensores de varias maneras y usarlos en cualquier proyecto con cualquier lenguaje de programación que realice solicitudes HTTP.

- Se realizó un banco de pruebas para probar las conexiones con el broker MQTT y la gestión de datos con la API REST, con la ruta de sensor y el método PATCH de la API se logró enviar parámetros de configuraciones hacia los microcontroladores en tiempo real.

- Durante las pruebas de rendimiento se determinó que lo óptimo para esta aplicación con sus recursos es menos de 100 usuarios enviando solicitudes con una cantidad máxima de 2000 datos, y que menos de 300 usuarios pueden consultar continuamente la cantidad de datos por defecto.

4.2 Recomendaciones

Con el desarrollo de este proyecto se recomienda lo siguiente:

- Ampliar las características de memoria y cpu del contenedor, crear clusters para mayor rendimiento de redes de sensores grandes o contratar servicios en la nube orientados solo a MQTT.

- Para soportar mayor cantidad de datos se debe ampliar las características del servicio, escalar vertical y horizontalmente según se requiera, además las conexiones y consultas a la base de datos deben ser rápidas por lo cual se recomienda contratar planes pagados de la base de datos.

- Se puede mejorar las consultas de datos agregando más variables como fechas, orden, filtros, etc, lo que llevaría a más tiempo de programación.

- Se puede realizar scripts internos para borrar datos antiguos y nodos, sensores de usuarios que ya no están activos o han sido eliminados.

MATERIALES DE REFERENCIA

Referencias Bibliográficas

[1] M. Quiñones, «SISTEMA DE MONITOREO DE VARIABLES MEDIO AMBIENTALES USANDO UNA RED DE SENSORES INALÁMBRICOS Y PLATAFORMAS DE INTERNET DE LAS COSAS,» Loja, 2017.

[2] B. H. Longa Chevarría, «REST API FOR MANAGEMENT OF ELECTRONIC DEVICES,» Lima, 2018.

[3] G. Bianchini, «Desarrollo de un API REST para transmisión de datos de sensores GPS,» Valencia, 2018.

[4] J. A. Benavídez Gómez y J. D. García Acevedo, «ARQUITECTURA REST PARA LA PLATAFORMA UAO-IoT,» Santiago de Cali, 2019.

[5] H. Garg y M. Dave, «Securing IoT Devices and SecurelyConnecting the Dots Using REST API and Middleware,» de *International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 2019.

[6] R. O. D. García, M. J. R. L. Huerta y M. M. G. Dueñas, «API like Services for Multiple Systems Oriented to IoT,» de *8th International Conference On Software Process Improvement (CIMPS)*, 2019.

[7] Equipo de Expertos Universidad Internacional de Valencia, «Viu,» 08 Enero 2021. [En línea]. Available: https://www.universidadviu.com/ec/actualidad/nuestros-expertos/dispositivos-iot-un-reto-para-la-seguridad-de-las-empresas-y. [Último acceso: 25 Octubre 2021].

[8] Editing Team, «chakray,» 2020. [En línea]. Available: https://www.chakray.com/es/retos-iot/. [Último acceso: 22 Octubre 2021].

[9] S. Cirani, G. Ferrari, M. Picone y L. Veltri, Internet of Things: Architectures, Protocols and Standards, Wiley, 2018.

[10] J. Cuevas, «Desarrollo de Servicio REST API para el uso seguro de dispositivos IoT,» Málaga, 2019.

[11] V. Shivangi, R. Jyotsnamayee, M. Janit, V. Saurav y P. Chetana, «Internet of Things (IoT): A vision, architectural elements, and security issues,» de

International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, 2017.

[12] Sathish y S. Smys, «A Survey on Internet of Things (IoT) Smart Systems,» *Journal of ISMAC,* p. 189, 2020.

[13] A. Medelllin, A. Escarcia y R. Aguilera, «Dynamic system for monitoring and control wireless sensor networks operating under,» Scielo, 2019.

[14] N. Chandrakant y L. Suresh, The Today and Future of WSN, AI, and IoT, BPB Publications, 2020.

[15] Tekniker, «Tekniker,» [En línea]. Available: https://www.tekniker.es/es/redes-de-sensores. [Último acceso: 01 Julio 2021].

[16] E. J. Guaña Moya, «DISEÑO DE UNA RED DE SENSORES INALÁMBRICOS (WSN) PARA MONITOREAR PARÁMETROS RELACIONADOS CON LA AGRICULTURA,» Quito, 2016.

[17] G. S.Karthick y P. Pankajavalli, «A Review on Human Healthcare Internet of Things: A Technical Perspective,» *SN Computer Science,* p. 198, 11 Junio 2020.

[18] «Hardware Libre,» [En línea]. Available: https://www.hwlibre.com/mqtt/. [Último acceso: 20 Diciembre 2021].

[19] Microsoft, «Azure,» [En línea]. Available: https://azure.microsoft.com/es-es/overview/internet-of-things-iot/iot-technology-protocols/. [Último acceso: 01 Julio 2021].

[20] L. del Valle Hernández, «Programar Facil,» [En línea]. Available: https://programarfacil.com/podcast/proyectos-iot-con-arduino/. [Último acceso: 28 Junio 2021].

[21] M. Yuan, «IBM Developer,» 4 Noviembre 2017. [En línea]. Available: https://developer.ibm.com/es/articles/iot-mqtt-why-good-for-iot/. [Último acceso: Diciembre 20].

[22] «Steve's Internet Guide,» [En línea]. Available: http://www.steves-internet-guide.com/mqtt/. [Último acceso: 20 Diciembre 2021].

[23] «Arrow,» 3 Junio 2017. [En línea]. Available: https://www.arrow.com/es-mx/research-and-events/articles/protocols-for-the-internet-of-things. [Último acceso: 1 Julio 2021].

[24] «Steve's Internet Guide,» [En línea]. Available: http://www.steves-internet-guide.com/mqtt-works/. [Último acceso: 19 Diciembre 2021].

[25] «Mqtt,» [En línea]. Available: https://mqtt.org/software/. [Último acceso: 26 Diciembre 2021].

[26] [En línea]. Available: https://mosquitto.org/man/mosquitto-8.html. [Último acceso: 26 Diciembre 2021].

[27] EMQ, «Documentación EMQ X,» [En línea]. Available: https://docs.emqx.io/en/broker/v4.3/. [Último acceso: 26 Diciembre 2021].

[28] «hivemq,» [En línea]. Available: https://www.hivemq.com/docs/hivemq/4.7/user-guide/introduction.html. [Último acceso: 26 Diciembre 2021].

[29] «HBMQTT,» [En línea]. Available: https://hbmqtt.readthedocs.io/en/latest/. [Último acceso: 26 Diciembre 2021].

[30] M. Yuan, «IBM Developer,» 2017. [En línea]. Available: https://developer.ibm.com/es/technologies/iot/articles/iot-mqtt-why-good-for-iot/. [Último acceso: 30 Junio 2021].

[31] M. Barrio, Internet de las cosas, Madrid: Reus, 2017.

[32] NIST, «National Institute of Standards and Technology,» [En línea]. Available: https://www.nist.gov/programs-projects/nist-cloud-computing-program-nccp. [Último acceso: 26 Diciembre 2021].

[33] Azure, «Microsoft Learn,» [En línea]. Available: https://azure.microsoft.com/es-es/overview/types-of-cloud-computing/. [Último acceso: 19 Octubre 2021].

[34] M. J. Parra Royón, «Servicios de minería de datos en Cloud Computing,» Granada, 2019.

[35] RedHat, «RedHat,» [En línea]. Available: https://www.redhat.com/es/topics/microservices. [Último acceso: 27 Diciembre 2021].

[36] AWS, «AWS,» [En línea]. Available: https://aws.amazon.com/es/microservices/. [Último acceso: 27 Diciembre 2021].

[37] A. Nebel, «Arquitectura de microservicios para plataformas de integración,» UR.FI.INCO, Montevideo, 2019.

[38] Microsoft, «Documentación Microsoft,» [En línea]. Available: https://docs.microsoft.com/es-es/devops/deliver/what-are-microservices. [Último acceso: 27 Diciembre 2021].

[39] D. López y E. Maya, «Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web,» de *Séptima Conferencia de Directores de Tecnología de Información, TICAL 2017 Gestión de las TICs para la Investigación y la Colaboración, San José, del XX al XX de julio de 2017*, Imbabura, 2017.

[40] C. Masso, «Teldat,» [En línea]. Available: https://www.teldat.com/blog/es/microservicios-containers-dockers-kubernetes-para-sd-wan/. [Último acceso: 28 Diciembre 2021].

[41] Docker, «Documentación de Docker,» [En línea]. Available: https://docs.docker.com/get-started/overview/. [Último acceso: 28 Diciembre 2021].

[42] Docker, «Docker,» [En línea]. Available: https://www.docker.com/resources/what-container. [Último acceso: 28 Diciembre 2021].

[43] Red Hat, «RedHat,» [En línea]. Available: https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces. [Último acceso: 01 Julio 2021].

[44] z. M. Pére, «Geek Theory,» [En línea]. Available: https://geekytheory.com/que-es-una-api-rest-y-para-que-se-utiliza. [Último acceso: 29 Junio 2021].

[45] H. Subramanian y P. Raj, Hands-On RESTful API Design Patterns and Best Practices, Packt Publishing, 2019.

[46] « IETF Datatracker,» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc3986. [Último acceso: 30 Diciembre 2021].

[47] «MDN Web docs,» [En línea]. Available: https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction. [Último acceso: 26 Diciembre 2021].

[48] nodejs, «Nodejs,» [En línea]. Available: https://nodejs.dev/learn. [Último acceso: 26 Diciembre 2021].

[49] F. Doglio, Pro REST API Development with Node.js, Canelones: Appress, 2018.

[50] H. Dhaduk, «simform,» 11 Mayo 2021. [En línea]. Available: https://www.simform.com/blog/express-vs-django/. [Último acceso: 27 Diciembre 2021].

[51] Laravel, [En línea]. Available: https://laravel.com/docs/8.x#why-laravel. [Último acceso: 27 Diciembre 2021].

[52] R. Altube, 31 Marzo 2021. [En línea]. Available: https://openwebinars.net/blog/que-es-laravel-caracteristicas-y-ventajas/. [Último acceso: 27 Diciembre 2021].

[53] H. Dhaduk, 26 Marzo 2021. [En línea]. Available: https://www.simform.com/blog/laravel-vs-spring/. [Último acceso: 27 Diciembre 2021].

[54] [En línea]. Available: https://ifgeekthen.nttdata.com/es/spring-framework. [Último acceso: 27 Diciembre 2021].

[55] X. Rodríguez, «openwebinars,» 21 Agosto 2019. [En línea]. Available: https://openwebinars.net/blog/django-vs-flask/. [Último acceso: 27 Diciembre 2021].

[56] J. Patel, «monocubed,» 16 Noviembre 2021. [En línea]. Available: https://www.monocubed.com/django-vs-express/. [Último acceso: 27 Diciembre 2021].

[57] H. Dhaduk, 11 Febrero 2021. [En línea]. Available: https://www.simform.com/blog/laravel-vs-nodejs/. [Último acceso: 27 Diciembre 2021].

[58] «Developer Mozilla,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Overview. [Último acceso: 29 Diciembre 2021].

[59] MDN Web Docs, «Developer Mozilla,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Methods. [Último acceso: 01 Julio 2021].

[60] Oracle, «Oracle,» [En línea]. Available: https://www.oracle.com/mx/database/what-is-database/. [Último acceso: 28 Diciembre 2021].

[61] D. F. Sarabia Jácome, «Arquitectura de análisis de datos generados por el Internet de las cosas (IoT) en tiempo real,» Valencia, 2020.

[62] MongoDB, «MongoDB,» [En línea]. Available: https://www.mongodb.com/scale/internet-of-things-applications. [Último acceso: 27 Diciembre 2021].

[63] A. Robledano, «openwebinars,» [En línea]. Available: https://openwebinars.net/blog/que-es-mongodb/. [Último acceso: 28 Diciembre 2021].

[64] M. García, «Condingornot,» [En línea]. Available: https://codingornot.com/mvc-modelo-vista-controlador-que-es-y-para-que-sirve. [Último acceso: 30 Junio 2021].

[65] IBM Cloud Education, «IBM,» 2021 Abril 2021. [En línea]. Available: https://www.ibm.com/mx-es/cloud/learn/rest-apis. [Último acceso: 21 Octubre 2021].

[66] «Developer Mozilla,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Headers. [Último acceso: 29 Diciembre 2021].

[67] Microsoft Azure, «Documentación Azure,» [En línea]. Available: https://docs.microsoft.com/es-es/azure/container-instances/container-instances-overview. [Último acceso: 28 Diciembre 2021].

ANEXOS

Anexo A: Instalación de Software necesario

El desarrollo del proyecto se realizó en WSL 2 (Windows Subsystem for Linux) - Ubuntu 20.04

Node.js

Se instala node.js mediante el administrador de versiones de node: nvm

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | zsh
```

```
source ~/.zshrc
```

Para la realización del proyecto se usó la versión de 14.18.1 de node.js

```
nvm install v14.18.1
```

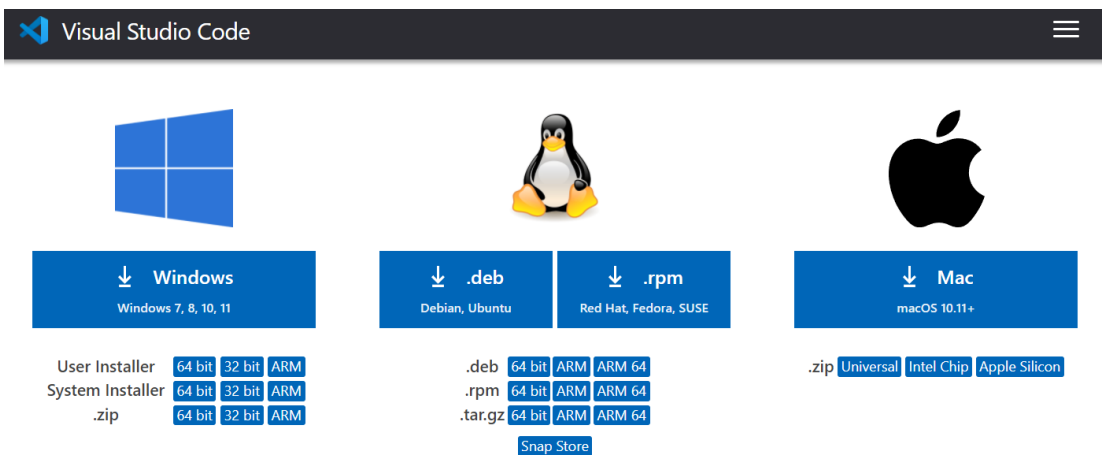
NPM

Es el sistema de gestión de paquetes por defecto para Node.js se instala de manera simultánea junto a node.js

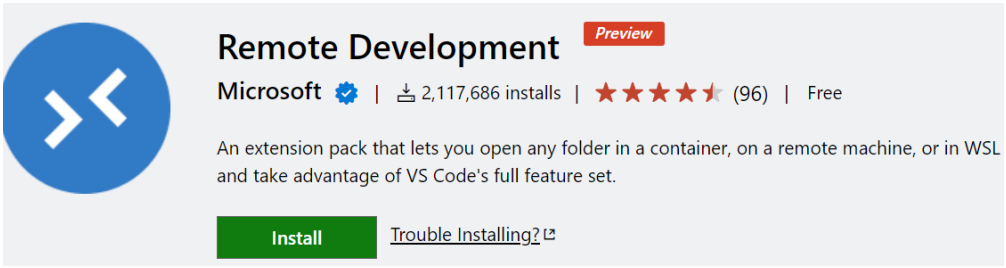
Visual Studio Code

Para la programación de la API REST se usó Visual Studio Code que se puede descargar desde su página code.visualstudio.com

Se descarga la versión para Windows



Para programar en WSL2-Ubuntu es necesario la siguiente extensión



Git

Se usa Git para el control de versiones del código y para subir el proyecto al repositorio de Github del investigador y posteriormente desplegar la API REST en Azure por medio de Github Actions.

Es posible que Git ya este instalado en Ubuntu, sino es el caso se instala con el comando siguiente:

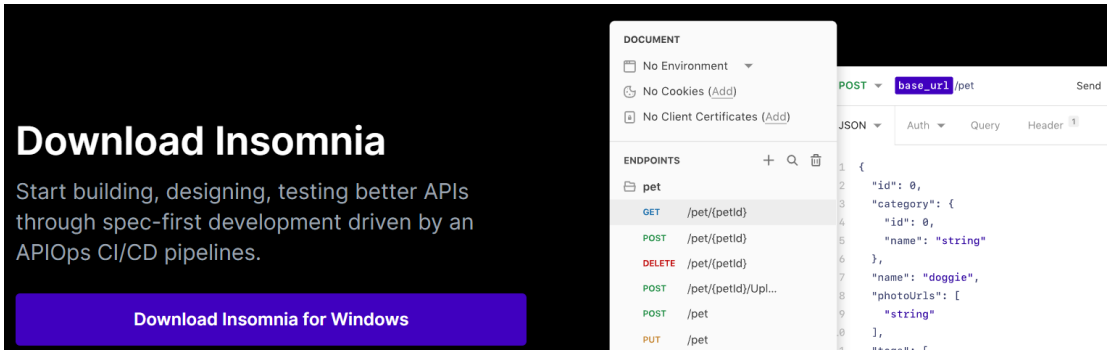
sudo apt install git

Para iniciar el repositorio se debe ubicar en el directorio del proyecto y ejecutar el siguiente comando:

git init

Insomnia

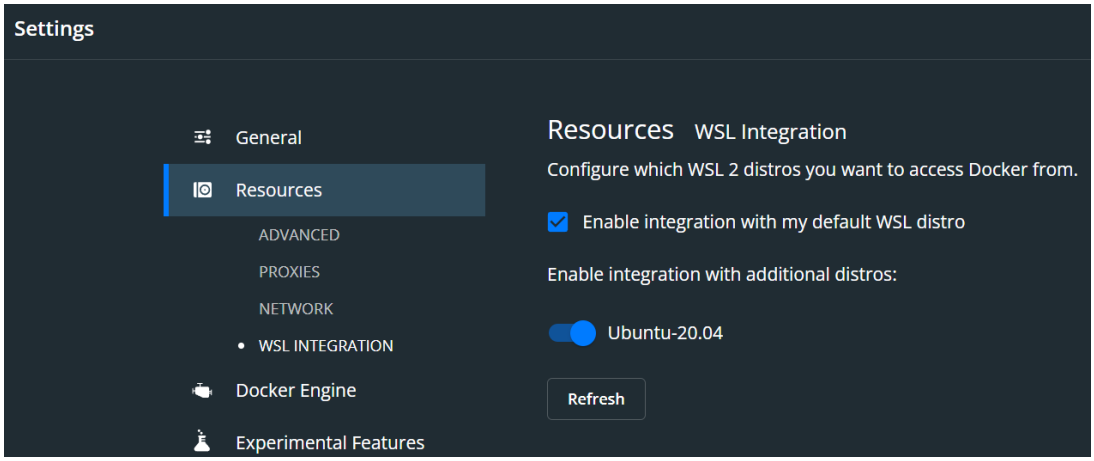
Insomnia es una aplicación de escritorio para probar la de API y nos ayuda en su desarrollo, está disponible en su página web de insomnia.rest



Docker Desktop

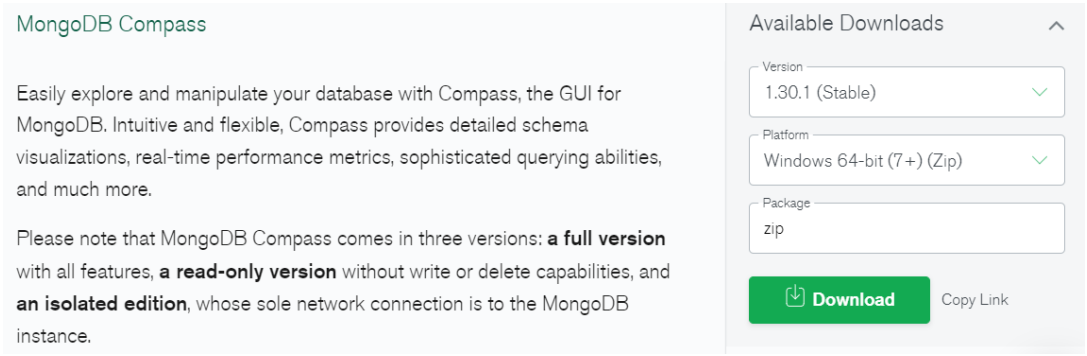
Para usar imágenes de contenedor Docker en Windows y WSL 2, es necesarios Docker Desktop, viene con mejoras en el uso compartido del sistema de archivos, el tiempo de arranque, etc. Está disponible en la página de docker.com

Se habilitó la integración con WSL 2 con la distribución de Ubuntu 20.04



MongoDB Compass

Para visualizar y gestionar la información de la base de datos MongoDB se usa MongoDB Compass que está disponible en la página web de mongodb.



Azure CLI

La CLI de Azure es una herramienta de línea de comandos para conectarse a Azure y ejecutar comandos administrativos en recursos de Azure.

Se instala en Ubuntu mediante el siguiente comando

```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

Anexo B: Levantamiento de contenedores para el desarrollo

Para el desarrollo y pruebas locales del proyecto se usó dos contenedores: emqx y mongo.

Para la conexión al bróker EMQ X y base de datos de MongoDB.

Se crea un archivo docker-compose.yml que tiene la siguiente configuración:

File: docker-compose.yml

```yaml
version: '3.8'
services:
  emqx:
    container_name: emqx
    image: emqx/emqx:4.3.11
    restart: always
    ports:
      - 18083:18083
      - 8083:8083
      - 1883:1883

  mongo:
    container_name: mongo_iot
    image: mongo:5.0.3
    restart: always
    ports:
      - 27020:27017
    volumes:
    - ./myData:/data/db
```

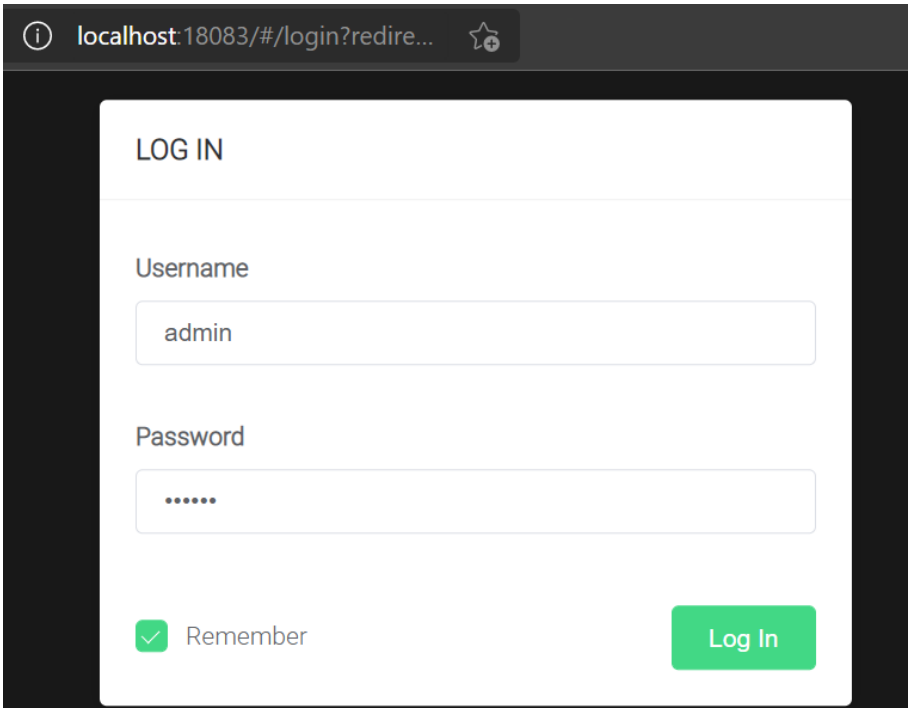
Levantamos los contenedores con el siguiente comando

```
~/api-rest-wsn-iot
docker-compose up -d
Creating network "api-rest-wsn-iot_default" with the default driver
Creating emqx      ... done
Creating mongo_iot ... done
```

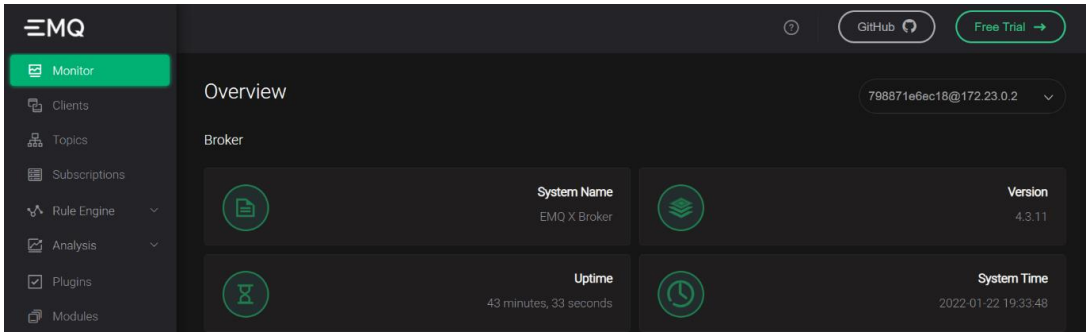
Verificamos los contenedores y sus puertos

```
❯ docker-compose ps
  Name                Command               State                          Ports
-------------------------------------------------------------------------------------------------------------
emqx       /usr/bin/docker-entrypoint ...   Up      11883/tcp, 0.0.0.0:18083->18083/tcp,
                                                     0.0.0.0:1883->1883/tcp, 4369/tcp, 4370/tcp, 5369/tcp,
                                                     6369/tcp, 6370/tcp, 8081/tcp, 0.0.0.0:8083->8083/tcp,
                                                     8084/tcp, 8883/tcp
mongo_iot  docker-entrypoint.sh mongod      Up      0.0.0.0:27020->27017/tcp
```

EMQ X nos provee una interfaz en el puerto 18083

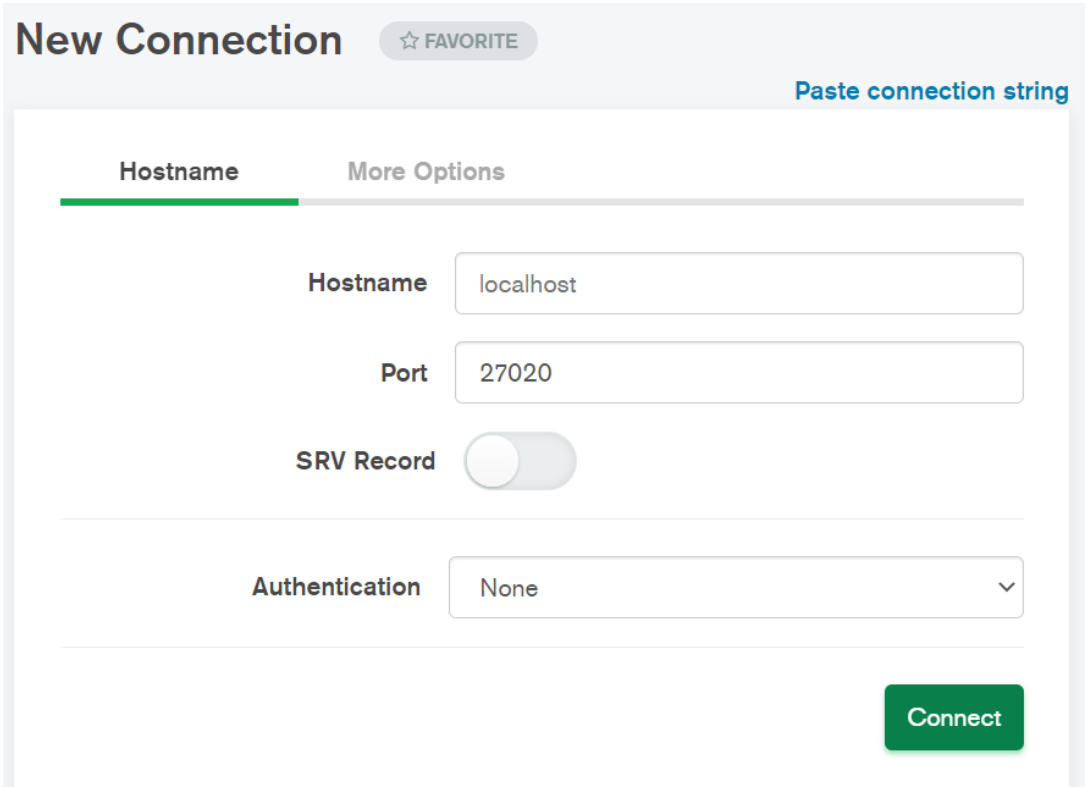


Este dashboard provee de varias opciones de visualización, configuración, plugins, además de mostrar los clientes, subscriptores, tópicos de las conexiones MQTT

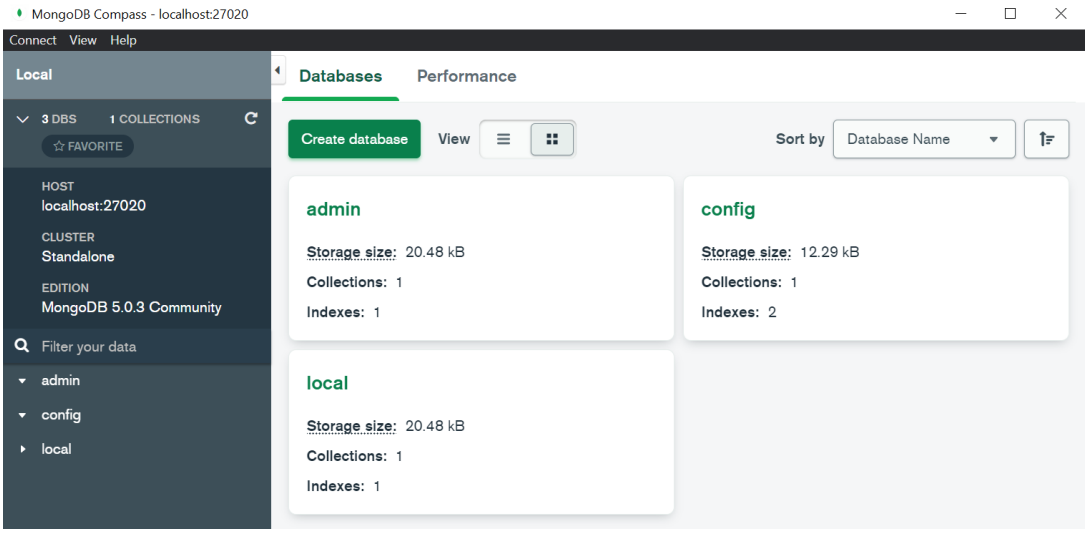


Anexo C: Conexión local a contenedor de MongoDB

Conexión al contenedor ya creado de mongo, esto sirve para el desarrollo local por lo cual no se manejan métodos de autenticación, pero si cuando se utilice los servicios en la nube.



MongoDB trae por defecto las siguientes colecciones



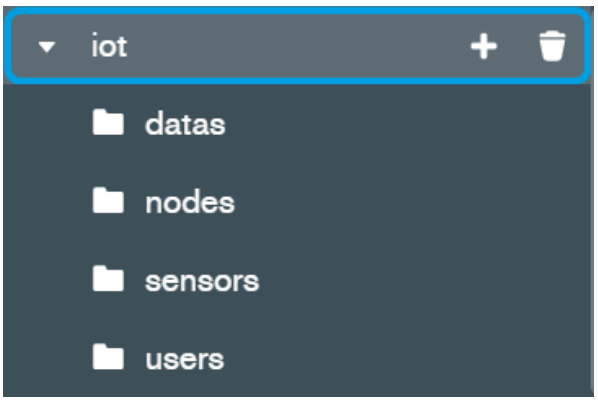
Para conectar la aplicación con esta base de datos es necesario declarar la URI en el directorio “config/config.js”

const URI = 'mongodb://localhost:27020/iot';

Si se ha logrado la conexión la consola de node muestra:

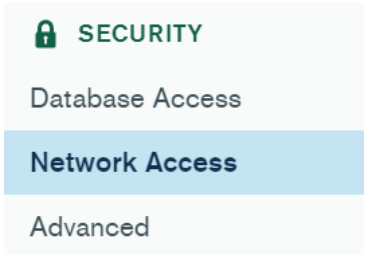
```
My port: 3000
[db] Conectada con éxito
```

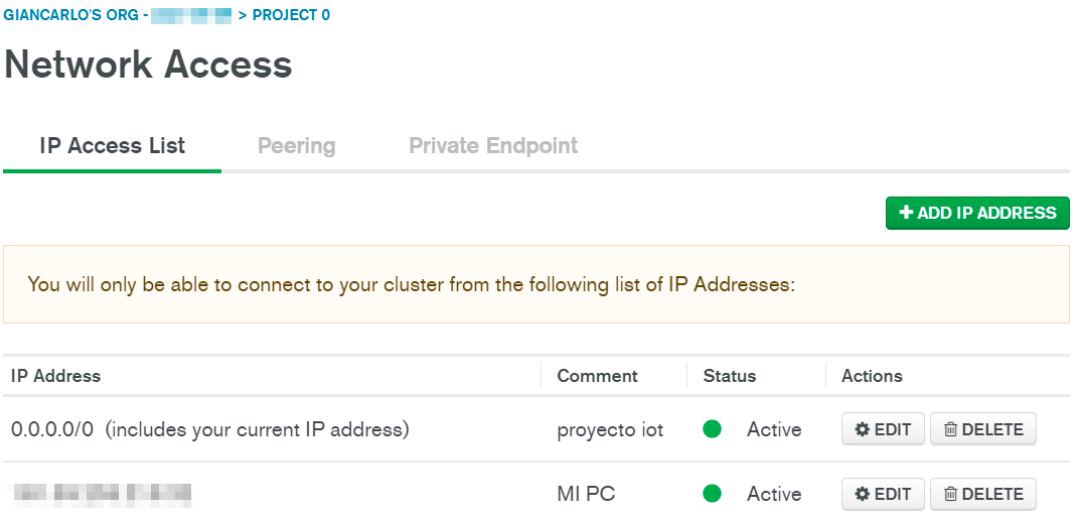
Y ha creado las siguientes colecciones con nombre de los modelos en plural:



Anexo D: Red de acceso MongoDB Atlas

Para acceder a la base de datos de MongoDB Atlas es necesario conceder el acceso a la IP

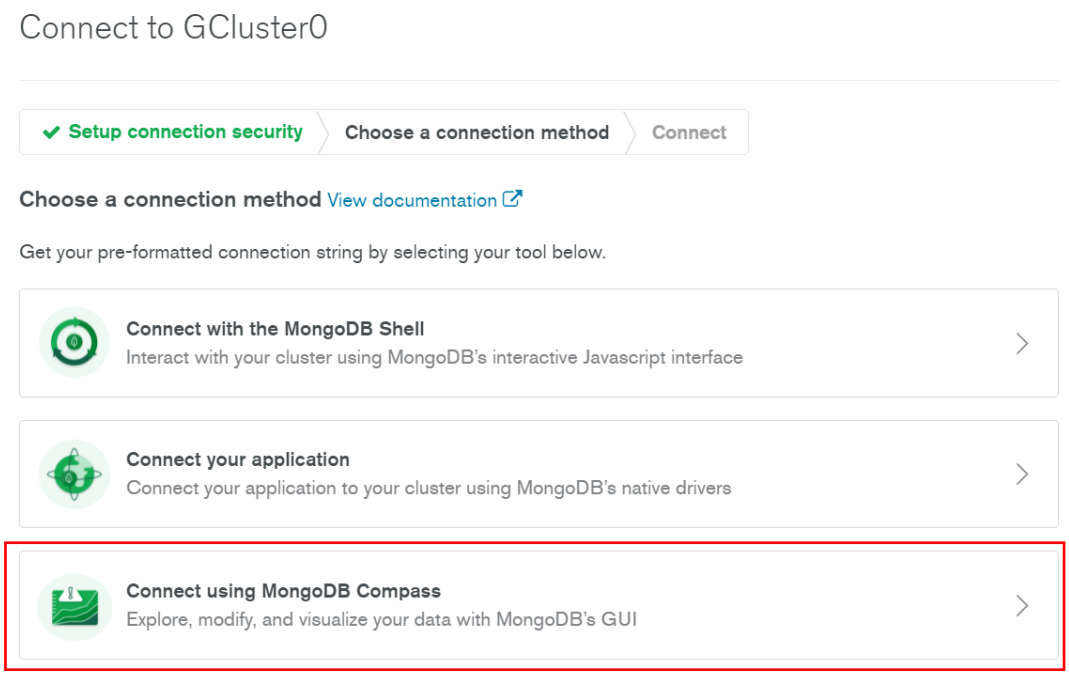




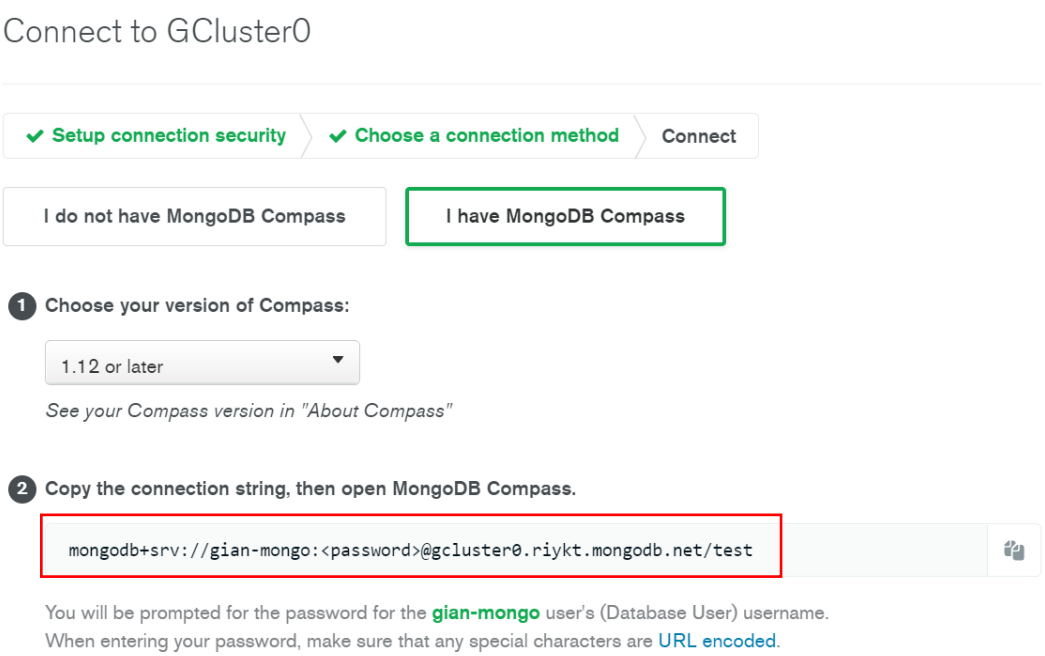
Anexo E: Conexión MongoDB Compass con MongoDB Atlas

Nos conectamos a nuestro cluster

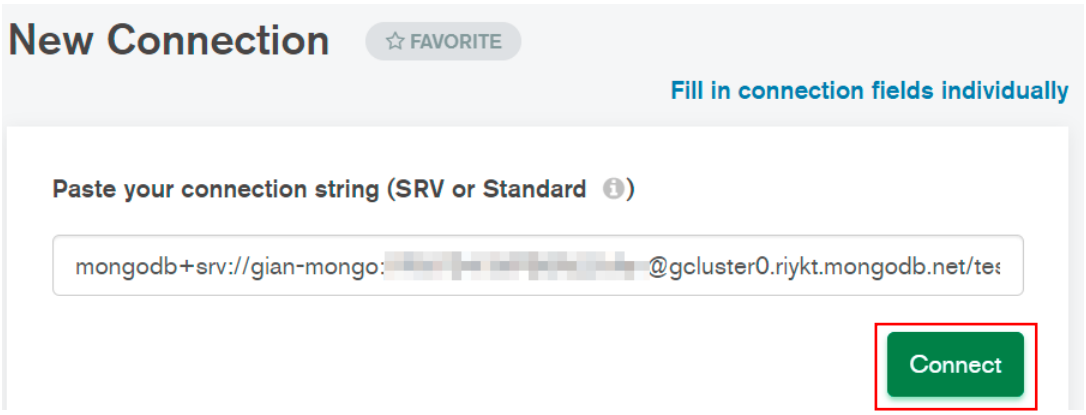
Usamos la opción MongoDB Compass



Copiamos el string de conexión

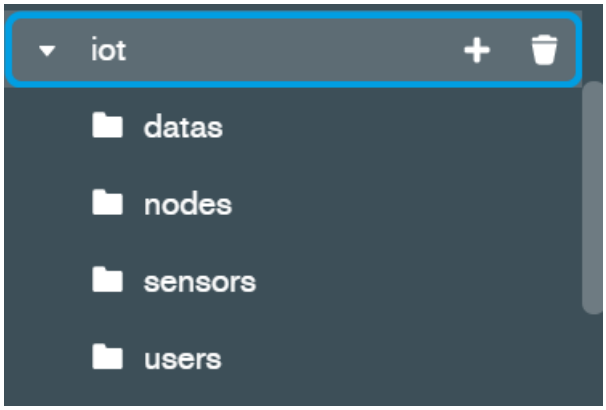


Pegamos el string de conexión y ponemos el campo password en la aplicación MongoDB Compass

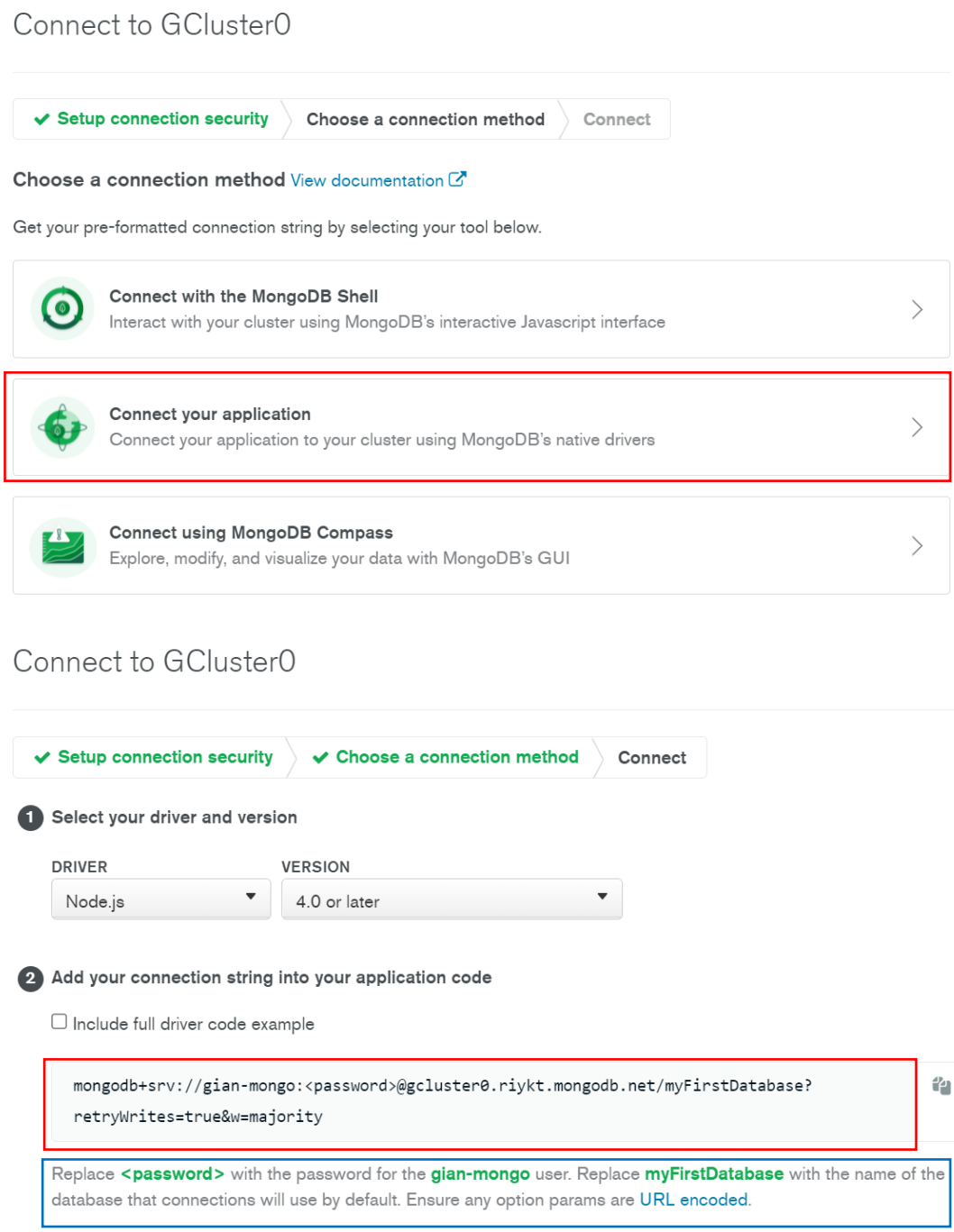


Al conectarse la aplicación

Creará las siguientes colecciones que corresponden a los modelos de los recursos.


iot
datas
nodes
sensors
users

Anexo F: Conexión aplicación con MongoDB Atlas



Anexo G: Ejemplo de documentación con swagger

Ruta de usuario

```
/**
 * @swagger
 * components:
 *  schemas:
 *    User:
 *      type: object
 *      properties:
 *        _id:
 *          type: string
 *          description: The unique identifier of the user
 *        name:
 *          type: string
 *          description: The name of the user
 *        email:
 *          type: string
 *          description: The email of the user
 *        password:
 *          type: string
 *          description: The password of the user
 *        enabled:
 *          type: boolean
 *          description: The enabled of the user
 *        role:
 *          type: string
 *          description: The role of the user
 *        createdAt:
 *          type: string
 *          format: date-time
 *          description: The date and time when the user was created
 *        updatedAt:
 *          type: string
 *          format: date-time
 *          description: The date and time when the user was updated
```

```
*      required:
*        - name
*        - email
*        - password
*        - role
*      example:
*        name: Nombre Apellido
*        email: the@email.com
*        role: user
*        password: "nombre12345"
*
*    UserDetails:
*      type: object
*      properties:
*        _id:
*          type: string
*          description: The unique identifier of the user
*        name:
*          type: string
*          description: The name of the user
*        email:
*          type: string
*          description: The email of the user
*        password:
*          type: string
*          description: The password of the user
```

```
*        enabled:
*          type: boolean
*          description: The enabled of the user
*        role:
*          type: string
*          description: The role of the user
*        createdAt:
*          type: string
*          format: date-time
*          description: The date and time when the user was created
*        updatedAt:
*          type: string
*          format: date-time
*          description: The date and time when the user was updated
```

```
*         nodes:
*           type: array
*           items:
*             type: object
*             properties:
*               _id:
*                 type: string
*                 description: The id of the node
*               name:
*                 type: string
*                 description: The name of the node
*               sensors:
*                 type: array
*                 items:
*                   type: object
*                   properties:
*                     id:
*                       type: string
*                       description: The id of the sensor
```

```
*     example:
*       _id: 61db294a76c09052c6dda367
*       name: Nombre Apellido
*       email: the@email.com
*       enabled: user
*       role: user
*       createdAt: 2022-01-09T18:28:50.218Z
*       updatedAt: 2022-01-09T18:28:50.218Z
*       nodes: [{id: "1", name: "nodo1", sensors: [{id: "1", name: "sensor1"}, {id: "2", name: "sensor2"}]}]
*
*  parameters:
*    UserId:
*      in: path
*      name: id
*      required: true
*      schema:
*        type: string
*      description: The user id
*
*  securitySchemes:
*    bearerAuth:
*      type: http
*      scheme: bearer
*      bearerFormat: JWT
*
* security:
* - bearerAuth: []
*
*/
```

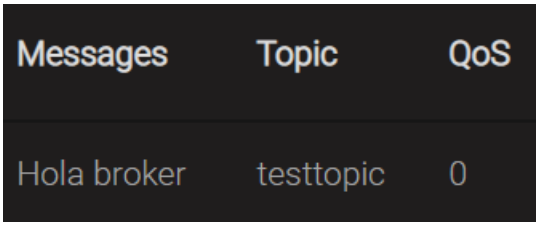
```
/**
 * @swagger
 * tags:
 *  name: Users
 *  description: Users endpoint
 */

/**
 * @swagger
 * /users:
 *  get:
 *    summary: Get all users
 *    tags: [Users]
 *    security:
 *      - bearerAuth: []
 *    responses:
 *      200:
 *        description: An array of users
 *        content:
 *          application/json:
 *            schema:
 *              type: array
 *              items:
 *                $ref: '#/components/schemas/User'
 *      401:
 *        description: Unauthorized
 *
 *
 *
 */
```

Anexo H: Pruebas de conectividad al bróker local y API REST local

Se publica mensajes al tópico “testopic” a la dirección ip donde esta levantado el contenedor local, para comprobar que la conexión se ha realizado y recibe los mensajes

```
$ mosquitto_pub -m "Hola broker" -t "testtopic" -h 192.168.100.161 -p 1883
```

El bróker recibe el mensaje



También se crea una suscripción a este tópico

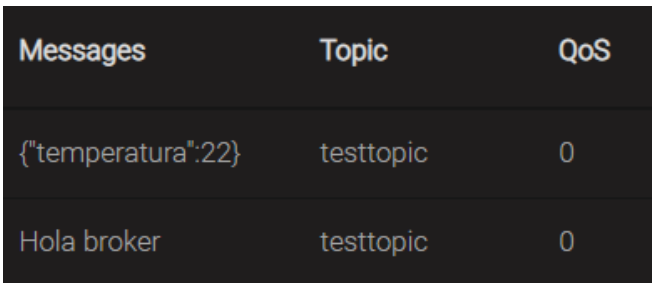
```
pi@raspberry1:~ $ mosquitto_sub -d -t "testtopic" -h 192.168.100.161 -p 1883
Client mosqsub|2033-raspberry1 sending CONNECT
Client mosqsub|2033-raspberry1 received CONNACK (0)
Client mosqsub|2033-raspberry1 sending SUBSCRIBE (Mid: 1, Topic: testtopic, QoS: 0)
Client mosqsub|2033-raspberry1 received SUBACK
Subscribed (mid: 1): 0
Client mosqsub|2033-raspberry1 received PUBLISH (d0, q0, r0, m0, 'testtopic', ... (11 bytes))
Hola broker
```

Se verifica la conexión y que los clientes pueden publicar y suscribirse

Ahora se simula un envió de datos de un sensor con su id “/61e59941f9fce835bfd3dcc4/data”

```
pi@raspberry1:~ $ mosquitto_pub -m "{\"temperatura\":22}" -t "/61e59941f9fce8
35bfd3dcc4/data" -h 192.168.100.161 -p 1883
```

El bróker recibe el mensaje

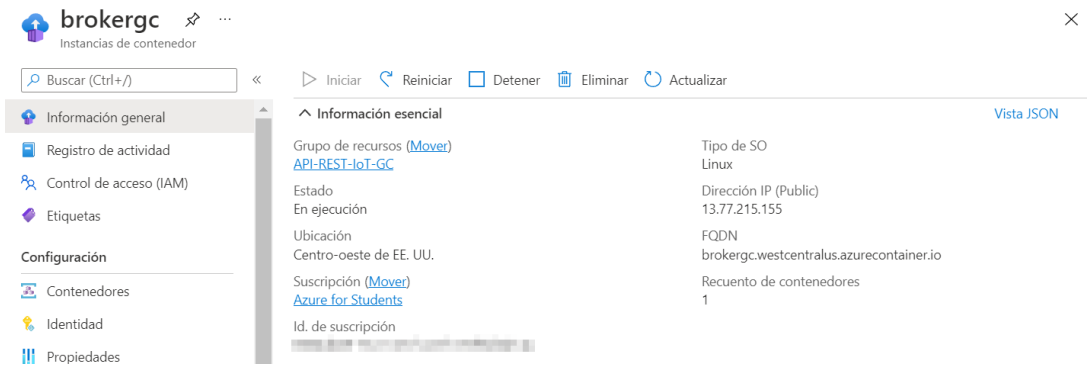


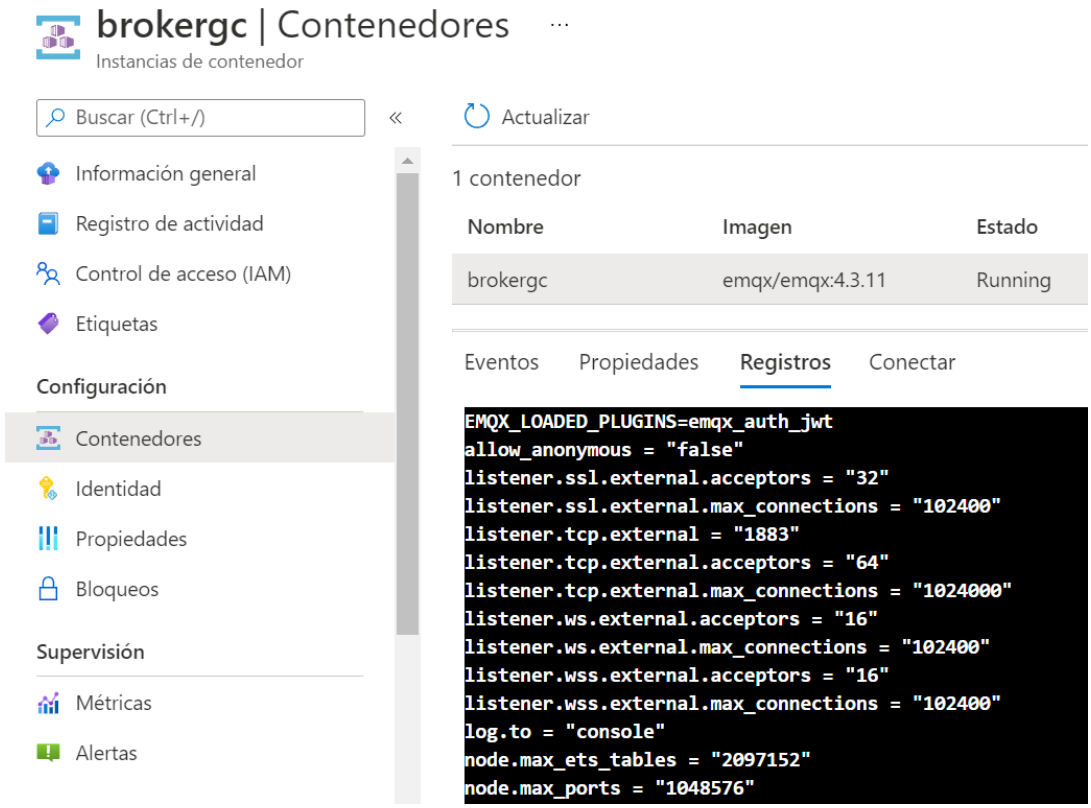
```
pi@raspberry1:~ $ mosquitto_sub -d -t "/61e59941f9fce835bfd3dcc4/data" -h 192.168
.100.161 -p 1883
Client mosqsub|2047-raspberry1 sending CONNECT
Client mosqsub|2047-raspberry1 received CONNACK (0)
Client mosqsub|2047-raspberry1 sending SUBSCRIBE (Mid: 1, Topic: /61e59941f9fce83
5bfd3dcc4/data, QoS: 0)
Client mosqsub|2047-raspberry1 received SUBACK
Subscribed (mid: 1): 0
Client mosqsub|2047-raspberry1 received PUBLISH (d0, q0, r0, m0, '/61e59941f9fce8
35bfd3dcc4/data', ... (18 bytes))
{"temperatura":22}
```

La aplicación que también esta suscrita le llega el mensaje:

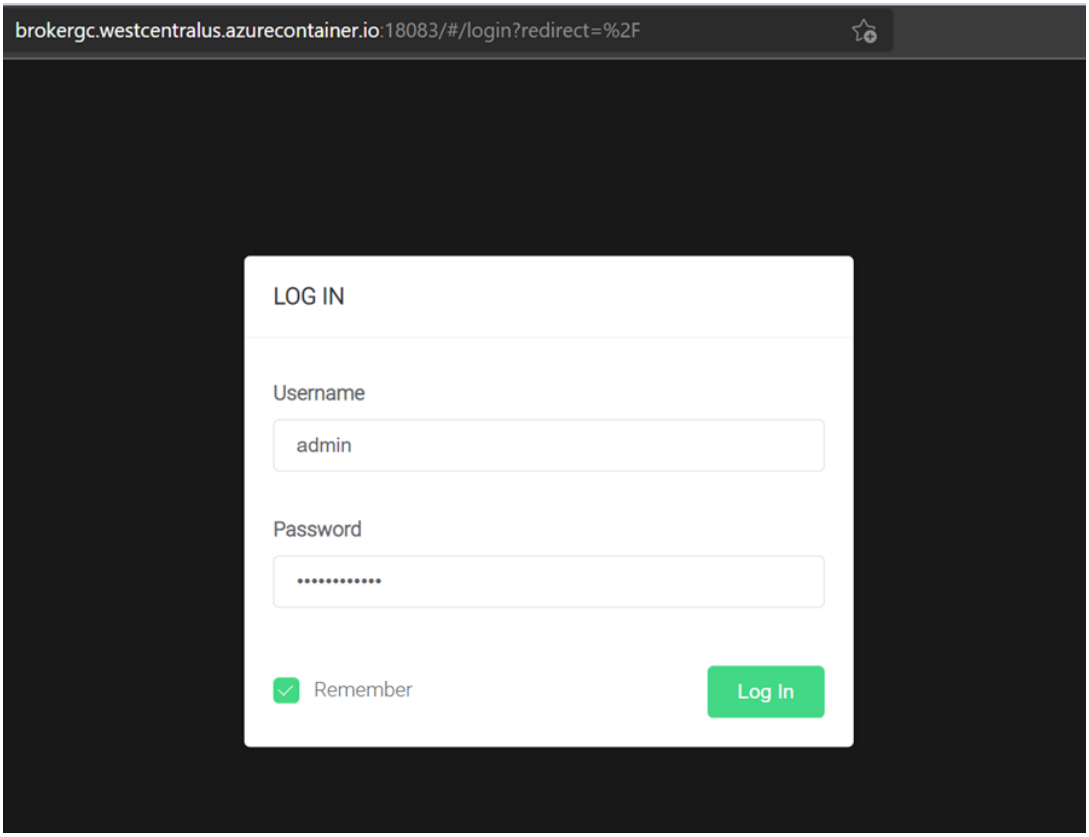
```
topic: /61e59941f9fce835bfd3dcc4/data
message: {"temperatura":22}
```

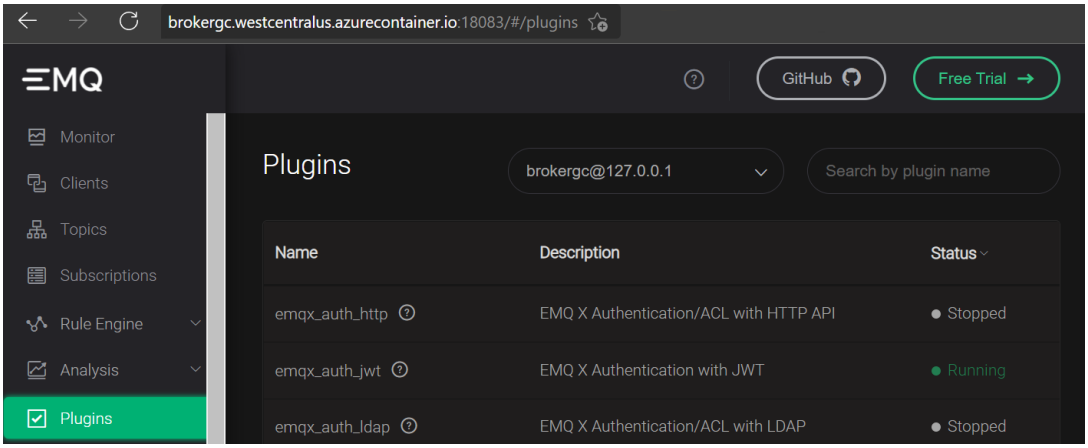
Anexo I: Resultado despliegue contenedor bróker MQTT en Azure Container Instances





```
node.max_ports = "1048576"
node.name = "brokergc@127.0.0.1"
node.process_limit = "2097152"
rpc.port_discovery = "manual"
Starting emqx on node brokergc@127.0.0.1
Start mqtt:tcp:internal listener on 127.0.0.1:11883 successfully.
Start mqtt:tcp:external listener on 0.0.0.0:1883 successfully.
Start mqtt:ws:external listener on 0.0.0.0:8083 successfully.
Start mqtt:ssl:external listener on 0.0.0.0:8883 successfully.
Start mqtt:wss:external listener on 0.0.0.0:8084 successfully.
Start http:management listener on 8081 successfully.
Start http:dashboard listener on 18083 successfully.
EMQ X Broker 4.3.11 is running now!
```




brokergc.westcentralus.azurecontainer.io:18083/#/plugins
EMQ
GitHub
Free Trial
Monitor
Clients
Topics
Subscriptions
Rule Engine
Analysis
Plugins
Plugins
brokergc@127.0.0.1
Search by plugin name
Name
Description
Status
emqx_auth_http
EMQ X Authentication/ACL with HTTP API
Stopped
emqx_auth_jwt
EMQ X Authentication with JWT
Running
emqx_auth_ldap
EMQ X Authentication/ACL with LDAP
Stopped

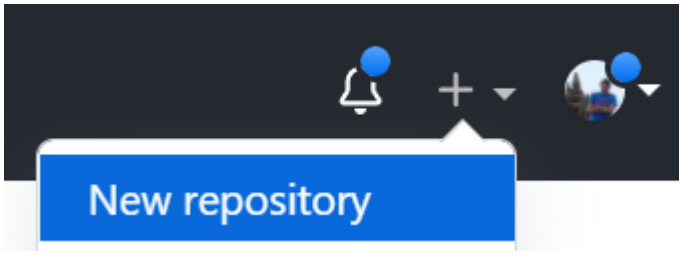
Anexo J: Proceso para subir el código del proyecto al repositorio de GitHub

Una vez que se tiene git instalado e iniciado el repositorio (ANEXO A)

Se debe realizar los commits del proyecto con:

git commit -m “Comentario”

Para utilizar github se necesita una cuenta, se utilizó la cuenta del investigador para subir el código en un nuevo repositorio de github



Se utiliza el repositorio con nombre: API-REST-WSN-IOT

A continuación, github da los comandos para subir el código:

Se cambia el nombre de la rama:

git branch -M main

Se añade el origen del repositorio

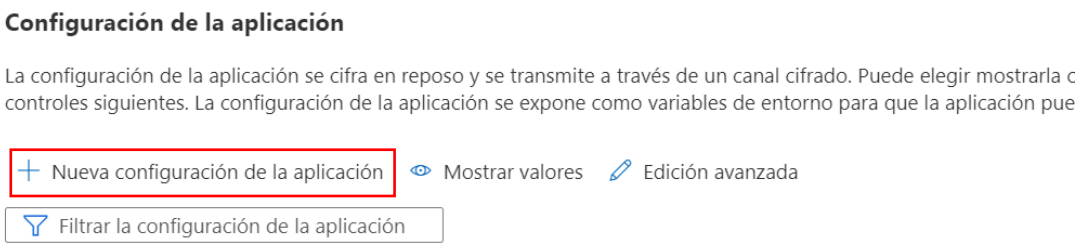
git remote add origin link

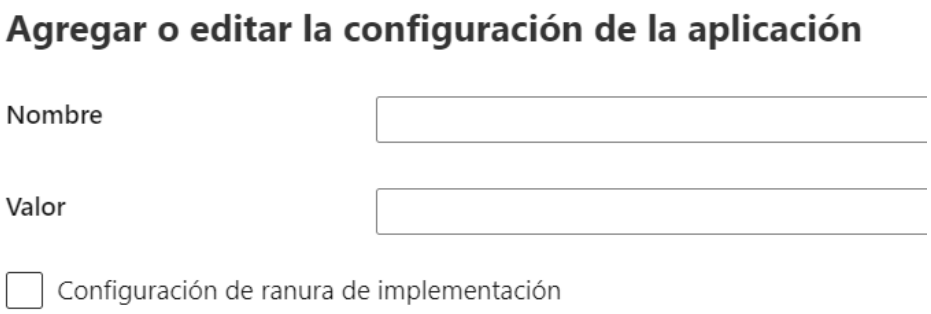
Se sube el código de la rama main

git push -u origin main

Anexo K: Configuración de la aplicación en App Service

Para la configuración de la aplicación se crean las mismas variables de entorno con los valores de la base de datos de MongoDB Atlas





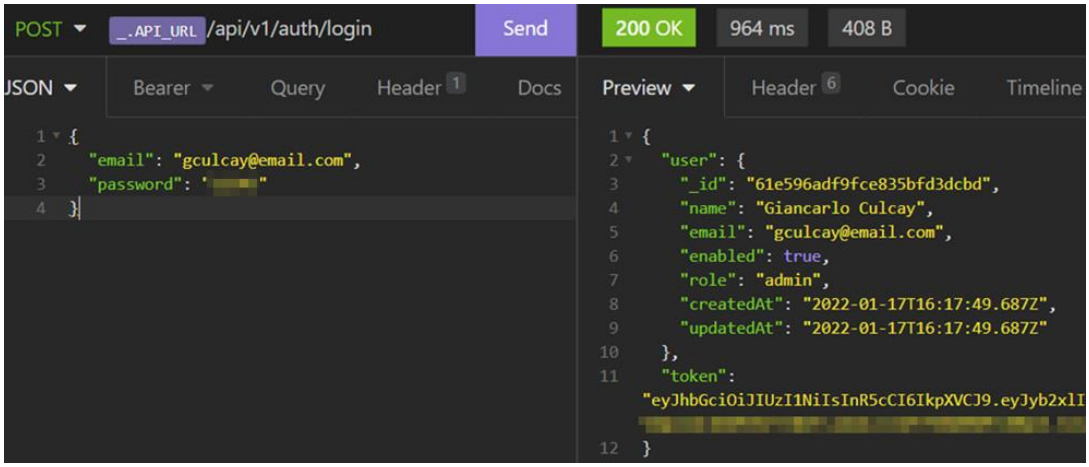
Resultado:

Nombre	Valor	Origen
DB	El valor está oculto. Haga clic para mostr	Configuración del servicio de la aplicació
JWT_SECRET	El valor está oculto. Haga clic para mostr	Configuración del servicio de la aplicació
MQTT_HOST	El valor está oculto. Haga clic para mostr	Configuración del servicio de la aplicació
MQTT_PASSWORD	El valor está oculto. Haga clic para mostr	Configuración del servicio de la aplicació
MQTT_PORT	El valor está oculto. Haga clic para mostr	Configuración del servicio de la aplicació
MQTT_USER	El valor está oculto. Haga clic para mostr	Configuración del servicio de la aplicació
URL_BASE	El valor está oculto. Haga clic para mostr	Configuración del servicio de la aplicació

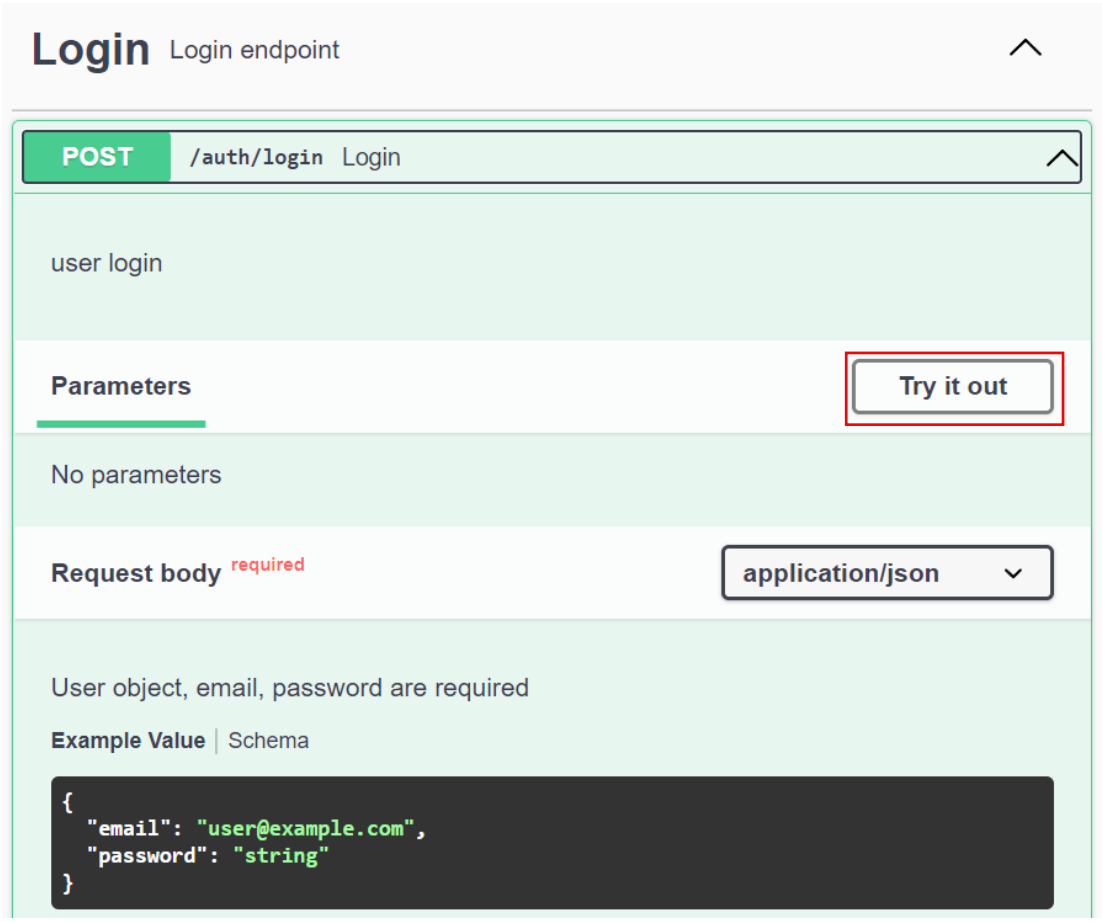
Anexo L: Login de usuario y uso de token

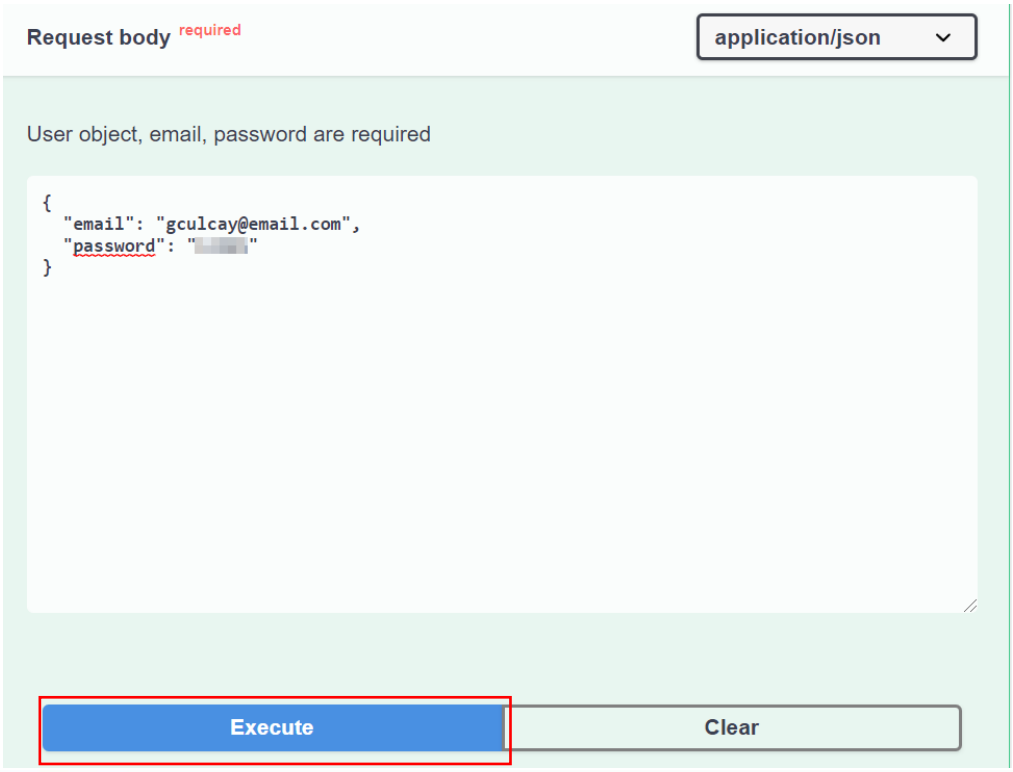
El acceso de usuario se realiza mediante el método POST al endpoint: "/auth/login", si los campos "email" y "password" son correctos el servidor responde con el usuario y el token de acceso.

Ejemplo de login usando Insomnia



Ejemplo de login usando la documentación interactiva de swagger:





Responses

Curl

```
curl -X 'POST' \
  'https://api-rest-wsn-iot.azurewebsites.net/api/v1/auth/login' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "email": "gculcay@email.com",
  "password": '[redacted]'
}'
```

Request URL

```
https://api-rest-wsn-iot.azurewebsites.net/api/v1/auth/login
```

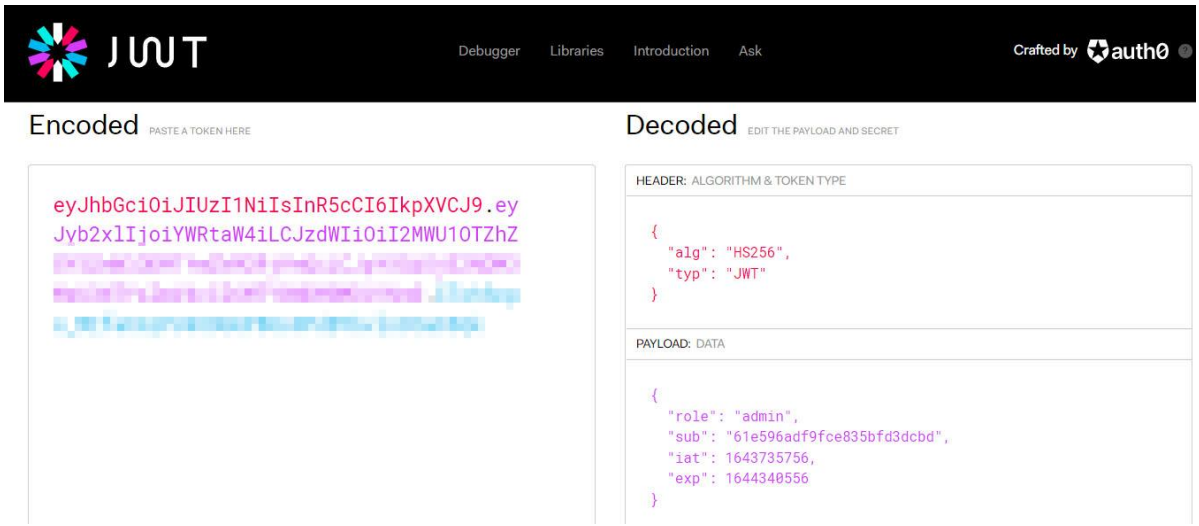
Server response

Code	Details
200	Response body

```
{
  "user": {
    "_id": "61e596adf9fce835bfd3dcbd",
    "name": "Giancarlo Culcay",
    "email": "gculcay@email.com",
    "enabled": true,
    "role": "admin",
    "createdAt": "2022-01-17T16:17:49.687Z",
    "updatedAt": "2022-01-17T16:17:49.687Z"
  },
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiYWRtaW4iLCJzdWIi
[redacted]
}
```

Download

El token de acceso tiene el siguiente payload y expiración



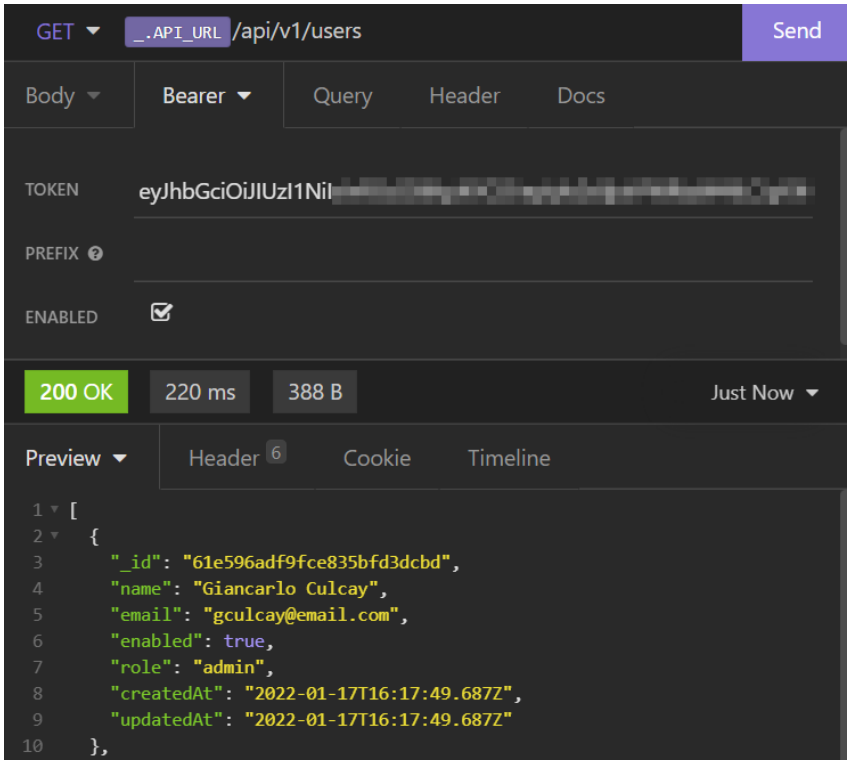
Para usar los endpotins de swagger es necesario proporcionar el token:

Ejemplo de login usando curl

```
❯ curl -X POST -H "Content-Type: application/json" -d '{"email":"gculcay@email.com","password":"[redacted]"}'
https://api-rest-wsn-iot.azurewebsites.net/api/v1/auth/login
{"user":{"_id":"61e596adf9fce835bfd3dcbd","name":"Giancarlo Culcay","email":"gculcay@email.com","enabled"
:true,"role":"admin","createdAt":"2022-01-17T16:17:49.687Z","updatedAt":"2022-01-17T16:17:49.687Z"},"toke
n":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiYWRtaW4iLCJzdWIiOiI2MWU1OTZhZ[redacted]
```

Todos estos métodos nos entregan un token de acceso solo para este usuario y este sirve como requisito para el acceso a los demás recursos

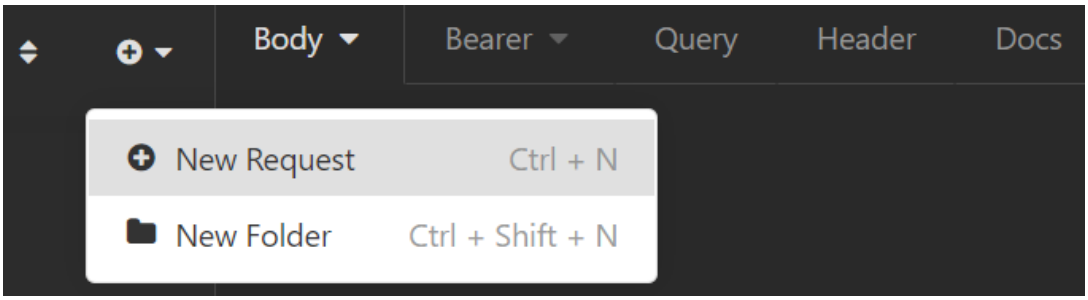
Ejemplo de uso de token en Insomnia:


GET
_.API_URL
/api/v1/users
Send
Body
Bearer
Query
Header
Docs
TOKEN
eyJhbGciOiJIUzI1Nil
PREFIX
ENABLED
200 OK
220 ms
388 B
Just Now
Preview
Header 6
Cookie
Timeline
[
{
"_id": "61e596adf9fce835bfd3dcbd",
"name": "Giancarlo Culcay",
"email": "gculcay@email.com",
"enabled": true,
"role": "admin",
"createdAt": "2022-01-17T16:17:49.687Z",
"updatedAt": "2022-01-17T16:17:49.687Z"
},

Anexo M: Pruebas de API REST con Insomnia

Para simplificar la url a la que se realiza las solicitudes se usa esta característica de Insomnia:

Creamos las solicitudes:

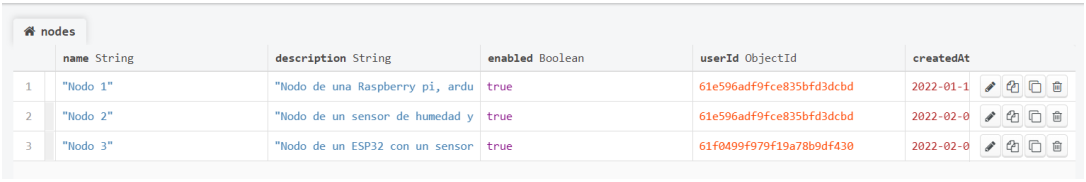


Anexo N: Colección usuarios en MongoDB Atlas

Colección usuarios


```
_id: ObjectId("61e596adf9fce835bfd3dcbd")
name: "Giancarlo Culcay"
email: "gculcay@email.com"
password: "$2b$10$wokAWnwyL73[redacted]"
enabled: true
role: "admin"
createdAt: 2022-01-17T16:17:49.687+00:00
updatedAt: 2022-01-17T16:17:49.687+00:00

_id: ObjectId("61f0499f979f19a78b9df430")
name: "Usuario X"
email: "usuariox@email.com"
password: "$2b$10$scIPVaRO[redacted]"
enabled: true
role: "user"
createdAt: 2022-01-25T19:03:59.976+00:00
updatedAt: 2022-01-25T19:03:59.976+00:00
```


Colección nodos



nodes

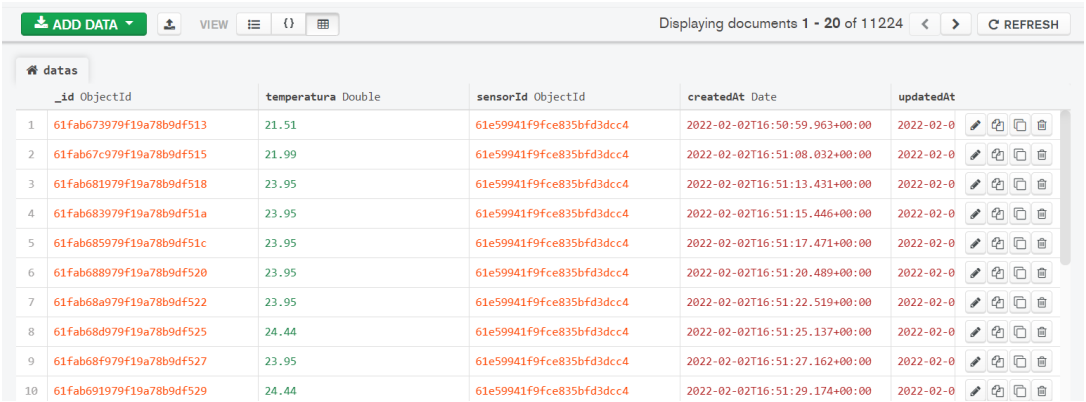
	name String	description String	enabled Boolean	userId ObjectId	createdAt
1	"Nodo 1"	"Nodo de una Raspberry pi, ardu	true	61e596adf9fce835bfd3dcbd	2022-01-1
2	"Nodo 2"	"Nodo de un sensor de humedad y	true	61e596adf9fce835bfd3dcbd	2022-02-0
3	"Nodo 3"	"Nodo de un ESP32 con un sensor	true	61f0499f979f19a78b9df430	2022-02-0

Colección sensores

sensors

	name String	enabled Boolean	nodeId ObjectId	createdAt Date	updatedAt
1	"Temperatura 1"	true	61e5981bf9fce835bfd3dcc1	2022-01-17T16:28:49.130+00:00	2022-02-02
2	"Sensor de temperatura y humeda	true	61fb408e979f19a78b9df5b3	2022-02-03T02:42:22.417+00:00	2022-02-03
3	"Sensores"	true	61fc1e2c2f9ecd80d40a034a	2022-02-03T18:28:24.415+00:00	2022-02-03

Colección datos



ADD DATA · VIEW · Displaying documents 1 - 20 of 11224 · REFRESH

datas

	_id ObjectId	temperatura Double	sensorId ObjectId	createdAt Date	updatedAt
1	61fab673979f19a78b9df513	21.51	61e59941f9fce835bfd3dcc4	2022-02-02T16:50:59.963+00:00	2022-02-0
2	61fab67c979f19a78b9df515	21.99	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:08.032+00:00	2022-02-0
3	61fab681979f19a78b9df518	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:13.431+00:00	2022-02-0
4	61fab683979f19a78b9df51a	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:15.446+00:00	2022-02-0
5	61fab685979f19a78b9df51c	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:17.471+00:00	2022-02-0
6	61fab688979f19a78b9df520	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:20.489+00:00	2022-02-0
7	61fab68a979f19a78b9df522	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:22.519+00:00	2022-02-0
8	61fab68d979f19a78b9df525	24.44	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:25.137+00:00	2022-02-0
9	61fab68f979f19a78b9df527	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:27.162+00:00	2022-02-0
10	61fab691979f19a78b9df529	24.44	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:29.174+00:00	2022-02-0

Anexo O: Código fuente Raspberry PI para el Nodo 1

datos_pi.py

```
import serial, json
import paho.mqtt.client as mqtt

broker = 'brokergc.westcentralus.azurecontainer.io'
port = 1883
topic_pub = '/61e59941f9fce835bfd3dcc4/data'
topic_sub = '/61e59941f9fce835bfd3dcc4/config'
client_id = 'pi-python-mqtt--01'
username = 'pi'
password = 'eyJhbGciOiJ… '
serial_port = serial.Serial("/dev/ttyACM0", 9600)
print("Estableciendo conexion serial con Arduino ", serial_port)

def on_connect(client, userdata, flags, rc):
    print("Se conecto con broker mqtt")
    client.subscribe(topic_sub)


def on_message(client, userdata, msg):
    my_msg=str(msg.payload.decode("utf-8"))
    print("Mensaje recibido: ", my_msg)
    serial_port.write(my_msg.encode())


client = mqtt.Client(client_id)
client.username_pw_set(username, password)
client.on_connect = on_connect
client.on_message = on_message

client.connect(broker, port, 60)

while True:
    client.loop()
    if serial_port.inWaiting() > 0:
        data  = serial_port.readline().decode('utf-8')
        data = float(data.rstrip())
        print("Mensaje recibido: ", data)
        my_data = {'temperatura': data}
        client.publish(topic_pub, payload=json.dumps(my_data), qos=0,
retain=False)

serial_port.close()
print("Conexion serial cerrada")
```

Anexo P: Código fuente Arduino UNO para el Nodo 1

```
#include <LiquidCrystal.h>
#include <ArduinoJson.h>

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

float temperatura;
unsigned long start=0;
int led_verde=7;
int led_rojo=8;
int S1=0;          //sensor de temperatura
int time_ms=8000; //tiempo de muestreo por defecto

void setup() {
  Serial.begin(9600);
  pinMode(led_verde, OUTPUT);
  pinMode(led_rojo, OUTPUT);

  lcd.begin(16,2);
  lcd.print("Bienvenido");
  delay(1000);

  lcd.clear();
}

void loop() {
  lcd.setCursor(0, 0);

    if(millis() > start + time_ms){
      start = millis();
      S1=analogRead(0);
      temperatura= (500.0*S1)/1023.0;
      Serial.println(temperatura);
      lcd.print("temp=");
      lcd.print(temperatura);
    }

  if(Serial.available()>0){
    lcd.setCursor(9, 0);
    String payload = Serial.readString();

    StaticJsonDocument<256> doc;
    DeserializationError err = deserializeJson(doc, payload);
    if (err){
      Serial.println("ocurrio un error");
      Serial.println(err.c_str());
```

```
      return;
    }

    bool led = doc["led"];
    time_ms = doc["time_ms"] | time_ms;
    String mensaje = doc["mensaje"] | "";

    if(led==1){
      digitalWrite(led_verde,HIGH);
      digitalWrite(led_rojo,LOW);
    }
    if(led==0){
      digitalWrite(verde,LOW);
      digitalWrite(led_rojo,HIGH);
    }
    if(mensaje.length()>0){
      lcd.setCursor(0, 1);
      lcd.print(mensaje);
    }
   }
}
```

Anexo Q: Código html para visualizar datos del Nodo 1

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.2/dist/css/bootstr
ap.min.css" rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
    <title>Datos Nodo 1</title>
</head>
<body>

    <div class="container mt-4 shadow-lg p3 mb-5 bg-body rounded">
        <h2 class="text-center">Nodo 1</h2>

        <p><b>SensorId: </b>61e59941f9fce835bfd3dcc4</p>

        <b>Datos:</b>
        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th>id</th>
                    <th>temperatura</th>
                    <th>fecha</th>
                </tr>
            </thead>
            <tbody id="datos-usuarios">
            </tbody>

        </table>
    </div>

    <script>
    async function obtenerData() {
        try {
            const url = 'https://api-rest-wsn-
iot.azurewebsites.net/api/v1/sensors/61e59941f9fce835bfd3dcc4/data?
limit=20';
            const response = await fetch(url, {
            method: 'GET',
            headers: {
                'Authorization': 'Bearer eyJhbGciOiJIUzI1NiIsInR…'
```

```
            }
            })
            const data = await response.json()
            return data
        } catch (error) {
            console.log(error);
        }
    }

    async function mostrarDatos(){
        try {
            let data = await obtenerData()
            console.log('datos', data)
            let contenido = ''
            for (let x of data) {
                contenido
+=  `<tr><td>${x._id}</td><td>${x.temperatura}</td><td>${new
Date(x.createdAt).toLocaleString()}</td></tr>`
            }
            document.getElementById('datos-usuarios').innerHTML =
contenido
        } catch (error) {
            console.log(error);
        }
    }
    mostrarDatos()
    </script>

</body>
</html>
```

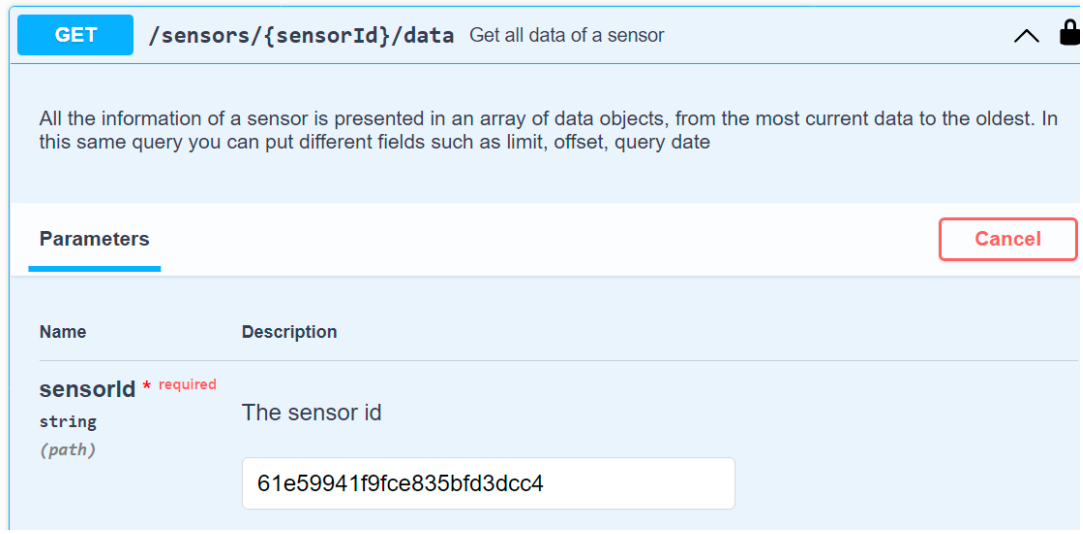
Anexo R: Respuestas Nodo 1

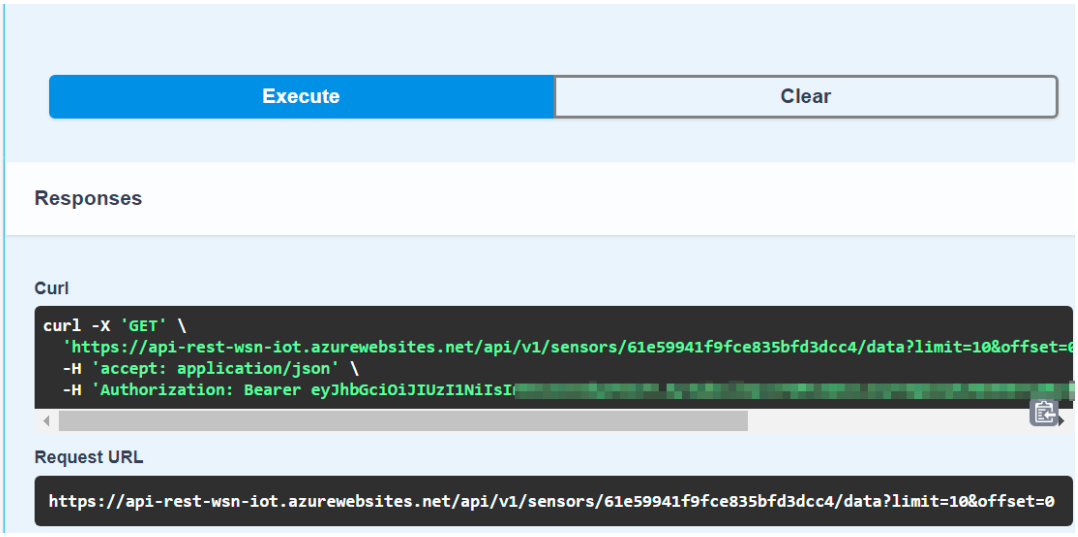
La salida del archivo datos_pi.py muestra los primeros datos medidos hasta los últimos

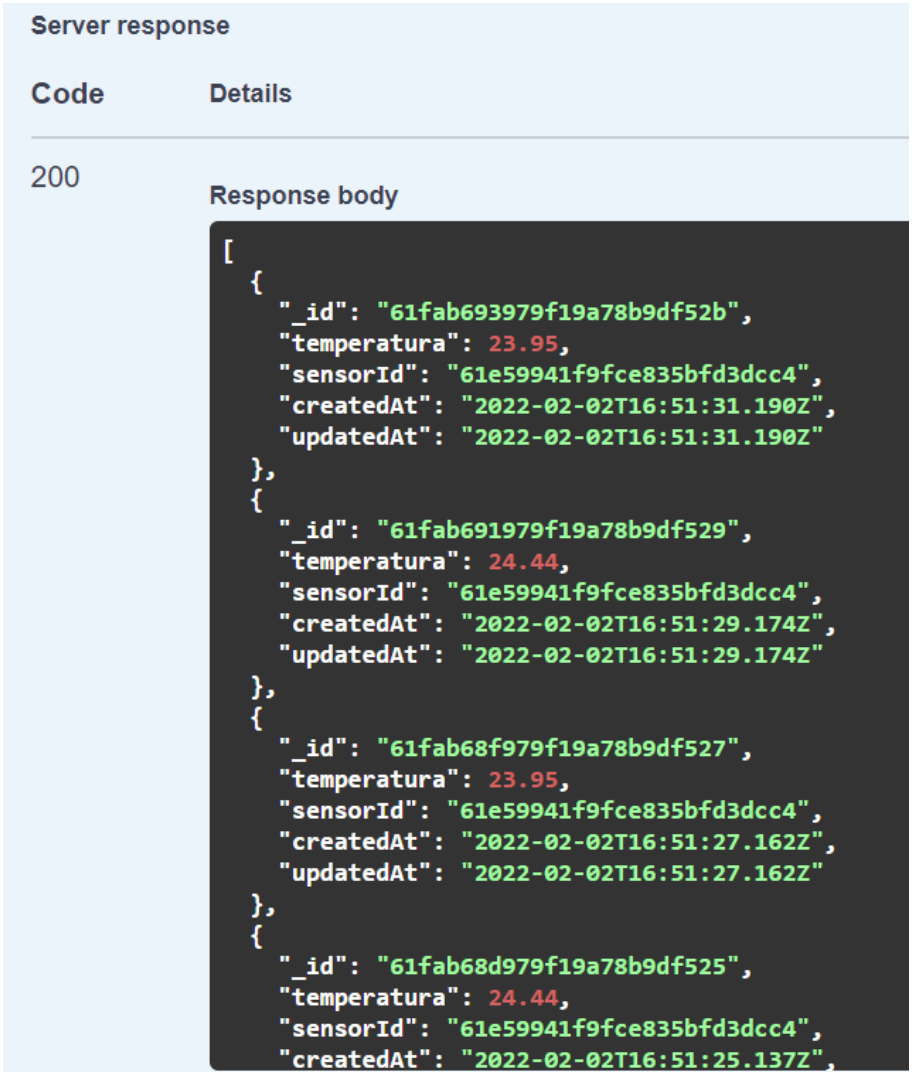
```
Se conecto con broker mqtt
Mensaje recibido:  21.51
Mensaje recibido:  21.99
Mensaje recibido:  {"led":1,"time_ms":2000,"mensaje":"Continuar "}
Mensaje recibido:  23.95
Mensaje recibido:  23.95
Mensaje recibido:  23.95
Mensaje recibido:  24.44
Mensaje recibido:  23.95
Mensaje recibido:  23.95
Mensaje recibido:  {"led":0,"time_ms":2000,"mensaje":"Detenerse "}
Mensaje recibido:  24.44
Mensaje recibido:  23.95
Mensaje recibido:  24.44
Mensaje recibido:  23.95
```

Insomnia, muestra el array de datos desde los últimos datos en sus primeras posiciones, según programación de este endpoint

Mediante la documentación de swagger también se pueden obtener los datos.







La siguiente página html, corresponde al código del Anexo Q y muestra los datos de la API

Nodo 1

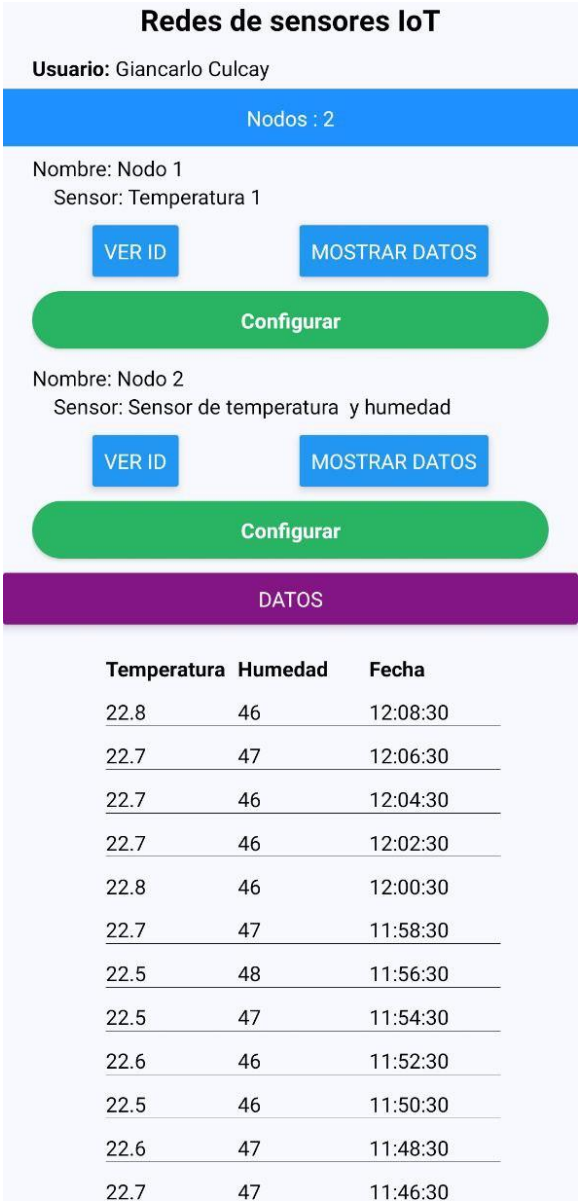
SensorId: 61e59941f9fce835bfd3dcc4

Datos:

id	temperatura	fecha
61fab693979f19a78b9df52b	23.95	2/2/2022 11:51:31
61fab691979f19a78b9df529	24.44	2/2/2022 11:51:29
61fab68f979f19a78b9df527	23.95	2/2/2022 11:51:27
61fab68d979f19a78b9df525	24.44	2/2/2022 11:51:25
61fab68a979f19a78b9df522	23.95	2/2/2022 11:51:22
61fab688979f19a78b9df520	23.95	2/2/2022 11:51:20
61fab686979f19a78b9df51e	24.44	2/2/2022 11:51:18
61fab685979f19a78b9df51c	23.95	2/2/2022 11:51:17
61fab683979f19a78b9df51a	23.95	2/2/2022 11:51:15
61fab681979f19a78b9df518	23.95	2/2/2022 11:51:13
61fab67c979f19a78b9df515	21.99	2/2/2022 11:51:08
61fab673979f19a78b9df513	21.51	2/2/2022 11:50:59

También se imprimen los datos en consola:

En la app desarrollada en React Native que consume los datos de la API, se puede apreciar:



Para el envió de parámetros, se debe seleccionar el botón configurar:

Se puede comprobar los datos almacenados en la base de datos de MongoDB Atlas

	_id ObjectId	temperatura Double	sensorId ObjectId	createdAt Date	updatedAt
3	61fab681979f19a78b9df518	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:13.431+00:00	2022-02-0
4	61fab683979f19a78b9df51a	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:15.446+00:00	2022-02-0
5	61fab685979f19a78b9df51c	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:17.471+00:00	2022-02-0
6	61fab686979f19a78b9df51e	24.44	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:18.488+00:00	2022-02-0
7	61fab688979f19a78b9df520	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:20.489+00:00	2022-02-0
8	61fab68a979f19a78b9df522	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:22.519+00:00	2022-02-0
9	61fab68d979f19a78b9df525	24.44	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:25.137+00:00	2022-02-0
10	61fab68f979f19a78b9df527	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:27.162+00:00	2022-02-0
11	61fab691979f19a78b9df529	24.44	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:29.174+00:00	2022-02-0
12	61fab693979f19a78b9df52b	23.95	61e59941f9fce835bfd3dcc4	2022-02-02T16:51:31.190+00:00	2022-02-0

A continuación, las consultas de usuario con su id, muestra los siguientes detalles

GET _.API_URL /api/v1/users/61e596adf9fce835bfd3dcbd Send

200 OK 454 ms 467 B Just Now

Preview Header 6 Cookie Timeline


```
{
  "_id": "61e596adf9fce835bfd3dcbd",
  "name": "Giancarlo Culcay",
  "email": "gculcay@email.com",
  "enabled": true,
  "role": "admin",
  "createdAt": "2022-01-17T16:17:49.687Z",
  "updatedAt": "2022-01-17T16:17:49.687Z",
  "nodes": [
    {
      "_id": "61e5981bf9fce835bfd3dcc1",
      "name": "Nodo 1",
      "sensors": [
        {
          "_id": "61e59941f9fce835bfd3dcc4",
          "name": "Temperatura 1"
        }
      ]
    },
    {
      "_id": "61fb408e979f19a78b9df5b3",
      "name": "Nodo 2",
      "sensors": [
        {
          "_id": "61fb410e979f19a78b9df5b5",
          "name": "Sensor de temperatura  y humedad"
        }
      ]
    }
```

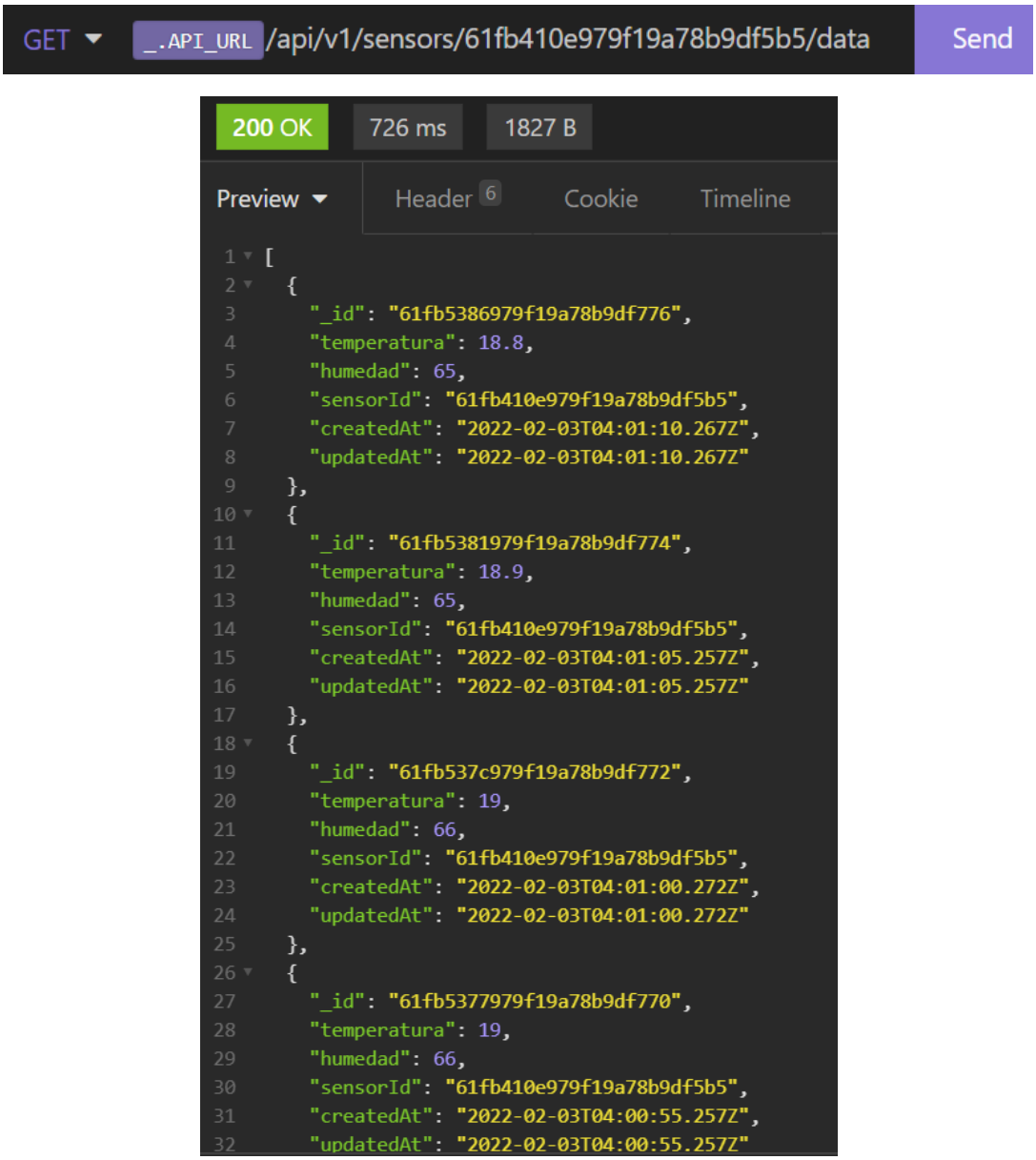

GET _.API_URL /api/v1/nodes/61e5981bf9fce835bfd3dcc1 Send

200 OK 281 ms 405 B Just Now

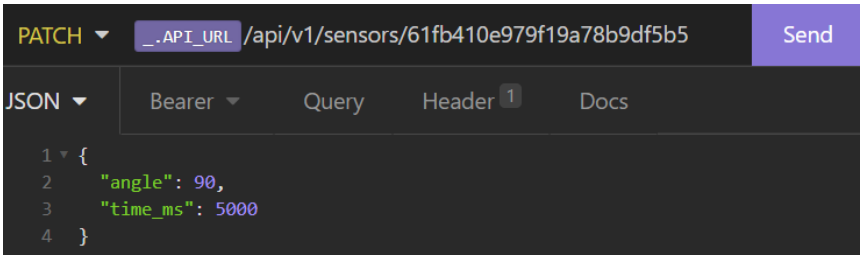
Preview Header 6 Cookie Timeline


```
{
  "_id": "61e5981bf9fce835bfd3dcc1",
  "location": "Pelileo",
  "name": "Nodo 1",
  "description": "Nodo de una Raspberry pi, arduino uno, lcd, sensor de temperatura y leds indicadores",
  "enabled": true,
  "userId": {
    "_id": "61e596adf9fce835bfd3dcbd",
    "name": "Giancarlo Culcay"
  },
  "createdAt": "2022-01-17T16:23:55.717Z",
  "updatedAt": "2022-02-04T05:07:40.272Z",
  "sensors": [
    {
      "_id": "61e59941f9fce835bfd3dcc4",
      "name": "Temperatura 1"
    }
  ]
}
```

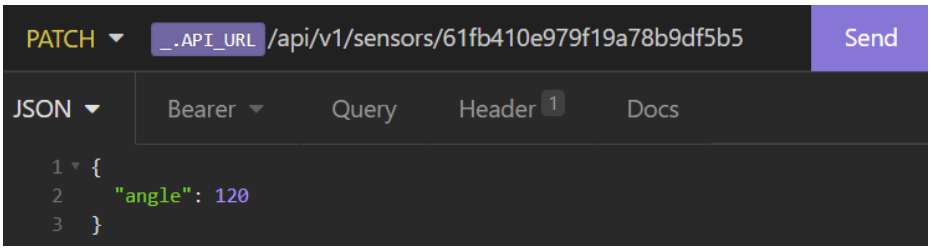
Anexo S: Respuestas Nodo 2

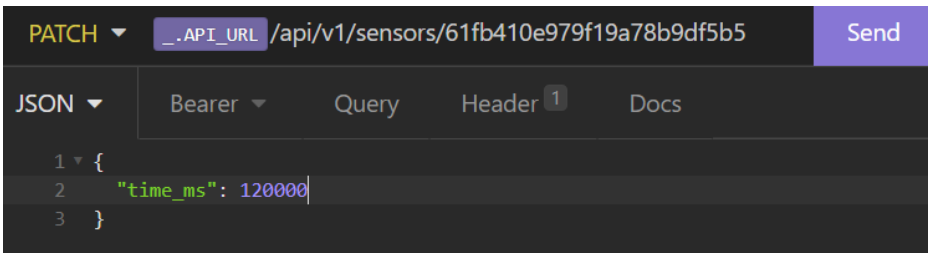


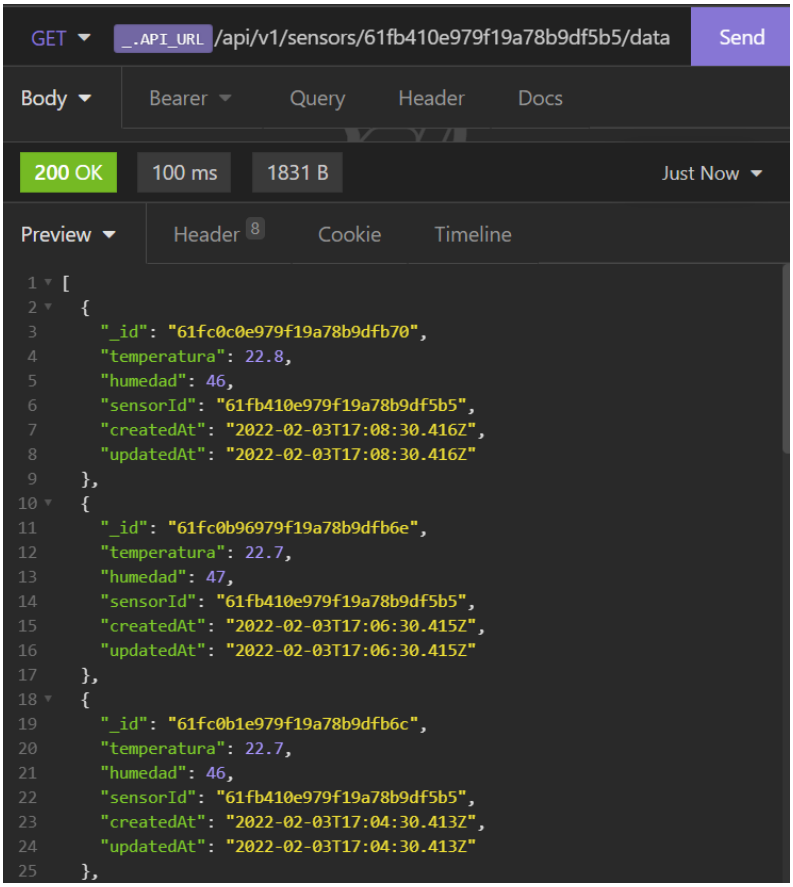
Envió de parámetros:



```
Publish message: {"temperatura":18.9,"humedad":65}
Publish message: {"temperatura":18.9,"humedad":65}
Publish message: {"temperatura":18.9,"humedad":65}
-->Message arrived [/61fb410e979f19a78b9df5b5/config] {"angle":90, "time_ms":5000}
angle: 90
time_ms: 5000
Publish message: {"temperatura":18.9,"humedad":65}
Publish message: {"temperatura":18.9,"humedad":65}
Publish message: {"temperatura":18.9,"humedad":66}
Publish message: {"temperatura":18.9,"humedad":66}
-->Message arrived [/61fb410e979f19a78b9df5b5/config] {"angle":120 }
angle: 120
time_ms: 5000
Publish message: {"temperatura":18.8,"humedad":66}
Publish message: {"temperatura":19,"humedad":66}
Publish message: {"temperatura":19,"humedad":66}
Publish message: {"temperatura":18.9,"humedad":65}
Publish message: {"temperatura":18.8,"humedad":65}
```







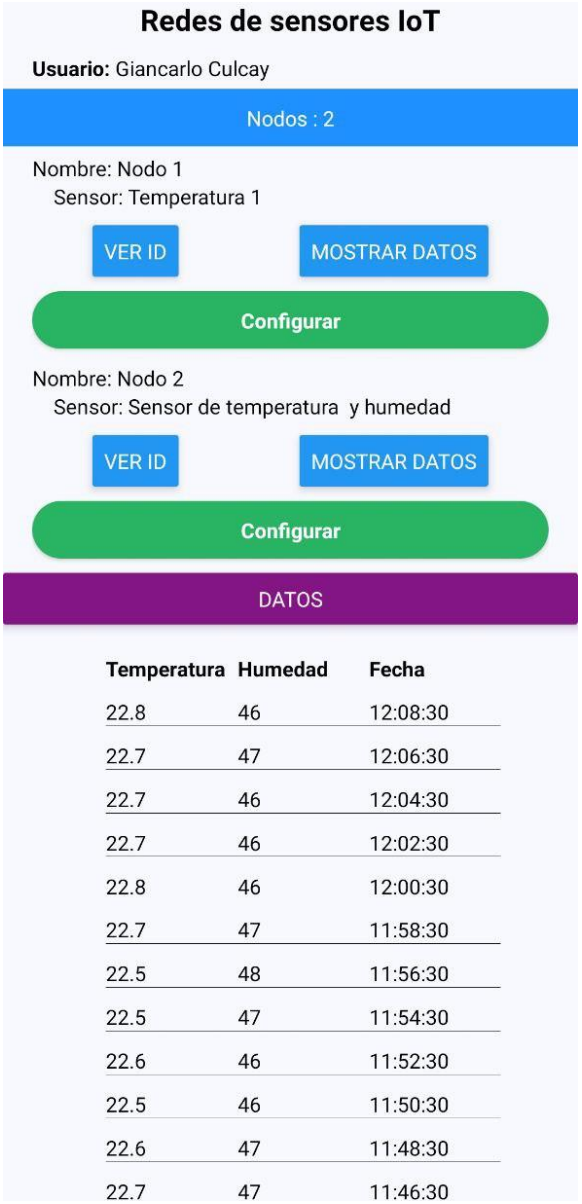
Se puede poner un límite de datos a presentar como: /data?limit=300

Nodo 2

SensorId: 61fb410e979f19a78b9df5b5

Datos:

temperatura	humedad	fecha
22.8	46	3/2/2022 12:08:30
22.7	47	3/2/2022 12:06:30
22.7	46	3/2/2022 12:04:30
22.7	46	3/2/2022 12:02:30
22.8	46	3/2/2022 12:00:30
22.7	47	3/2/2022 11:58:30
22.5	48	3/2/2022 11:56:30
22.5	47	3/2/2022 11:54:30
22.6	46	3/2/2022 11:52:30
22.5	46	3/2/2022 11:50:30
22.6	47	3/2/2022 11:48:30
22.7	47	3/2/2022 11:46:30

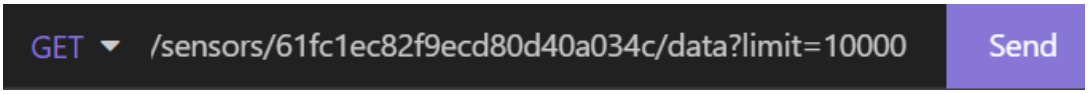

Redes de sensores IoT
Usuario: Giancarlo Culcay
Nodos : 2
Nombre: Nodo 1
Sensor: Temperatura 1
VER ID
MOSTRAR DATOS
Configurar
Nombre: Nodo 2
Sensor: Sensor de temperatura y humedad
VER ID
MOSTRAR DATOS
Configurar
DATOS
Temperatura Humedad Fecha
22.8 46 12:08:30
22.7 47 12:06:30
22.7 46 12:04:30
22.7 46 12:02:30
22.8 46 12:00:30
22.7 47 11:58:30
22.5 48 11:56:30
22.5 47 11:54:30
22.6 46 11:52:30
22.5 46 11:50:30
22.6 47 11:48:30
22.7 47 11:46:30

Anexo T: Respuestas Nodo 3

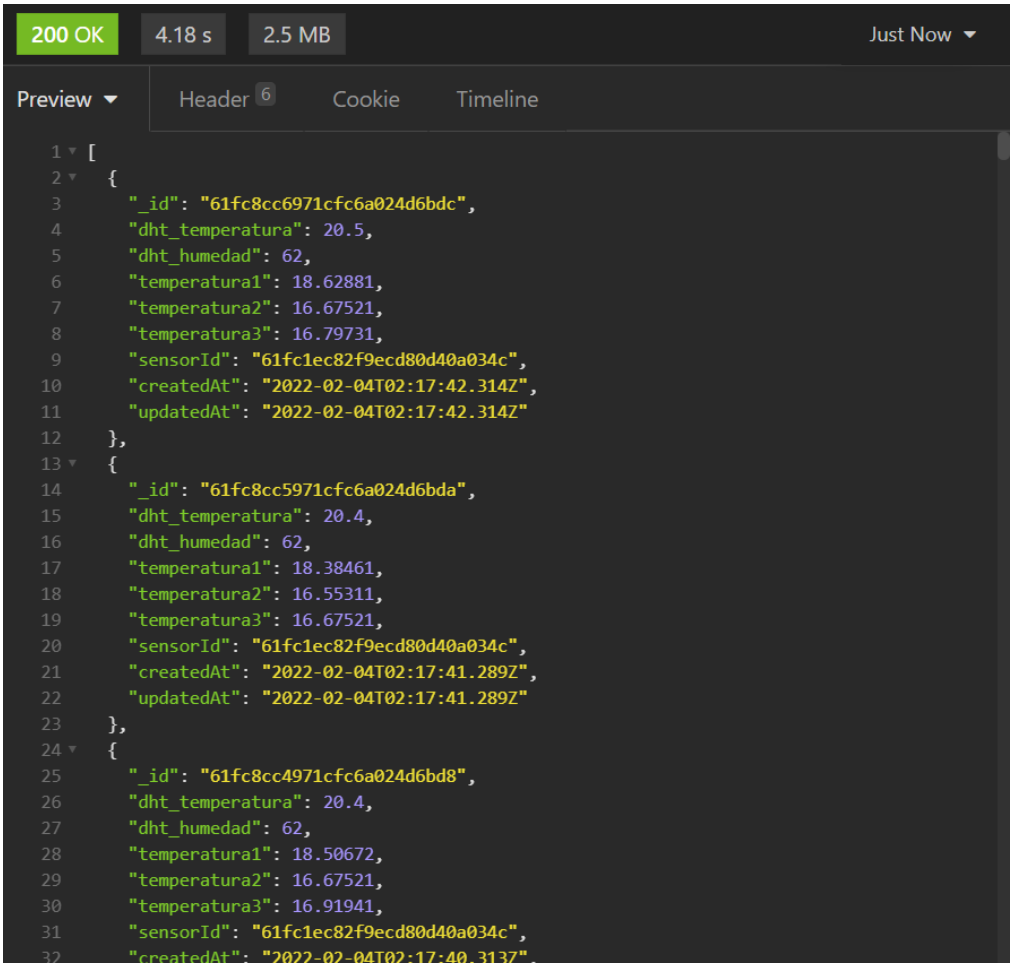
Los datos de este nodo son un array de 10510 elementos

```
$ py cantidad_datos.py
Obteniendo datos...
TIEMPO DE EJECUCION:  5.883671522140503
LONGITUD DE LA LISTA:  10510
```

Se puede indicar el número de datos así:



Respuesta:



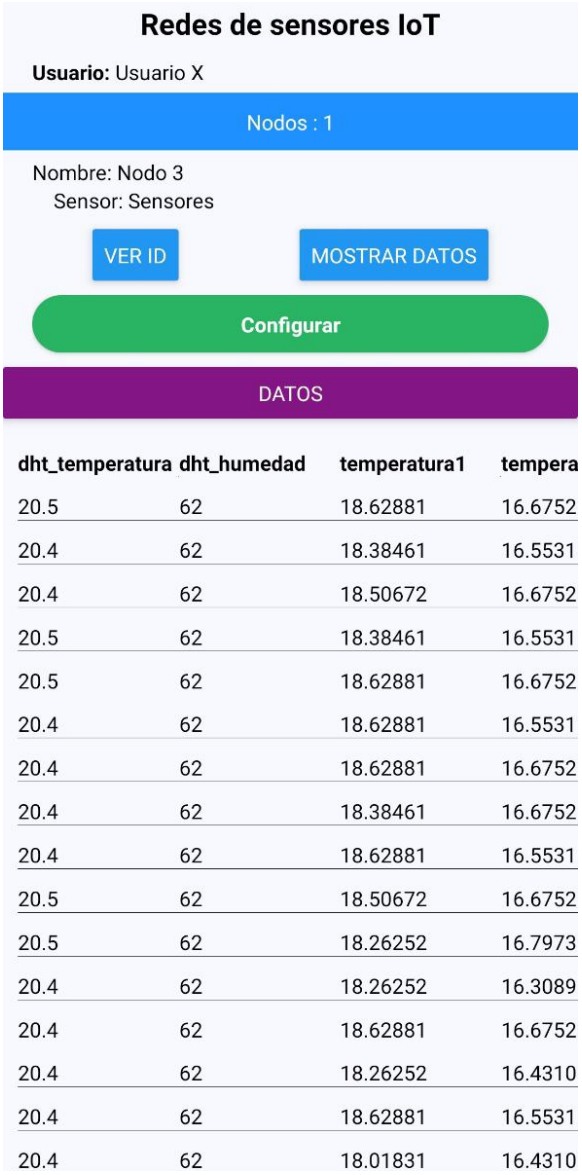

```
[
  {
    "_id": "61fc8cc6971cfc6a024d6bdc",
    "dht_temperatura": 20.5,
    "dht_humedad": 62,
    "temperatura1": 18.62881,
    "temperatura2": 16.67521,
    "temperatura3": 16.79731,
    "sensorId": "61fc1ec82f9ecd80d40a034c",
    "createdAt": "2022-02-04T02:17:42.314Z",
    "updatedAt": "2022-02-04T02:17:42.314Z"
  },
  {
    "_id": "61fc8cc5971cfc6a024d6bda",
    "dht_temperatura": 20.4,
    "dht_humedad": 62,
    "temperatura1": 18.38461,
    "temperatura2": 16.55311,
    "temperatura3": 16.67521,
    "sensorId": "61fc1ec82f9ecd80d40a034c",
    "createdAt": "2022-02-04T02:17:41.289Z",
    "updatedAt": "2022-02-04T02:17:41.289Z"
  },
  {
    "_id": "61fc8cc4971cfc6a024d6bd8",
    "dht_temperatura": 20.4,
    "dht_humedad": 62,
    "temperatura1": 18.50672,
    "temperatura2": 16.67521,
    "temperatura3": 16.91941,
    "sensorId": "61fc1ec82f9ecd80d40a034c",
    "createdAt": "2022-02-04T02:17:40.313Z",
```

Nodo 3

SensorId: 61fc1ec82f9ecd80d40a034c

Datos:

dht_temperatura	dht_humedad	temperatura 1	temperatura 2	temperatura 3	fecha
20.5	62	18.62881	16.67521	16.79731	3/2/2022 21:17:42
20.4	62	18.38461	16.55311	16.67521	3/2/2022 21:17:41
20.4	62	18.50672	16.67521	16.91941	3/2/2022 21:17:40
20.5	62	18.38461	16.55311	16.67521	3/2/2022 21:17:39
20.5	62	18.62881	16.67521	16.79731	3/2/2022 21:17:38
20.4	62	18.62881	16.55311	16.67521	3/2/2022 21:17:38
20.4	62	18.62881	16.67521	16.67521	3/2/2022 21:17:38
20.4	62	18.38461	16.67521	16.55311	3/2/2022 21:17:35
20.4	62	18.62881	16.55311	16.91941	3/2/2022 21:17:34
20.5	62	18.50672	16.67521	16.67521	3/2/2022 21:17:33
20.5	62	18.26252	16.79731	16.67521	3/2/2022 21:17:32
20.4	62	18.26252	16.30891	16.67521	3/2/2022 21:17:31
20.4	62	18.62881	16.67521	16.91941	3/2/2022 21:17:30
20.4	62	18.26252	16.43101	16.67521	3/2/2022 21:17:29
20.4	62	18.62881	16.55311	16.67521	3/2/2022 21:17:28
20.4	62	18.01831	16.43101	16.67521	3/2/2022 21:17:27
20.4	62	18.38461	16.43101	16.67521	3/2/2022 21:17:26
20.4	62	18.01831	16.55311	16.43101	3/2/2022 21:17:25
20.4	62	18.62881	16.67521	16.79731	3/2/2022 21:17:24
20.4	62	18.01831	16.55311	16.67521	3/2/2022 21:17:23
20.4	62	18.26252	16.30891	16.67521	3/2/2022 21:17:22
20.4	62	18.26252	16.43101	16.67521	3/2/2022 21:17:21
20.4	62	18.62881	16.67521	16.67521	3/2/2022 21:17:20
20.4	62	18.26252	16.43101	16.67521	3/2/2022 21:17:19
20.4	62	18.50672	16.55311	16.79731	3/2/2022 21:17:18



dht_temperatura	dht_humedad	temperatura1	tempera
20.5	62	18.62881	16.6752
20.4	62	18.38461	16.5531
20.4	62	18.50672	16.6752
20.5	62	18.38461	16.5531
20.5	62	18.62881	16.6752
20.4	62	18.62881	16.5531
20.4	62	18.62881	16.6752
20.4	62	18.38461	16.6752
20.4	62	18.62881	16.5531
20.5	62	18.50672	16.6752
20.5	62	18.26252	16.7973
20.4	62	18.26252	16.3089
20.4	62	18.62881	16.6752
20.4	62	18.26252	16.4310
20.4	62	18.62881	16.5531
20.4	62	18.01831	16.4310

temperatura1	temperatura2	temperatura3	Fecha
18.62881	16.67521	16.79731	21:17:42
18.38461	16.55311	16.67521	21:17:41
18.50672	16.67521	16.91941	21:17:40
18.38461	16.55311	16.67521	21:17:39
18.62881	16.67521	16.79731	21:17:38
18.62881	16.55311	16.67521	21:17:38
18.62881	16.67521	16.67521	21:17:38
18.38461	16.67521	16.55311	21:17:35
18.62881	16.55311	16.91941	21:17:34
18.50672	16.67521	16.67521	21:17:33
18.26252	16.79731	16.67521	21:17:32
18.26252	16.30891	16.67521	21:17:31
18.62881	16.67521	16.91941	21:17:30
18.26252	16.43101	16.67521	21:17:29
18.62881	16.55311	16.67521	21:17:28
18.01831	16.43101	16.67521	21:17:27

Anexo U: Código fuente ESP8266

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <DHT.h>
#include <DHT_U.h>
#include "Servo.h"

const char* ssid = "****";
const char* password = "****";
const char* mqtt_server = "brokergc.westcentralus.azurecontainer.io";
const char* mqtt_pass = "eyJhbGc****";

WiFiClient espClient;
PubSubClient client(espClient);
DHT dht(D1, DHT11);

unsigned long start_time = 0;
float temperatura, humedad;
int time_ms=1000;

int servo_pin = D7;
Servo myservo;
int angle = 0;

void setup_wifi() {
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  randomSeed(micros());
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
```

```
  Serial.print("-->Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();


  StaticJsonDocument<128> doc;
  DeserializationError error = deserializeJson(doc, payload);
  if (error) return;

  int angle = doc["angle"];
  time_ms = doc["time_ms"] | time_ms;

  Serial.print("angle: ");
  Serial.print(angle);
  Serial.println();
  Serial.print("time_ms: ");
  Serial.print(time_ms);
  Serial.println();

  myservo.write(angle);
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);

    // Attempt to connect
    if (client.connect("gian","nodemcu",mqtt_pass)) {
      Serial.println("connected");
      client.subscribe("/61fb410e979f19a78b9df5b5/config");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 2 seconds");
      // Wait 2 seconds before retrying
      delay(2000);
    }
  }
}

void setup() {
```

```
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  dht.begin();
  myservo.attach(servo_pin);
}

void loop() {
  if(!client.connected()) {
    reconnect();
  }
  client.loop();
  String payload = "";

  if(millis() - start_time >= time_ms){
    start_time = millis();

    temperatura = dht.readTemperature();
    humedad = dht.readHumidity();

    DynamicJsonDocument doc(192);
    doc["temperatura"] = temperatura;
    doc["humedad"] = humedad;

    serializeJson(doc, payload);
    Serial.print("Publish message: ");
    Serial.println(payload);
    client.publish("/61fb410e979f19a78b9df5b5/data", payload.c_str());

  }

}
```

Anexo V: Código fuente ESP32

```
#include <ArduinoJson.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
#include <DHT_U.h>

const char *mqtt_server = "brokergc.westcentralus.azurecontainer.io";
const int mqtt_port = 1883;
const char *mqtt_user = "esp32gc";
const char *mqtt_pass = "eyJhbGciOiJIU***";
const char *topic_subscribe = "/61fc1ec82f9ecd80d40a034c/config";
const char *topic_publish = "/61fc1ec82f9ecd80d40a034c/data";

const char* ssid = "****";
const char* password = "****";

WiFiClient espClient;
PubSubClient client(espClient);

#define DHTPIN 23
DHT dht(DHTPIN, DHT11);
unsigned long start_time = 0;

float dht_temperatura, dht_humedad;
float temperatura1, temperatura2, temperatura3;
int time_ms=1000;
int S1,S2,S3;
void callback(char* topic, byte* payload, unsigned int length);
void reconnect();
void setup_wifi();

void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
  dht.begin();
}

void setup_wifi(){
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
```

```
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  randomSeed(micros());
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {

  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP32Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str(), mqtt_user, mqtt_pass)) {
      Serial.println("connected");

      // Suscripcion
      if(client.subscribe(topic_subscribe)){
        Serial.println("Suscripcion ok");
      }else{
        Serial.println("fallo Suscripciión");
      }
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 2 seconds");
      // Wait 2 seconds before retrying
      delay(2000);
    }
  }
}


void callback(char* topic, byte* payload, unsigned int length){
  String incoming = "";
  Serial.print("Mensaje recibido desde -> ");
  Serial.print(topic);
  Serial.println("");
  for (int i = 0; i < length; i++) {
```

```
    incoming += (char)payload[i];
  }
  incoming.trim();
  Serial.println("Mensaje -> " + incoming);

}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

    String payload = "";

  if(millis() - start_time >= time_ms){
    start_time = millis();

    dht_temperatura = dht.readTemperature();
    dht_humedad = dht.readHumidity();


    S1 = analogRead(39);
    S2 = analogRead(34);
    S3 = analogRead(35);

    temperatura1 = (500.0 * S1)/4095.0;
    temperatura2 = (500.0 * S2)/4095.0;
    temperatura3 = (500.0 * S3)/4095.0;

    DynamicJsonDocument doc(192);
    doc["dht_temperatura"] = dht_temperatura;
    doc["dht_humedad"] = dht_humedad;

    doc["temperatura1"] = temperatura1;
    doc["temperatura2"] = temperatura2;
    doc["temperatura3"] = temperatura3;

    serializeJson(doc, payload);
    Serial.print("Publish message: ");
    Serial.println(payload);
    client.publish(topic_publish, payload.c_str());

  }

}
```