Diseño de un prototipo basado en una plataforma IoT para la detección de alertas tempranas ante el desbordamiento de ríos.

Carlos Iván Ramírez Díaz y Daniel Eduardo Cobos Ayala

Trabajo de Grado para Optar el Título de Ingeniero de Sistemas

Director

Ing. José Geralbert Rubiano

Especialista en Redes de Computadoras

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2025

Dedicado a

Mi madre, Martha Lucía Ayala Jaimes, quien me enseñó el verdadero significado de la responsabilidad. Gracias por levantarme cada mañana en mi infancia, inculcándome el valor del estudio y la importancia de cumplir con mis deberes. Tus enseñanzas y ejemplo me motivaron a siempre dar lo mejor de mí y a esforzarme por sobresalir en cada etapa de mi vida.

Mi padre, Eduardo Cobos Ortega, por su inigualable compañía y por estar siempre presente, brindándome apoyo incondicional. Tus sabias palabras, especialmente "siempre adelante", han sido mi guía en los momentos difíciles y mi impulso para seguir avanzando con determinación.

Mi abuela, Graciela Jaimes de Ayala, quien me ha acompañado desde los inicios de mi vida. Gracias por estar siempre presente para mí, ofreciéndome tu amor, tu apoyo y tu sabiduría en cada momento que lo he necesitado.

Mis hermanos, por su confianza en mí, por su apoyo constante y por recordarme que, con esfuerzo y dedicación, los sueños se pueden alcanzar.

Nuestro director, José Geralbert Rubiano, por su guía, paciencia y compromiso durante el desarrollo de este proyecto. Sus enseñanzas y consejos fueron fundamentales para llevar esta idea a la realidad.

Mi coautor, Carlos Iván Ramírez Díaz, por su colaboración y dedicación en cada etapa de este trabajo. Gracias por ser un excelente compañero en este desafío.

Este trabajo es un reflejo del amor, la compañía y las lecciones que he recibido de todos ustedes.

Daniel Eduardo Cobos Ayala

Dedicado a

Mi madre, Luz Amparo Díaz García, por su apoyo incondicional, su esfuerzo incansable y el cariño que siempre me ha brindado. Gracias por ser mi pilar en cada paso de esta carrera, por tus palabras de aliento en los momentos difíciles y por enseñarme con tu ejemplo el valor de la dedicación y la perseverancia.

A mis hermanos, Félix y Nataly, por su apoyo constante y su motivación. Han sido una fuente de inspiración y fuerza para seguir adelante. Sus palabras de ánimo y confianza en mí han sido fundamentales para alcanzar este logro.

A mi compañero de proyecto, Daniel Eduardo Cobos, por su compromiso, paciencia y determinación. Juntos enfrentamos desafíos y superamos adversidades para sacar adelante tanto esta carrera como este proyecto. Su esfuerzo y dedicación han sido clave para lograrlo, y por ello, le expreso mi más sincero agradecimiento.

A nuestro director, Rubiano, quien pese a todas las dificultades siempre estuvo allí para guiarnos. Gracias por no abandonarnos, incluso cuando tardamos tanto en culminar este proyecto, y por impulsarnos a continuar con ánimo y determinación. Su compromiso ha sido esencial en este proceso.

A todos ustedes, quienes han sido parte esencial de este camino, dedico este proyecto con profundo amor, gratitud y admiración.

Carlos Ivan Ramirez Diaz

Agradecimientos

A los profesores José Geralbert Rubiano y Duvan Yahir Sanabria Echeverry por el invaluable aprendizaje y las enseñanzas que han impartido en nosotros a lo largo de esta carrera. Su compromiso con nuestra formación académica, su paciencia y su dedicación nos han guiado y motivado a superar los retos que enfrentamos. Gracias por compartir no sólo sus conocimientos, sino también su pasión por enseñar y formar profesionales.

A nuestras familias, por su apoyo incondicional, amor y sacrificio. Gracias a nuestras madres, padres, hermanos y demás seres queridos que siempre estuvieron allí para brindarnos aliento, motivación y comprensión en los momentos más desafiantes. Su fe en nosotros nos dio la fuerza para continuar incluso en los días más complicados, y su confianza en nuestro potencial ha sido el motor que nos impulsó a alcanzar este objetivo.

A nuestros amigos, por estar siempre presentes con sus palabras de ánimo, consejos y apoyo sincero. Su compañía fue un refugio en los momentos de estrés, y sus palabras de aliento nos ayudaron a mantenernos enfocados en nuestra meta.

A todos los profesores que nos acompañaron en este camino, no solo por transmitirnos conocimientos, sino también por inspirarnos a ser mejores cada día. Su pasión por enseñar y su disposición para apoyarnos nos dejaron aprendizajes que trascendieron las aulas.

Finalmente, agradecemos profundamente a todos aquellos que, de una forma u otra, aportaron a nuestro desarrollo personal y profesional. Este proyecto y este logro son un reflejo del esfuerzo colectivo y el respaldo que nos brindaron a lo largo de este camino.

Contenido

Lista de Tablas

Lista de Figuras

Glosario

Angular: Framework de desarrollo web de código abierto y gratuito, diseñado para construir aplicaciones de una sola página (SPA) utilizando TypeScript. Proporciona una arquitectura robusta que facilita la creación de interfaces de usuario interactivas y dinámicas, promoviendo la reutilización de componentes y la organización del código.

API: Interfaz de programación de aplicaciones; conjunto de reglas que permite a aplicaciones de software comunicarse y compartir datos, características y funcionalidades.

AWS: Amazon Web Services; la nube más completa y adoptada globalmente, que ofrece más de 200 servicios con funcionalidades completas desde centros de datos a nivel mundial.

Azure: Plataforma de computación en la nube desarrollada por Microsoft, que permite la gestión, acceso y desarrollo de aplicaciones y servicios a través de su infraestructura global.

Back-End: Parte del desarrollo de software que se encarga de la lógica del servidor, la gestión de bases de datos y la comunicación entre el cliente y el servidor. El back-end se ejecuta en el servidor y es responsable de procesar las solicitudes del usuario, gestionar la lógica de la aplicación y almacenar o recuperar datos.

C: Lenguaje de programación de propósito general, diseñado para ser simple y eficiente. C se utiliza ampliamente en el desarrollo de software de sistemas, controladores y aplicaciones que requieren un alto rendimiento y acceso a recursos de hardware.

C + +: Lenguaje de programación que extiende C al incorporar características de programación orientada a objetos. C + + permite a los desarrolladores crear aplicaciones más complejas y estructuradas, combinando la eficiencia del lenguaje C con la flexibilidad de la programación orientada a objetos.

CI/CD: Integración Continua/Despliegue Continuo; un conjunto de prácticas que permite a los desarrolladores integrar cambios en el código con frecuencia y desplegar automáticamente las aplicaciones. Esto mejora la eficiencia y la calidad del software.

Clean Architecture: Filosofía de diseño de software que organiza el código en capas, separando la lógica de negocio de los detalles de implementación y permitiendo una mayor mantenibilidad y flexibilidad.

Cloud Computing: Disponibilidad bajo demanda de recursos informáticos (como almacenamiento e infraestructura) como servicios a través de Internet.

Contenedor docker: Unidad estándar de software que empaqueta el código y sus dependencias, permitiendo que una aplicación se ejecute de manera rápida y confiable en diversos entornos.

HTTP: Protocolo de transferencia de hipertexto; base de la World Wide Web, utilizado para cargar páginas web a través de enlaces de hipertexto, permitiendo la comunicación entre clientes y servidores.

CRUD: Acrónimo de Crear, Leer, Actualizar y Eliminar; conjunto básico de operaciones que se pueden realizar en una base de datos o sistema de gestión de datos.

CSS: Cascading Style Sheets; lenguaje de hojas de estilo utilizado para describir la presentación de documentos HTML. Permite a los desarrolladores aplicar estilos a elementos web, controlando el diseño, el color, las fuentes y la disposición de la página.

Daemon: Proceso que se ejecuta en segundo plano en un sistema operativo, sin interacción directa con el usuario, y que espera recibir solicitudes para realizar tareas específicas.

Docker: Conjunto de productos de plataforma como servicio (PaaS) que utiliza virtualización a nivel de sistema operativo para entregar software en paquetes llamados contenedores.

Docker Compose: Herramienta que permite definir y compartir aplicaciones multicontenedor a través de un archivo YAML, facilitando su gestión con un solo comando.

Dockerfile: Documento de texto que contiene los comandos necesarios para ensamblar una imagen de Docker.

Docker image: Archivo utilizado para ejecutar código en un contenedor de Docker, funcionando como plantilla que contiene las instrucciones necesarias para crear el contenedor.

DRY: Principio "No te repitas"; buena práctica en desarrollo de software que recomienda evitar la duplicación de código.

Edge Computing: Marco de computación distribuida que acerca las aplicaciones empresariales a las fuentes de datos, como dispositivos IoT o servidores locales.

EMQX: Broker MQTT de alto rendimiento y escalabilidad, que facilita la comunicación en tiempo real entre dispositivos y aplicaciones, especialmente en entornos IoT.

Express: Framework web minimalista y flexible para Node.js, que facilita la construcción de aplicaciones y API. Proporciona un conjunto robusto de características para el desarrollo web y móvil, permitiendo a los desarrolladores gestionar rutas, solicitudes y middleware de manera eficiente.

Fog Computing: Tecnología en la nube donde los datos generados por dispositivos finales se procesan en minicentros de datos descentralizados, en lugar de cargarse directamente en la nube.

Framework: Colección de componentes de software reutilizables que permiten desarrollar nuevas aplicaciones de forma más eficiente.

Front-End: Parte del desarrollo de software que se ocupa de la interfaz de usuario y la experiencia del usuario en una aplicación o sitio web. El front-end incluye todo lo que los usuarios ven y con lo que interactúan directamente, como el diseño visual, los elementos de la interfaz y la interactividad del lado del cliente.

GIT: Sistema de control de versiones distribuido que permite a los desarrolladores rastrear cambios en el código fuente a lo largo del tiempo, facilitando la colaboración en proyectos de software y la gestión de versiones.

Google Cloud: Conjunto de servicios de computación en la nube ofrecidos por Google, que incluye computación, almacenamiento, análisis de datos y aprendizaje automático, junto con herramientas de gestión.

GPIO: Puerto de entrada/salida de propósito general; gestiona señales digitales entrantes y salientes. Como puerto de entrada, se puede utilizar para comunicar a la CPU las señales de encendido/apagado recibidas de interruptores o las lecturas digitales provenientes de sensores.

Heroku: Plataforma en la nube que permite a las empresas crear, entregar, monitorear y escalar aplicaciones de manera eficiente.

HTTPS: Protocolo seguro de transferencia de hipertexto; versión segura de HTTP que cifra la comunicación entre un navegador web y un sitio web, aumentando la seguridad en la transferencia de datos, especialmente al manejar información confidencial como credenciales bancarias o datos personales.

IoT: Internet de las cosas; Red de dispositivos físicos, vehículos, electrodomésticos y otros objetos físicos que cuentan con sensores, software y conectividad de red integrados, lo que les permite recopilar y compartir datos.

Java: Lenguaje de programación de alto nivel, orientado a objetos y multiplataforma, que permite escribir código una vez y ejecutarlo en cualquier lugar (WORA).

JSON: JavaScript Object Notation; formato de intercambio de datos abierto y legible por humanos, que utiliza pares atributo-valor y estructuras de matriz para almacenar y transmitir objetos de datos.

JWT: JSON Web Token; estándar abierto que define una forma compacta y autónoma de transmitir información de forma segura entre partes, verificable gracias a su firma digital.

Lenguaje de programación: Sistema de notación utilizado para escribir programas informáticos, caracterizado por su sintaxis (forma) y semántica (significado).

JavaScript: Lenguaje de programación que permite implementar funcionalidades complejas en páginas web.

Microservicios: Enfoque arquitectónico en el desarrollo de software que divide el sistema en pequeños servicios independientes que se comunican a través de API bien definidas.

MongoDB: Base de datos multiplataforma orientada a documentos, clasificada como NoSQL, que utiliza documentos JSON con esquemas opcionales.

MQTT: Protocolo de mensajería diseñado para facilitar la comunicación entre máquinas en entornos con recursos y ancho de banda limitados, como dispositivos IoT.

Next.js: Framework creado por la empresa Vercel que proporciona aplicaciones web basadas en React con renderizado del lado del servidor y generación de sitios web estáticos.

Node.js: Entorno de ejecución para JavaScript en el lado del servidor, basado en el motor V8 de Google Chrome. Permite a los desarrolladores crear aplicaciones de red escalables y eficientes, utilizando un modelo orientado a eventos y no bloqueante.

Nest.js: Framework para construir aplicaciones del lado del servidor utilizando

Node.js. Está diseñado para ser escalable y eficiente, empleando principios de programación orientada a objetos y modular, lo que facilita la creación de aplicaciones complejas y mantenibles.

Spring Boot: Framework basado en Java que simplifica el desarrollo de aplicaciones empresariales y microservicios. Proporciona un conjunto de convenciones y configuraciones automáticas para acelerar el proceso de desarrollo, permitiendo a los desarrolladores centrarse en la lógica de negocio.

NoSQL: Tipo de bases de datos diseñadas para modelos de datos específicos, que almacenan datos en esquemas flexibles y escalables, ideales para aplicaciones modernas.

PostgreSQL: Sistema de base de datos relacional de código abierto que extiende el lenguaje SQL y maneja cargas de trabajo de datos complejas de manera segura.

Railway: Plataforma en la nube que simplifica la implementación de aplicaciones y bases de datos, permitiendo a los desarrolladores gestionar la infraestructura sin configuraciones complejas.

React: Biblioteca de JavaScript frontend gratuita y de código abierto para crear interfaces de usuario basadas en componentes de Facebook Inc.

Responsive Design: Diseño web responsivo; enfoque que garantiza que las páginas web se visualicen adecuadamente en todos los tamaños y resoluciones de pantalla, optimizando la usabilidad.

REST: Transferencia de estado representacional; estilo arquitectónico que establece normas para la comunicación entre sistemas en la web, caracterizado por ser sin estado y separar las responsabilidades entre cliente y servidor.

REST API: Interfaz de programación de aplicaciones que sigue los principios del estilo arquitectónico REST.

RXJS: Biblioteca para programación reactiva en JavaScript que facilita la manipulación de flujos de datos asíncronos mediante observables. Proporciona herramientas para gestionar eventos, datos y la comunicación entre componentes de manera eficiente y sencilla.

SPA: Aplicación web que carga una única página HTML y actualiza el contenido dinámicamente a medida que el usuario interactúa con la aplicación, sin necesidad de recargar la página. Este enfoque mejora la experiencia del usuario al proporcionar una navegación más fluida y rápida.

SQL: Lenguaje de consulta estructurado que permite almacenar y procesar información en bases de datos relacionales, utilizando un formato tabular de filas y columnas.

TypeScript: Superset de JavaScript desarrollado por Microsoft que añade tipado estático opcional y otras características avanzadas. Diseñado para facilitar el desarrollo de aplicaciones a gran escala.

UI (Interfaz de Usuario): Punto de interacción entre el usuario y un dispositivo o aplicación, que incluye elementos visuales y de audio como pantallas, botones, iconos y tipografías.

UX (Experiencia de Usuario): Proceso de diseño enfocado en crear productos que ofrezcan experiencias significativas y relevantes a los usuarios, abarcando todo el ciclo de uso del producto.

Vercel: Empresa estadounidense de plataformas en la nube que mantiene el marco de desarrollo web Next.js, facilitando la implementación de aplicaciones.

Vue.js: Framework progresivo de código abierto para el desarrollo de interfaces de usuario y aplicaciones de una sola página (SPA). Se centra en la capa de vista del modelo de

diseño y es conocido por su facilidad de integración y su enfoque en la reactividad y la composición de componentes.

WebHook: Comunicación liviana basada en eventos que envía automáticamente datos entre aplicaciones a través de HTTP, activando flujos de trabajo mediante eventos específicos.

WebSocket: Tecnología que permite establecer una comunicación interactiva y bidireccional entre el navegador del usuario y un servidor, sin necesidad de sondeos constantes.

Resumen

Título: Diseño de un prototipo basado en una plataforma IoT para la detección de alertas tempranas ante el desbordamiento de ríos.*

Autores: Carlos Iván Ramírez Díaz, Daniel Eduardo Cobos Ayala**

Palabras clave: IoT, Sistema de alerta temprana, Desbordamientos de ríos, Desarrollo web, Computación en la nube, Aplicación, Control, Prevención de desastres naturales.

Descripción: Colombia, como uno de los países con mayor diversidad de recursos hídricos en el mundo. Sus diversos ríos y arroyos están ligados a la posibilidad de desastres naturales como lo son los desbordamientos fluviales. Estos siniestros ambientales representan una de las principales amenazas para las comunidades ubicadas en áreas cercanas a cuerpos de agua; estos pueden provocar daños irreparables en infraestructura, pérdidas humanas y daños ambientales.

Ante este panorama, surge la necesidad de implementar sistemas de monitoreo avanzados que permitan la detección temprana de potenciales desbordamientos de ríos en Colombia. El proyecto consta del diseño de una plataforma IoT conformada por un conjunto de sensores que permiten la obtención en tiempo real de mediciones como lo son el nivel del río, velocidad del caudal y detección de lluvia, las cuales serán almacenada en una plataforma web con el fin de hacer un seguimiento y generar alertas vía whatsapp a los diferentes usuarios en caso de una variación considerable en la zona del cauce.

---

*Proyecto de grado.

**Facultad de Fisicomécanica. Escuela de Ingeniería de Sistemas e Informática. Ingeniería de Sistemas. Director: Jose Geralbert Rubiano. Magister en TIC para la Educación

Abstract

Title: Design of a prototype based on an IoT platform for the detection of early warnings of river overflows.*

Authors: Carlos Iván Ramírez Díaz, Daniel Eduardo Cobos Ayala**

Keywords: IoT, Early warning system, River overflows, Web development, Cloud computing, Application, Control, Natural disaster prevention.

Description: Colombia, as one of the countries with the greatest diversity of water resources in the world. Its diverse rivers and streams are linked to the possibility of natural disasters such as river overflows. These environmental disasters represent one of the main threats to communities located in areas near bodies of water; they can cause irreparable damage to infrastructure, human losses and environmental damage.

Given this scenario, there is a need to implement advanced monitoring systems that allow early detection of potential river overflows in Colombia. The project consists of the design of an IoT platform formed by a set of sensors that allow obtaining real-time measurements such as river level, flow velocity and rainfall detection, which will be stored in a web platform in order to track and generate alerts via whatsapp to different users in case of a considerable variation in the area of the riverbed.

---

*Degree Work.
**Faculty of Physico Mechanical Engineering. School of Systems and Computer Engineering. Systems Engineering. Advisor: Jose Geralbert Rubiano. Master's in ICT for Education.

Introducción

Las inundaciones no solo representan uno de los desastres naturales más devastadores en diversas regiones del mundo, sino que también son de los más recurrentes (World Meteorological Organization, 2023). En particular, en áreas urbanas, las inundaciones derivadas de tormentas extremas en lapsos cortos han ocasionado considerables pérdidas materiales, económicas, ambientales y humanas a nivel global.

Según Ávila et al. (2019), los eventos de lluvia intensa en Colombia, junto con alteraciones ambientales como la deforestación y/o la urbanización, pueden desencadenar la incidencia de peligros hidrológicos, como inundaciones, crecidas repentinas y deslizamientos de tierra, especialmente en áreas tropicales. En Colombia, estos peligros hidrológicos ocurren principalmente como resultado de precipitaciones intensas locales durante la fase fría de la Oscilación del Sur de El Niño (ENSO), conocida como La Niña.

Ante este panorama, surge la necesidad imperante de implementar sistemas de monitoreo avanzados que permitan la detección temprana de potenciales desbordamientos de ríos en Colombia. Es en este contexto que se enmarca el proyecto de investigación "Diseño de un prototipo basado en una plataforma IoT para la detección de alertas tempranas ante el desbordamiento de ríos", el cual se propone como uno de los primeros pasos hacia la mitigación de este riesgo en el país.

El objetivo principal de este proyecto es desarrollar un prototipo que permita la detección temprana de desbordamientos de ríos y la transmisión oportuna de alertas precisas a la comunidad afectada. Para ello, se realizará un exhaustivo análisis conceptual y referencial, que abordará conceptos clave relacionados con el problema y se basará en implementaciones

similares llevadas a cabo en otras partes del mundo. Estas experiencias previas servirán como guía y proporcionarán información útil para diseñar e implementar soluciones efectivas en el contexto colombiano.

1 Objetivos

1.1 Objetivo General

Diseñar un prototipo basado en una plataforma IoT con interfaz web para la detección de alertas tempranas ante el desbordamiento de ríos, integrando hardware, software y tecnologías de comunicación que permitan monitorear variables críticas en tiempo real.

1.2 Objetivos Específicos

- Caracterizar el desbordamiento de ríos para la determinación de las variables, junto con la definición de los requerimientos funcionales y no funcionales para el prototipo y diseño de la plataforma IoT.
- Seleccionar los elementos a nivel de hardware y software para la creación del prototipo y la plataforma IoT.
- Diseñar la estructura del hardware y software asegurando la integración efectiva de los elementos seleccionados a fin de garantizar una interacción eficaz.
- Verificar el funcionamiento de la plataforma IoT desarrollada mediante el prototipo funcional en un entorno controlado.

2 Marco referencial

2.1 Marco conceptual

2.1.1 Internet de las cosas (IoT)

El concepto de Internet de las cosas (IoT), según la Unión Internacional de Telecomunicaciones (ITU) en su informe de 2005 sobre el Internet de las Cosas, se define como: "La idea básica del IoT es que prácticamente cualquier cosa física en este mundo también puede convertirse en una computadora conectada a Internet" (ITU, 2005). Sin embargo Fleisch (2010) hace una aclaración: “Para ser más precisos, las cosas no se convierten en computadoras, pero pueden contener computadoras diminutas. Cuando lo hacen, a menudo se les llama ‘cosas inteligentes’, porque pueden actuar de manera más inteligente que las cosas que no han sido etiquetadas”.

Un sistema IoT puede recopilar datos del entorno a través de sensores, transmitir estos datos a través de redes de comunicación y utilizar plataformas de análisis para procesar la información. Esta interconexión facilita la automatización de tareas, la toma de decisiones inteligentes y la creación de experiencias personalizadas para los usuarios, transformando la manera en que la humanidad interactúa con el mundo físico.

2.1.2 Protocolo MQTT (Message Queuing Telemetry Transport)

El protocolo MQTT es un estándar de mensajería ligero diseñado para la comunicación en dispositivos IoT con recursos limitados. Utiliza un modelo de publicación-suscripción, en el cual los clientes (como los ESP32) se comunican a través de un broker central que distribuye mensajes mediante tópicos.

Figura 1.

Esquema de un clúster de EMQX con múltiples publicadores y suscriptores.

Nota. Tomado de Shi, Z. (s.f.). Exploring the Basics of EMOX MQTT Broker Clustering: An Introduction. EMOX. https://www.emox.com/en/blog/mqtt-broker-clustering

2.1.3 Sensor

Un sensor es un dispositivo que detecta y responde a algún tipo de estímulo o cambio en su entorno físico o químico, convirtiendo esta información en una señal eléctrica o digital que puede ser interpretada por otros dispositivos, sistemas o seres humanos. Los sensores pueden detectar una amplia variedad de variables, como temperatura, luz, sonido, presión, movimiento, humedad, entre otras.

Los sensores de IoT tienen una amplia variedad de aplicaciones en diferentes entornos, incluyendo hogares, campos agrícolas, vehículos, aviones y entornos industriales. El uso de sensores permite cerrar la brecha entre el mundo físico y el mundo lógico.

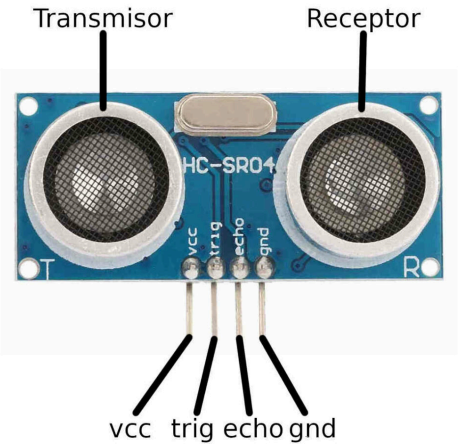
Este tipo de dispositivos desempeñan un papel fundamental en el Internet de las cosas (IoT) ya que permiten crear un ecosistema para recopilar y procesar datos sobre un entorno específico para que pueda monitorearse, gestionarse y controlarse de manera más fácil y eficiente.

2.1.3.1 HC-SR04 (Ultrasónico)

Es un sensor diseñado para medir distancia por medio de ondas de sonido. Funciona emitiendo un pulso ultrasónico desde su transmisor, que viaja a través del aire hasta que se encuentra un objeto que refleja la onda de vuelta al receptor. El sensor mide el tiempo que tarda en viajar la onda en regresar y utiliza dicho valor para calcular la distancia entre el sensor y el objeto. Con un rango de medición efectivo de 2 a 400 centímetros, con una precisión aproximada de 3 milímetros (Cytron Technologies, 2013).

Figura 2.

Esquema del sensor ultrasónico HC-SR04 con sus conexiones: VCC, TRIG, ECHO y GND.



*Nota***.** Tomado de Para Arduino. (2024, 10 de abril). Sensor ultrasónico HC-SR04 - Para Arduino. https://paraarduino.com/sensores/sensor-ultrasonico-hc-sr04/

2.1.3.2 Sensor de Lluvia YL-83

Este sensor consta de una placa sensible recubierta de níquel que opera bajo el principio de resistencia variable en función de la humedad. En condiciones secas cuando no hay gotas de lluvia en la placa, la resistencia es alta. A medida que las gotas caen sobre la placa, la resistencia disminuye proporcionalmente a la cantidad de agua presente, es decir la

resistencia eléctrica es inversamente proporcional a la cantidad de agua detectada, lo que permite un medir con cierta precisión los niveles de lluvia en función de la variación de la señal analogica generada.

Figura 3.

Módulo de Sensor de lluvia YL-83.

Nota. Tomado de Cortés, A. (2021, 16 de mayo). Proyecto 26 – Sensor de lluvia. Acortes Software. Recuperado de https://acortes.co/proyecto-26-sensor-de-lluvia/

2.1.3.3 Sensor de Flujo de Agua

Es un dispositivo diseñado para medir la cantidad de agua que fluye a través de un canal. Su funcionamiento se basa en los pulsos generados por un rotor interno que gira cuando es impulsado por el agua en movimiento. La velocidad de rotación del rotor es directamente proporcional al caudal del agua, y el número de pulsos generados en un determinado período se utiliza para calcular el flujo en litros por minuto (L/min).

Figura 4.

Sensor de flujo o caudalímetro.

*Nota***.** Tomado de Admin. (2022, 26 de diciembre). *Sensores de Flujo*. Proyectos con Arduino. https://proyectosconarduino.com/sensores/flujo-caudalimetro/

2.1.4 Sistemas de alerta temprana

El concepto de los sistemas de alerta temprana es crucial y se entrelaza estrechamente en la gestión del riesgo de desbordamientos de ríos. Según la Oficina de las Naciones Unidas para la Reducción del Riesgo de Desastres o UNISDR (2012) estos sistemas se definen como: "El conjunto de capacidades necesarias para generar y difundir información de alerta oportuna y significativa que permita a las personas, comunidades y organizaciones amenazadas por un peligro prepararse y actuar de manera apropiada y con tiempo suficiente para reducir la posibilidad de daño o pérdida es crucial".

La implementación de este tipo de sistemas implica el uso de tecnologías avanzadas, como sensores remotos y algoritmos de análisis de datos, para detectar patrones y anomalías que puedan indicar la ocurrencia de siniestro.

2.1.5 Infraestructura tecnológica

La infraestructura tecnológica se refiere al conjunto de componentes físicos, recursos y servicios necesarios para el funcionamiento y soporte de sistemas de tecnología de la información (TI) y comunicaciones (TIC).

Gupta y Sharma (2002) señalan que una infraestructura tecnológica, abarca componentes de hardware y software que respaldan las aplicaciones y los requisitos de gestión de la información de una empresa. Entre los componentes de la infraestructura tecnológica se encuentran el hardware de computadora, el software de sistemas, los sistemas de comunicación y redes, las herramientas de desarrollo y el software de aplicaciones. La arquitectura tecnológica mapea la infraestructura física con la información, los procesos y las estructuras organizativas. Las arquitecturas indican cómo se mapean las diversas aplicaciones, almacenes de información y vínculos en el modelo físico.

2.1.6 Cloud computing

El cloud computing o computación en la nube, según la definición proporcionada por Hamdaqa & Tahvildari, (2012), se define como el modelo para habilitar el acceso a la red bajo demanda a un conjunto compartido de recursos informáticos configurables (servidores, redes, servicios, almacenamiento y aplicaciones) que pueden ser liberados fácilmente con la mínima interacción del proveedor de servicios de Internet y con esfuerzos de gestión mínimos. Este enfoque revolucionario en la tecnología de la información permite a las organizaciones acceder a recursos informáticos de manera flexible y escalable, eliminando la necesidad de infraestructura física costosa y compleja, y permitiendo una mayor eficiencia operativa y agilidad empresarial.

2.1.7 Edge Computing

Según Scale Computing (2023), Edge computing tiene como objetivo abordar las limitaciones de la computación en la nube distribuyendo la computación, el almacenamiento y las redes más cerca de los dispositivos y sensores de borde. Este enfoque descentralizado reduce la latencia, mejora la privacidad de los datos y mejora la eficiencia del procesamiento de datos. Este tipo de computación permite una toma de decisiones más rápida, lo que lo hace ideal para aplicaciones como IoT, Industria 4.0, IA de visión por computadora y análisis en tiempo real.

2.1.8 Fog Computing

Scale Computing (2023) establece que el fog computing es un modelo de computación distribuida diseñado para complementar el edge computing. Proporciona una capa intermedia de infraestructura informática, conocida como ‘capa de niebla’, que extiende las capacidades de los dispositivos de borde al proporcionar recursos y servicios adicionales. Esto permite un procesamiento más eficiente de datos y una respuesta más rápida a las aplicaciones y servicios, al tiempo que reduce la carga en la infraestructura de la nube.

2.1.9 Arquitectura Basada en Microservicios

La arquitectura basada en microservicios es un estilo de desarrollo de software que organiza una aplicación como un conjunto de servicios pequeños e independientes. Cada uno de estos microservicios ejecuta un proceso propio y se comunica con otros a través de mecanismos ligeros como API REST o mensajes (Mamani Rodríguez et al., 2020). Este enfoque permite que los microservicios puedan ser desarrollados, desplegados y mantenidos de manera independiente, lo que mejora la flexibilidad, escalabilidad y tolerancia a fallos del sistema (Loza Peralta, 2020). En contraste con las arquitecturas monolíticas tradicionales,

donde todas las funcionalidades están empaquetadas en una sola unidad, los microservicios permiten a los equipos de desarrollo trabajar de forma más ágil y eficiente, ya que pueden actualizar o escalar solo una parte específica del sistema sin afectar al resto (López & Maya, 2017). Este enfoque ha sido crucial para la incorporación de nuevas tecnologías como el Internet de las Cosas, donde es esencial que los dispositivos puedan comunicarse y adaptarse fácilmente.

2.1.10 Docker

Docker es una plataforma de software que permite crear, probar e implementar aplicaciones de manera rápida y eficiente. Su principal característica es la capacidad de empaquetar aplicaciones en contenedores, los cuales incluyen todas las dependencias necesarias como bibliotecas, herramientas de sistema, código y versiones ejecutables (Amazon Web Services [AWS], n.d.). Esto garantiza que las aplicaciones puedan ejecutarse de manera consistente en cualquier entorno, eliminando problemas de configuración o compatibilidad.

Entre sus principales beneficios se destacan la escalabilidad, que permite construir arquitecturas de microservicios distribuidas y adaptables a las necesidades del sistema; la integración y entrega continua (CI/CD), que simplifica la implementación al eliminar conflictos entre entornos de desarrollo y producción; y el ahorro de costos, al optimizar el uso de recursos en un solo servidor. Además, Docker facilita la transferencia de aplicaciones desde equipos de desarrollo locales a entornos de producción con facilidad, gracias a la estandarización que ofrece (Docker Docs, n.d.).

Finalmente, Docker se integra de manera eficiente con servicios en la nube como Amazon ECS y AWS Fargate, permitiendo la gestión de contenedores a gran escala y ofreciendo herramientas como Amazon Elastic Container Registry (ECR) para el

almacenamiento seguro de imágenes de Docker. Esto lo convierte en una herramienta fundamental para proyectos que buscan aprovechar arquitecturas modernas y procesos automatizados de desarrollo y despliegue.

2.1.11 Base de datos

Una base de datos es un sistema diseñado para recopilar, organizar y almacenar información de manera eficiente, permitiendo su acceso, modificación y análisis. Se clasifican en bases de datos relacionales, que estructuran los datos en tablas con relaciones definidas mediante claves primarias y foráneas, y bases de datos no relacionales o NoSQL, que ofrecen modelos flexibles para manejar grandes volúmenes de datos no estructurados o semiestructurados (AWS, n.d.).

Las bases de datos relacionales, como MySQL o PostgreSQL, son ideales para aplicaciones donde la consistencia y la integridad de los datos son críticas. Por otro lado, las bases de datos NoSQL, como MongoDB o DynamoDB, están diseñadas para escalar horizontalmente y gestionar datos heterogéneos, siendo una opción óptima para proyectos IoT o big data debido a su alta velocidad y capacidad de adaptación (AWS, n.d.).

En el contexto del proyecto, el enfoque NoSQL proporciona ventajas clave como la escalabilidad y la eficiencia en la gestión de datos distribuidos, lo que asegura un rendimiento óptimo en sistemas que requieren flexibilidad y tolerancia a fallos. Este modelo resulta esencial para manejar los datos generados por sensores y dispositivos IoT en tiempo real, garantizando la disponibilidad y consistencia eventual de la información.

2.1.12 CI/CD pipeline

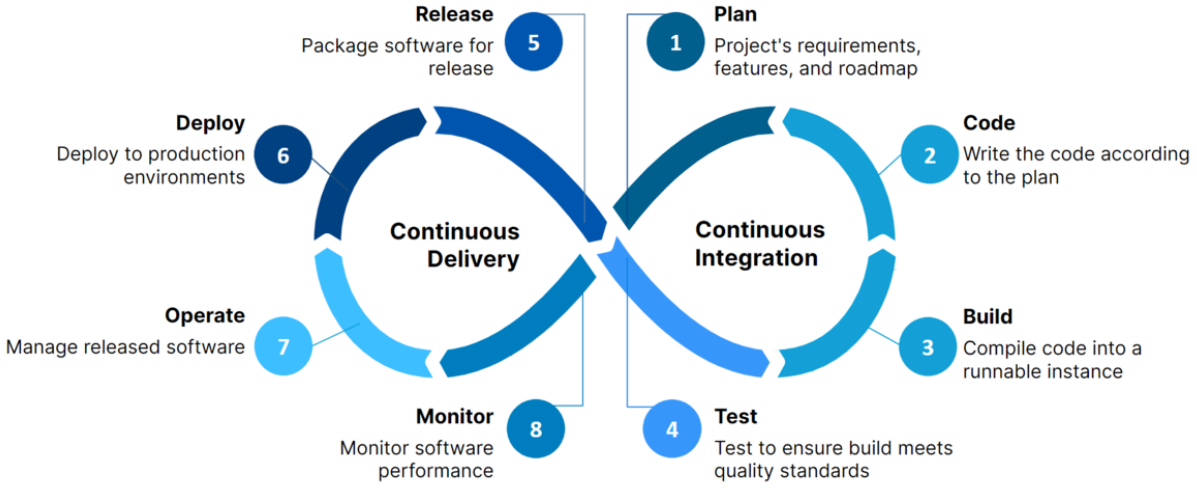
El pipeline de CI/CD (integración y entrega/despliegue continuos) es una metodología clave en el desarrollo moderno de software, diseñada para optimizar la entrega rápida y confiable de aplicaciones. La integración continua (CI) consiste en la automatización de la incorporación de cambios en el código a un repositorio central, donde se ejecutan pruebas automáticas para detectar errores de forma temprana. Esto permite a los equipos mantener una base de código consistente y funcional (Civo, 2024).

Por otro lado, la entrega continua (CD) automatiza el proceso de liberación de software, garantizando que los cambios que pasan las pruebas sean implementados en ambientes de producción de manera segura y eficiente. Cuando este proceso incluye despliegues automáticos en producción, se denomina despliegue continuo, lo que asegura que las nuevas funcionalidades y correcciones de errores lleguen a los usuarios finales con rapidez.

En entornos de computación en la nube, el pipeline de CI/CD aprovecha la escalabilidad y flexibilidad de las plataformas para ejecutar pruebas y despliegues en paralelo, reduciendo tiempos y costos. Además, integra herramientas avanzadas para la monitorización y retroalimentación en tiempo real, mejorando la calidad y seguridad del software entregado. Estas capacidades hacen del pipeline de CI/CD una pieza fundamental en proyectos que requieren agilidad y adaptabilidad continuas.

Figura 5.

Diagrama del ciclo CI/CD (Integración y Entrega Continua).



*Nota***.** Tomado de Arna, U. F. (2024, 18 de junio). The role of the CI/CD pipeline in cloud computing.Ciro.com.

https://www.ciro.com/blog/the-role-of-the-ci-cd-pipeline-in-cloud-computing

2.1.13 API REST

Las API REST (Transferencia de Estado Representacional) son interfaces de programación diseñadas para facilitar la comunicación entre sistemas mediante principios arquitectónicos que promueven la flexibilidad, escalabilidad e independencia tecnológica. Estas APIs permiten acceder y manipular recursos a través de métodos HTTP estándar como GET, POST, PUT y DELETE, utilizando formatos de datos como JSON o XML para la representación de los recursos (IBM, n.d.).

Entre sus características principales se destacan la independencia entre cliente y servidor, la ausencia de estado, que simplifica el manejo de sesiones, y la capacidad de almacenamiento en caché para mejorar el rendimiento. Estas propiedades hacen que las API

REST sean ideales para conectar microservicios y componentes en arquitecturas distribuidas. Además, la flexibilidad del diseño REST permite su implementación en diversos lenguajes de programación, adaptándose fácilmente a distintos entornos tecnológicos (AWS, n.d.).

En el contexto del proyecto, las API REST se utilizan para exponer y gestionar los datos generados por los sensores IoT, permitiendo a los usuarios interactuar de manera eficiente con el sistema desde aplicaciones web o móviles. Este enfoque asegura una comunicación fluida y segura entre los componentes, facilitando la integración con otros servicios o plataformas y garantizando la escalabilidad necesaria para futuras ampliaciones del sistema.

2.1.14 Aplicación web

Una aplicación web es un software que se ejecuta en un navegador y permite a los usuarios acceder a funcionalidades complejas sin necesidad de instalar software adicional. Estas aplicaciones, desarrolladas con tecnologías como HTML, CSS y JavaScript, están diseñadas para almacenar datos en la nube y se accede a ellas mediante una conexión a internet. Su arquitectura cliente-servidor se divide en scripts del lado del cliente, que manejan la interfaz de usuario, y scripts del lado del servidor, que gestionan el procesamiento de datos y las respuestas a las solicitudes del cliente (AWS, n.d.; ESIC Formación Profesional Superior, 2024).

2.1.15 Single Page Application (SPA)

Una Single-Page Application (SPA) es un tipo de aplicación web que opera sobre un único documento cargado inicialmente en el navegador. En lugar de recargar páginas completas desde el servidor, utiliza tecnologías como JavaScript, Ajax o WebSocket para actualizar dinámicamente el contenido en respuesta a la interacción del usuario. Esto permite

una navegación fluida, tiempos de respuesta más rápidos y una experiencia más interactiva y optimizada, sin necesidad de recargar toda la página con cada acción (López Mora, 2024; MDN Web Docs, n.d.).

Gracias a su arquitectura, las SPA se sustentan en frameworks y librerías como React, Angular o Vue.js, combinados con HTML y CSS para el frontend, mientras que el backend, generalmente, entrega datos en formato JSON mediante APIs REST. Estas aplicaciones son ideales para proyectos que priorizan velocidad y usabilidad, aunque presentan desafíos en aspectos como SEO, seguridad y mantenimiento del estado del cliente, lo que hace crucial evaluar si son la solución adecuada según los requisitos del proyecto (López Mora, 2024; MDN Web Docs, n.d.).

2.2 Estado del arte

Aunque la implementación de sistemas de alerta temprana para desastres naturales es un campo de investigación en constante evolución, la disponibilidad de estudios específicos sobre este tema en Colombia es limitada. Sin embargo, diversos autores han logrado implementar con éxito sistemas de este tipo en contextos internacionales. Estos logros han contribuido significativamente a la comprensión y aplicación de tecnologías para la detección y mitigación de riesgos naturales. A pesar de estos avances, todavía existe una brecha en la implementación efectiva de sistemas de alerta temprana en Colombia, lo que subraya la necesidad de una investigación más amplia y detallada en este campo para abordar los desafíos específicos del país.

2.2.1 Gestión de desastres naturales: un análisis de los avances, oportunidades y desafíos tecnológicos (krichen et al.,2024)

El artículo presentado por Krichen, Abdalzaher, Elwekeil y Fouda (2024) destaca que la gestión de desastres naturales es de suma importancia debido a los devastadores impactos que estos eventos pueden tener en la vida humana, la infraestructura y el medio ambiente. En este contexto, las nuevas tecnologías, como el Internet de las Cosas (IoT), desempeñan un papel crucial al ofrecer herramientas innovadoras para la prevención, preparación y respuesta ante emergencias.

El IoT, en conjunto con internet, sensores remotos, teléfonos celulares y redes sociales, proporciona una variedad de aplicaciones que pueden ser vitales en la gestión de desastres naturales, según menciona el artículo. Estas tecnologías permiten un monitoreo continuo y alertas tempranas al recopilar datos en tiempo real sobre condiciones ambientales, niveles de agua, movimientos sísmicos, entre otros. Esto facilita la detección temprana de posibles desastres y la emisión de alertas a las autoridades y la población afectada.

Además, estas herramientas mejoran la comunicación y coordinación durante situaciones de emergencia, permitiendo una difusión rápida y efectiva de información crucial, la coordinación de operaciones de rescate y la asistencia a las personas afectadas. Asimismo, pueden utilizarse para evaluar los daños causados por desastres naturales, lo que permite a las autoridades priorizar acciones de respuesta y asignar recursos de manera eficiente, tal como se menciona en el estudio.

El artículo también subraya las fases de gestión de desastres, cuyo entendimiento es fundamental si se busca desarrollar un prototipo basado en una plataforma IoT para la detección de alertas tempranas ante el desbordamiento de ríos.

Inicialmente se habla de la fase de preparación, esta se enfoca en actividades previas al desastre, como el desarrollo de planes de contingencia, evaluaciones de riesgos y capacitación de equipos de respuesta de emergencia, garantizando la disponibilidad de recursos y personal para una respuesta efectiva.

En segundo lugar se encuentra la fase de respuesta, la cual implica acciones inmediatas para abordar las consecuencias del desastre, como operaciones de búsqueda y rescate, atención médica de emergencia y distribución de recursos básicos. Aunque crucial, coordinar una respuesta rápida puede ser un desafío, según lo destacado en el artículo, especialmente en desastres extensos o que afectan múltiples comunidades.

Estas dos primeras fases conllevan a la fase de recuperación, esta se centra en la restauración de la comunidad afectada a su estado previo al desastre, implicando la reconstrucción de viviendas e infraestructuras, asistencia financiera y restablecimiento de servicios esenciales. Este proceso puede ser lento y desafiante especialmente tras daños significativos.

Finalmente, la fase de mitigación, según se menciona en el artículo, esta fase busca reducir el riesgo de futuros desastres mediante la implementación de códigos de construcción, mejoras en la infraestructura y la investigación de las causas y efectos de los desastres.

2.2.2 Sistema de soporte de decisiones (DSS) en el medio oeste de brasil (mattos et al., 2022)

El proyecto realizado por Mattos et al. (2022) se centra en el desarrollo de un Sistema de Soporte a Decisiones (DSS) junto con una aplicación web de alerta de inundaciones en la cuenca urbana tropical de Prosa, en Brasil. El objetivo principal es enviar mensajes de alerta

temprana de inundaciones a la población común, utilizando tecnologías avanzadas de modelado hidrológico.

En términos de logros, el proyecto logró desarrollar con éxito una aplicación web de alerta de inundaciones que integra un sistema de alerta temprana para la cuenca de Prosa. Se implementaron sistemas de pronóstico de inundaciones y se utilizaron modelos hidrológicos avanzados para predecir y monitorear los niveles de agua.

Este proyecto destaca por varios aspectos clave que pueden servir como fundamentos para la creación de un prototipo basado en una plataforma IoT destinada a la detección de alertas tempranas frente al desbordamiento de ríos.

Primero, se aborda el Modelado Hidrológico Avanzado. Esto implica la utilización de modelos avanzados de hidrología, como el HEC-HMS y el HEC-RAS, para simular el proceso de escorrentía de lluvia y calcular los niveles de agua en una cuenca urbana.

Luego, se destaca el Desarrollo de Aplicaciones Web Interactivas. Aquí, se propone la creación de una aplicación web interactiva y receptiva, aprovechando tecnologías como Progressive Web Application (PWA). Esta aplicación permitirá visualizar datos de inundaciones, mapas y enviar alertas a los usuarios de manera eficaz.

Además, se considera el uso de plataformas de almacenamiento de datos en tiempo real. Se plantea la utilización de bases de datos no relacionales alojadas en plataformas como Google Firebase. Esta elección permite almacenar y acceder a datos actualizados en tiempo real, facilitando una comunicación efectiva entre la aplicación y los usuarios.

Por último, se menciona la implementación de Sistemas de Notificación. Aquí se propone la integración de servicios de mensajería como Google Cloud Messaging. Esto permitirá enviar alertas de riesgo de inundaciones a los usuarios en ubicaciones específicas dentro de la cuenca, mejorando así la capacidad de respuesta ante situaciones de emergencia.

2.2.3 Sistema de soporte de decisiones (DSS) basado en web para la gestión de inundaciones e incendios forestales en áreas urbanas y periurbanas (kochilakis et al.,2016)

Este sistema desarrollado por Kochilakis et al. (2016) utiliza datos hiper temporales de observación de la Tierra de Copernicus Sentinels para modelar el comportamiento de los incendios y evaluar los riesgos de inundaciones en áreas urbanas y periurbanas. El FLIRE DSS también integra datos de información meteorológica para ambos modelos, proporcionando información en tiempo real a usuarios como agencias de Protección Civil y agencias de Bomberos. La participación y aceptación del usuario fueron aspectos clave durante la fase de diseño, con una Comunidad de Práctica implementada para recopilar las necesidades y requisitos de los usuarios. FLIRE DSS se diseñó para ser fácil de usar y proporcionar capacidades GIS sin la necesidad de instalaciones de software complejas y costosas.

El presente artículo resalta la utilidad de emplear información de datos hiper temporales de observación de la Tierra como medida preventiva o de alerta ante posibles inundaciones fluviales. Además, resulta crucial examinar la arquitectura orientada a servicios (SOA) implementada en este sistema para evaluar su viabilidad en la creación de un prototipo basado en una plataforma IoT destinada a la detección temprana de desbordamientos de ríos.

En este sentido, es relevante analizar los protocolos utilizados por dicho sistema, destacando el uso de FTP y HTTP, y determinar su potencial utilidad en nuestro proyecto.

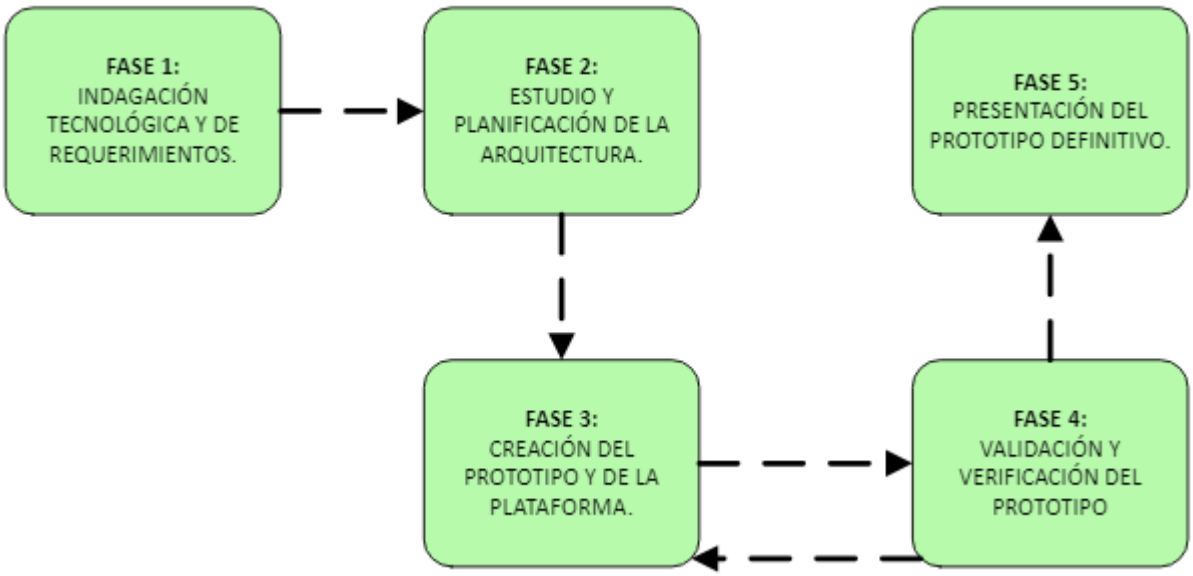
Asimismo, se subraya la importancia de garantizar la disponibilidad continua del sistema, respaldado por sistemas de backup ante cualquier eventualidad. Además, se hace hincapié en la accesibilidad del sistema desde cualquier ubicación sin necesidad de instalaciones adicionales.

3 Marco Metodológico

Para la planificación metodológica del proyecto, se evaluaron distintas metodologías de desarrollo. Al tratarse de un prototipo que combina hardware y software, se optó por un enfoque híbrido. Se siguió principalmente el modelo en cascada por su estructura bien definida, pero en la fase de desarrollo se integró el prototipado, permitiendo iteraciones entre la creación y validación del sistema. Esto hizo posible ajustar sensores, mejorar la transmisión de datos y verificar los requerimientos de manera continua, asegurando un prototipo más preciso y funcional antes de la fase final.

Figura 6.

Esquema de metodología de trabajo.



3.1 Fase 1: Indagación tecnológica y de requerimientos.

En esta etapa, se llevará a cabo una investigación sobre las tecnologías básicas necesarias para la ejecución del proyecto. Se tomarán en cuenta conceptos como IoT, Cloud

Computing, el software y hardware asociados al proyecto. Además de la identificación y el análisis de sus requerimientos.

Actividades

- **A.1.1.** Estudio e indagación de conceptos fundamentales para el desarrollo de un proyecto de IoT.
- **A.1.2.** Evaluación de tecnologías específicas para la implementación del sistema de monitoreo de ríos.
- **A.1.3.** Revisión de casos de estudio pertinentes sobre la detección de desbordamientos fluviales en entornos reales.
- **A.1.4.** Identificación y selección de herramientas esenciales para el desarrollo del prototipo IoT.

Resultados

- **R.1.1.** Adquisición de conocimientos sólidos sobre los conceptos clave para la detección anticipada de desbordamientos fluviales.
- **R.1.2.** Análisis del panorama tecnológico actual aplicable al monitoreo de ríos y alertas tempranas.
- **R.1.3.** Identificación de tecnologías viables para la implementación efectiva del sistema de alerta temprana.
- **R.1.4.** Establecimiento claro de los requisitos esenciales para el desarrollo exitoso del prototipo IoT.

3.2 Fase 2: Estudio y planificación de la arquitectura.

Se llevará a cabo un estudio detallado de la viabilidad y alcance del proyecto para elegir la arquitectura más adecuada para su ejecución. Se definirán los componentes del sistema, incluyendo hardware y software necesarios para la adquisición y monitoreo de los datos.

Actividades

- **A.2.1.** Evaluación de la viabilidad y alcance del proyecto.
- **A.2.2.** Planificación y elaboración del diseño de la plataforma de software específica para el sistema de monitoreo de ríos.
- **A.2.3.** Selección de los indicadores más importantes para evaluar y dar seguimiento al proyecto de alerta temprana.
- **A.2.4.** Definición detallada de la arquitectura necesaria para el prototipo IoT, abarcando dispositivos, conectividad, infraestructura y aplicación.

Resultados

- **R.2.1.** Requisitos necesarios para la arquitectura del prototipo IoT y la plataforma de software.

3.3 Fase 3: Creación del prototipo y de la plataforma.

Se desarrollará el prototipo y la plataforma según lo establecido en las fases anteriores. Se diseñará la interfaz de usuario, se implementará el esquema de sensores y componentes de hardware, se construirá el prototipo a escala y se establecerá la conectividad de los dispositivos con la plataforma web.

Actividades

- **A.3.1.** Diseño y desarrollo de la interfaz de usuario para el prototipo de alertas tempranas de desbordamientos de ríos.
- **A.3.2.** Integración de los sensores y componentes de hardware en el prototipo IoT según el diseño establecido.
- **A.3.3.** Ensamblaje del prototipo a escala siguiendo las especificaciones previas y los requisitos definidos.
- **A.3.4.** Configuración de la red de conectividad de los dispositivos y su vinculación con la plataforma web de alertas tempranas.

Resultados

- **R.3.1.** Arquitectura tecnológica del sistema de alertas tempranas.
- **R.3.2.** Despliegue de la base de datos y la plataforma web para el prototipo de detección de alertas tempranas.
- **R.3.3.** Creación de un prototipo funcional y preparado para pruebas en el contexto de desbordamiento de ríos.

3.4 Fase 4: Validación y verificación del prototipo.

Verificar, a través de pruebas de ejecución, el cumplimiento de los requerimientos del prototipo. Se analizarán los resultados obtenidos para identificar posibles problemas y realizar ajustes en el diseño. Se evaluará la eficacia y eficiencia del prototipo para determinar si es necesario realizar mejoras o ajustes.

Actividades

- **A.4.1.** Implementación del almacenamiento de datos en la nube y su interacción con el prototipo.
- **A.4.2.** Validación de la coherencia y la ejecución del sistema de alertas tempranas.
- **A.4.3.** Validación del prototipo a través de pruebas funcionales, unitarias, de rendimiento y otras pruebas relevantes.
- **A.4.4.** Análisis de los resultados de las pruebas realizadas para identificar posibles problemas y ajustes necesarios.

Resultados

- **R.4.1.** Elaboración de un informe detallado de los resultados obtenidos durante las pruebas y las correcciones para futuras mejoras del prototipo.

3.5 Fase 5: Presentación del prototipo definitivo.

En esta fase, la tesis será documentada oficialmente y presentada ante la Escuela de Ingeniería de Sistemas e Informática de la Universidad Industrial de Santander. El proyecto de grado concluirá así, presentando el prototipo final con las modificaciones realizadas.

Actividades

- **A.5.1.** Presentación del prototipo definitivo con los ajustes necesarios.

Resultados

- **R.5.1.** Prototipo final.

4 Desarrollo del proyecto

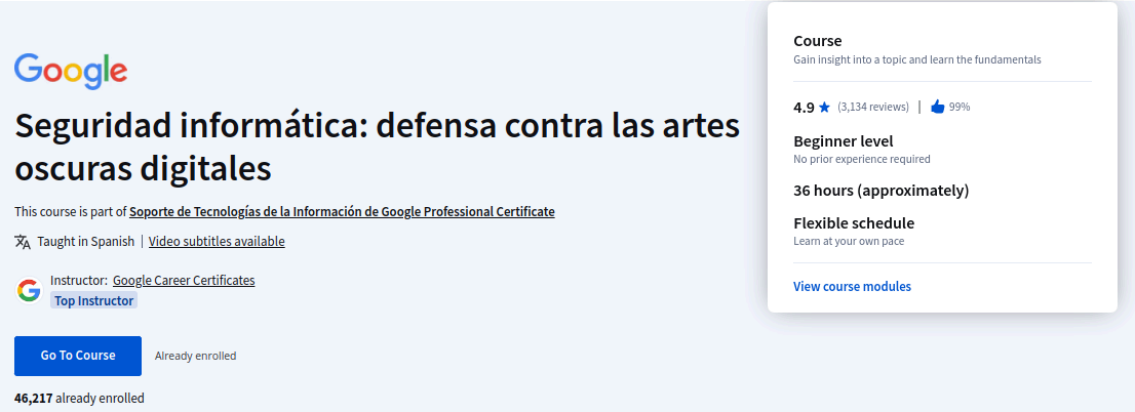
4.1 Capacitación de tecnologías

Con el propósito de llevar a cabo el proyecto utilizando las tecnologías más avanzadas y los conocimientos más recientes en frameworks, además de asegurar la protección y la integridad de los datos involucrados, se recurrió a la realización de diversos cursos especializados, impartidos a través de plataformas de formación en línea como Coursera y Udemy. Estos cursos resultaron fundamentales para adquirir y profundizar en los conocimientos y habilidades necesarios en las tecnologías implementadas en el prototipo. A continuación, se describen los cursos más relevantes.

Uno de los cursos más destacados fue **Seguridad informática: defensa contra las artes oscuras digitales**. Este curso cubrió una amplia gama de conceptos, herramientas y mejores prácticas de seguridad informática, desde la introducción de amenazas y ataques hasta el estudio de algoritmos de cifrado utilizados para proteger datos. También se abordaron los sistemas de seguridad de la información como la autenticación, la autorización y la contabilidad. Este curso permitió adquirir una comprensión sólida sobre cómo implementar una arquitectura de seguridad en capas para proteger eficazmente los datos en un entorno IoT (ver figura 7).

Figura 7.

Curso Seguridad informática: defensa contra las artes oscuras digitales.

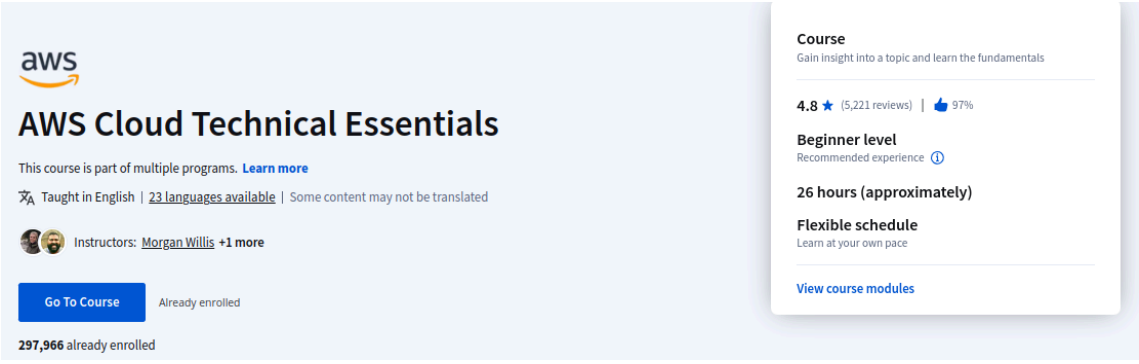


Nota. Tomado de Seguridad informática: defensa contra las artes oscuras digitales. (2021, 24 de septiembre). Coursera. https://www.coursera.org/learn/seguridad-informatica

Otro curso fundamental fue el **AWS Cloud Technical Essentials**, que proporcionó una base sólida en los servicios y productos de Amazon Web Services (AWS). Este curso fue esencial para adquirir los conocimientos necesarios en la nube. (ver figura 8).

Figura 8.

AWS Cloud Technical Essentials.



Nota. Tomado de AWS Cloud Technical Essentials. (2021, 23 de mayo). Coursera. *https://www.coursera.org/learn/aws-cloud-technical-essentials*

El **curso de NodeJS: De cero a experto** fue otro pilar en la formación. Este curso ofreció una comprensión integral de Node.js, una tecnología clave para el desarrollo backend, abarcando desde la creación de aplicaciones de consola hasta la implementación de API RESTful y la autenticación, todo ello utilizando TypeScript. Además, permitió trabajar con bases de datos como MongoDB y PostgreSQL, e incluyó despliegues en plataformas como Railway y la integración de servicios como WebHooks y tareas automáticas con CRON jobs (ver figura 9).

Figura 9.

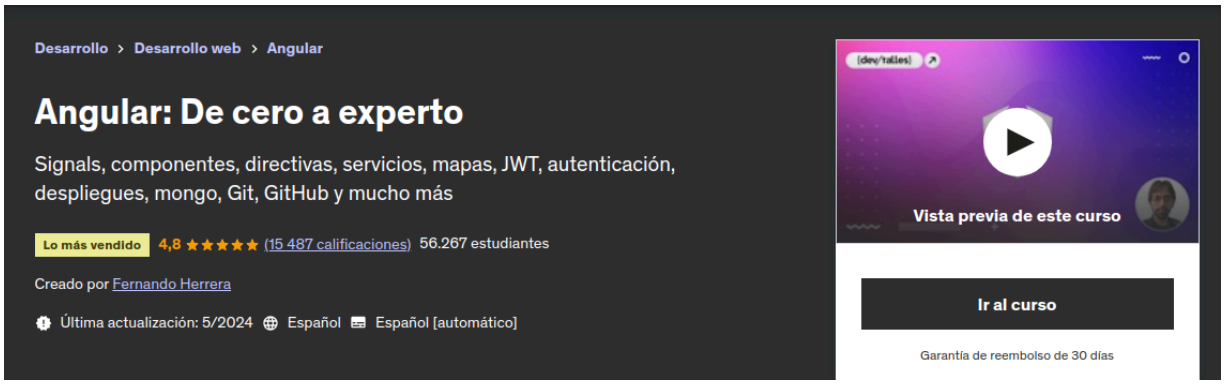
NodeJS: De cero a experto.

Nota. Tomado de Node JS: De cero a experto. (s.f.). Udemy. https://www.udemy.com/course/nodejs-de-cero-a-experto/

En cuanto al desarrollo front-end, el curso **Angular: De cero a experto** resultó esencial para dominar este framework. Se cubrieron aspectos fundamentales como la creación de componentes, servicios y la reutilización de módulos, así como temas avanzados como la autenticación con JSON Web Tokens (JWT) y el consumo de servicios mediante APIs REST (ver figura 10).

Figura 10.

Angular: De cero a experto.

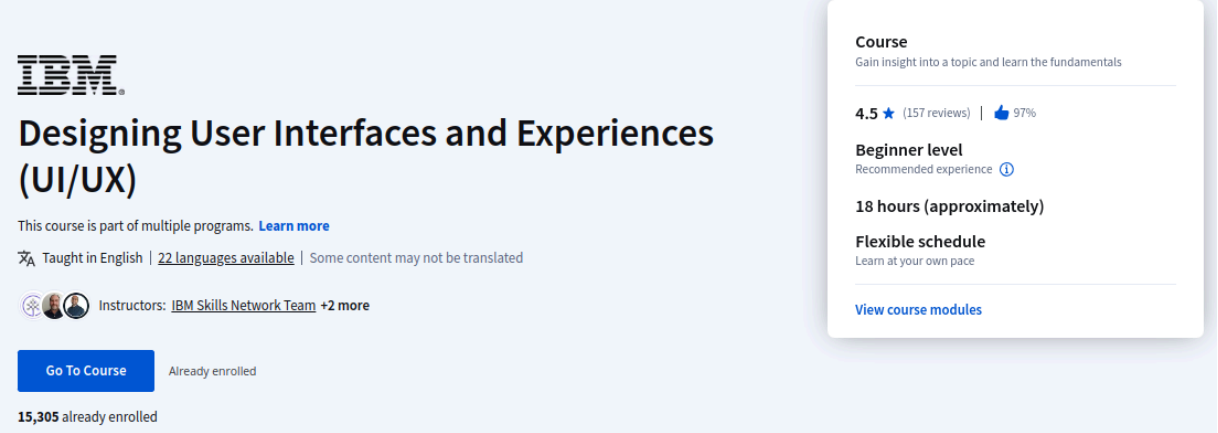


Nota. Tomado de Angular: De cero a experto. (s.f.). Udemy. https://www.udemy.com/course/angular-fernando-herrera/

En el ámbito de las interfaces y experiencias de usuario, se completó el curso de **Diseño de interfaces y experiencias de usuario (UI/UX)**, donde se exploraron las mejores prácticas de diseño visual, incluyendo la teoría del color, la tipografía y la usabilidad. Además, se profundizó en herramientas como Figma, utilizadas para el diseño colaborativo de interfaces (ver figura 11).

Figura 11.

Designing User Interfaces and Experiences (UI/UX).

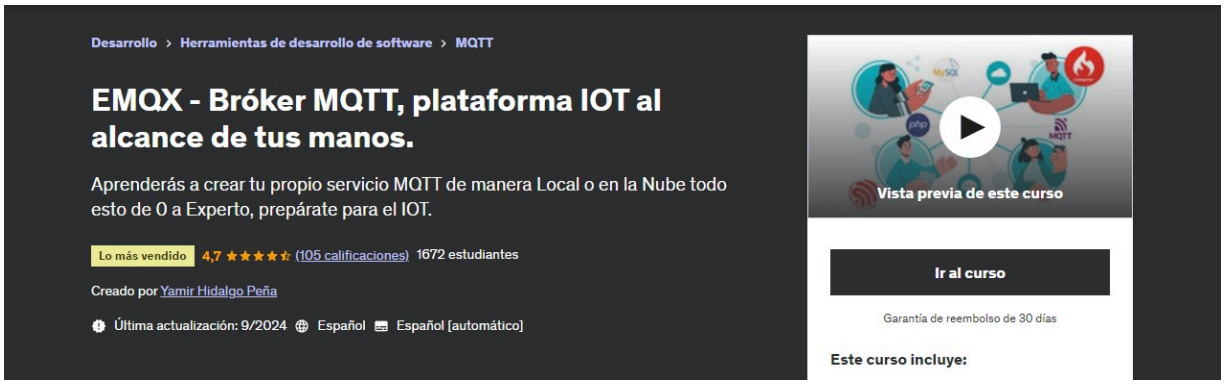


Nota. Tomado de Designing user Interfaces and experiences (UI/UX). (2024, 19 de septiembre 19). Coursera. https://www.coursera.org/learn/designing-user-interfaces-and-experiences-uiux

Complementariamente, el curso de **EMQX - Bróker MQTT, plataforma IoT al alcance de tus manos** proporcionó un conocimiento detallado del protocolo MQTT y su implementación en plataformas IoT. Este curso permitió la configuración de servidores MQTT y la comunicación con dispositivos IoT, facilitando la integración de estos elementos en el prototipo (ver figura 12).

Figura 12.

EMQX - Bróker MQTT, plataforma IOT al alcance de tus manos.



Nota. Tomado de EMQX - Bróker MQTT, plataforma IOT al alcance de tus manos. (s.f.). Udemy. https://www.udemy.com/course/emqx-broker-servidor-iot/

También se realizó el **IoT Bootcamp! Nuxt - Node - Mongo - Emqx**, que ofreció una formación intensiva en el desarrollo de soluciones IoT, combinando tecnologías clave como Docker, AWS, Node.js y MongoDB. Este curso fue crucial para la integración de backend y frontend en la plataforma IoT desarrollada (ver figura 13).

Figura 13.

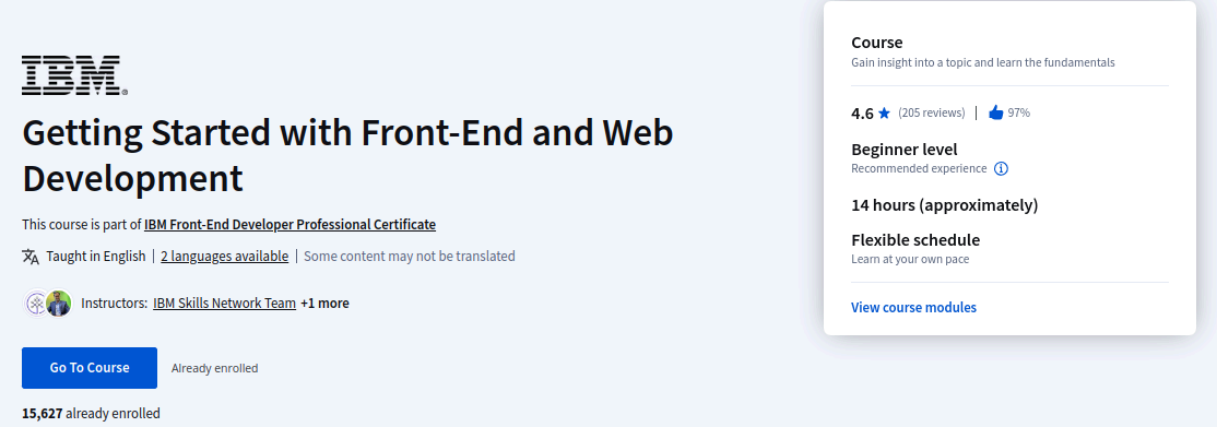
IoT Bootcamp! Nuxt - Node - Mongo - Emqx.

Nota. Tomado de IoT Bootcamp! Nuxt - Node - Mongo - Emqx - ¡Más de 60 horas! (s.f.). Udemy. https://www.udemy.com/course/iot-god-level

Por último, el curso de **Introducción al Desarrollo Front-End y Web Development** proporcionó las bases para crear aplicaciones interactivas y dinámicas, permitiendo un mayor enfoque en el diseño de interfaces web accesibles y de alta calidad (ver figura 14).

Figura 14.

Getting Started with Front-End and Web Development.



Nota. Tomado de Getting Started with Front-End and Web Development. (2024, 22 de abril). Coursera. https://www.coursera.org/learn/getting-started-with-front-end-and-web-development

Estas formaciones fueron complementadas con la revisión de documentación técnica y tutoriales en línea, lo que permitió profundizar en la implementación de tecnologías avanzadas en el prototipo.

4.2 Análisis y definición de arquitectura

4.2.1 Comparación y servicios de arquitectura

En la actualidad, existen múltiples compañías que prestan servicios de computación en la nube. A continuación, se realiza una comparación entre las siguientes plataformas: Amazon Web Services (AWS), Microsoft Azure, Google Cloud, DigitalOcean, Railway, y Vercel, con un análisis detallado de las características principales de cada una y una justificación de la elección final.

4.2.1.1 Amazon Web Services (AWS)

AWS es una de las plataformas más completas y ampliamente utilizadas a nivel mundial. Ofrece una amplia variedad de servicios en áreas como almacenamiento, computación, bases de datos, aprendizaje automático y más. Sus ventajas incluyen:

- **Escalabilidad:** Soporta cargas de trabajo de cualquier tamaño, desde startups hasta empresas de nivel empresarial.
- **Ecosistema amplio:** Más de 200 servicios disponibles.
- **Cobertura global:** Centros de datos en múltiples regiones.

Análisis de costo: Para este proyecto, el servicio de AWS ideal es una combinación de EC2 (Elastic Compute Cloud) y EBS (Elastic Block Store). Estos servicios permiten desplegar el backend, el frontend y la funcionalidad de MQTT de manera flexible y escalable. Además, se ha considerado utilizar Linux como sistema operativo, lo cual es gratuito, lo que contribuye a reducir los costos operativos.

EC2 (Elastic Compute Cloud)

Se ha decidido analizar la instancia c6g.2xlarge de EC2, que es una opción adecuada para este tipo de aplicación. Esta instancia incluye:

- **8 vCPUs (CPUs virtuales):** Suficientes para manejar las solicitudes del backend, frontend y las conexiones de MQTT simultáneas.
- **16 GB de memoria RAM:** Proporciona la capacidad necesaria para ejecutar los servicios de manera eficiente.

- **Network performance de 10 Gbps:** Ideal para una comunicación rápida entre los servicios y con los usuarios finales.

El costo mensual para esta instancia, con cobro por demanda, es de aproximadamente **$99.28 USD** al mes, considerando un uso promedio del **50%**. Es importante tener en cuenta que el uso real puede variar dependiendo del tráfico y la carga del sistema una vez desplegado, lo que puede resultar en un costo mayor o menor.

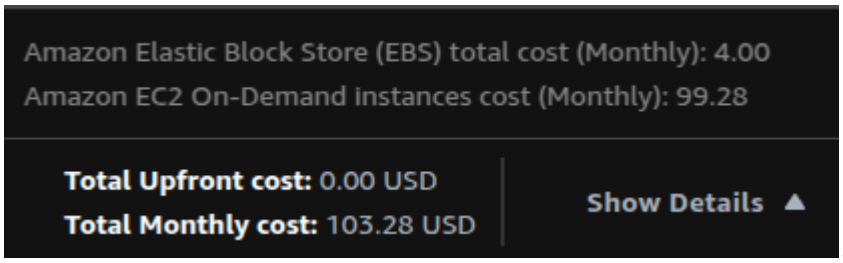
EBS (Elastic Block Store)

Para el almacenamiento, se han definido 100 GB de almacenamiento utilizando el servicio EBS (gp3). Los volúmenes gp3 ofrecen un buen balance entre rendimiento y precio, siendo ideales para aplicaciones que necesitan un almacenamiento rápido y de bajo costo. El costo aproximado para 100 GB con gp3 es de **$4.00 USD** al mes.

General Purpose SSD (gp3): Este tipo de almacenamiento ofrece un rendimiento de hasta 16,000 IOPS y 1,000 MB/s de rendimiento, lo cual es más que suficiente para soportar el funcionamiento del backend, frontend y MQTT.

Figura 15.

Calculadora de precios AWS.



Nota. Tomado de AWS Pricing Calculator. (s.f.).

https://calculator.aws/#/createCalculator/ec2-enhancement

4.2.1.2 Microsoft Azure

Azure destaca por su integración con herramientas empresariales y el ecosistema de Microsoft, como Office 365, Windows Server y Active Directory. Ventajas clave:

- **Compatibilidad**: Ideal para organizaciones que ya utilizan productos de Microsoft.
- **Variedad de servicios**: Ofrece soluciones avanzadas para inteligencia artificial, análisis de datos y computación en la nube.
- **Descuentos educativos y sin fines de lucro**: Opciones atractivas para reducir costos en algunos escenarios.

Análisis de costo: Para este proyecto, el servicio ideal en Azure es el uso de Azure Virtual Machines. Estas máquinas virtuales permiten desplegar tanto el backend, el frontend como la funcionalidad de MQTT de manera flexible y escalable.

Azure Virtual Machines

La instancia seleccionada para este proyecto tiene las siguientes características:

- **8 núcleos de CPU:** Adecuados para gestionar el procesamiento necesario para el backend, frontend y conexiones de MQTT.
- **32 GB de RAM:** Suficientes para manejar el procesamiento de datos, ejecución de servicios y solicitudes simultáneas.
- **64 GB de almacenamiento temporal:** Utilizado para almacenamiento efímero durante el funcionamiento de la instancia, ideal para archivos temporales y otros datos no permanentes.

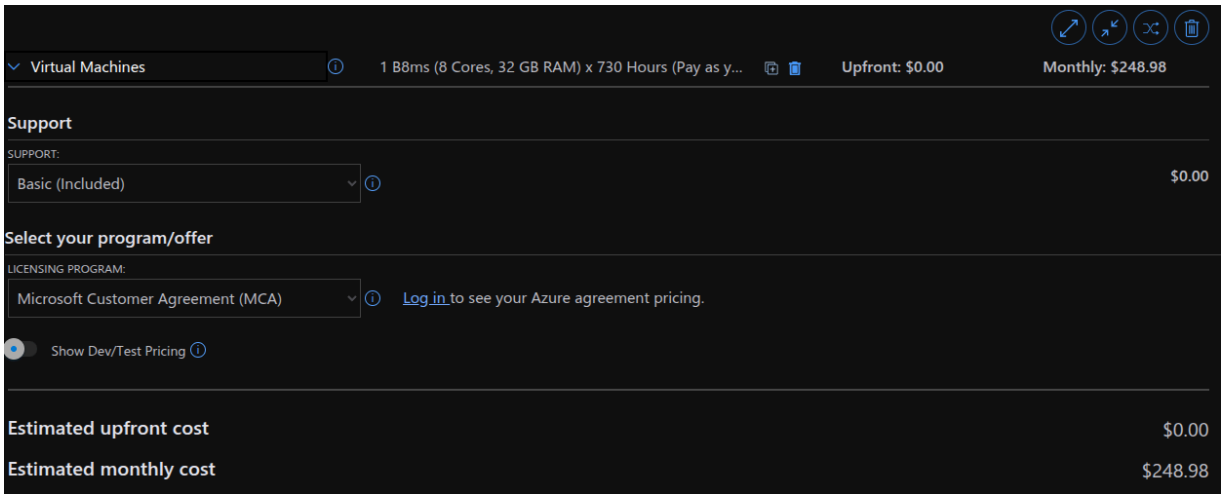
Esta configuración se considera óptima para el proyecto, con un costo estimado de **$243.09 USD** al mes en modalidad de pago por uso. Este modelo de pago es flexible, ya que solo se paga por el uso real de los recursos, aunque el costo puede variar dependiendo de la carga y el uso de la máquina virtual.

Almacenamiento

Para el almacenamiento, se ha seleccionado un disco Standard HDD con 128 GB de capacidad. El costo mensual para este tipo de disco es de **$5.89 USD**. El almacenamiento de tipo HDD es adecuado para proyectos que no requieren un rendimiento extremadamente alto, lo que lo convierte en una opción rentable para almacenamiento de datos persistentes en este proyecto.

Figura 16.

Calculadora de precios Azure.



Nota. Tomado de Pricing - Linux Virtual Machines | Microsoft Azure. (s.f.). Microsoft Azure. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/#pricing

4.2.1.3 Google Cloud

Google Cloud se caracteriza por su enfoque en el análisis de datos, inteligencia artificial y aprendizaje automático. También ofrece servicios de almacenamiento, redes y computación. Ventajas:

- **Tecnología avanzada**: Herramientas innovadoras como BigQuery y TensorFlow.
- **Precios competitivos**: Opciones flexibles con descuentos por compromiso a largo plazo.
- **Infraestructura global**: Alto rendimiento y confiabilidad.

Análisis de costo: Para este proyecto, el servicio ideal de Google Cloud es el uso de Google Compute Engine, que permite ejecutar máquinas virtuales escalables y eficientes para el backend, el frontend y la funcionalidad de MQTT.

Google Compute Engine

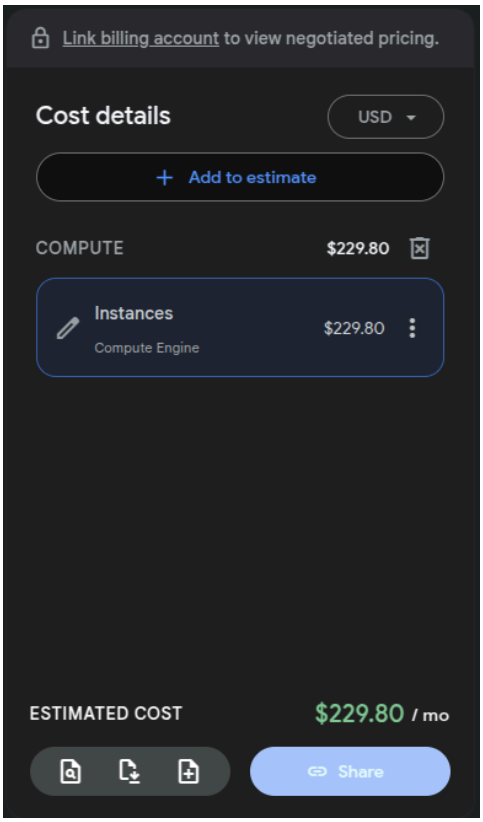
La instancia seleccionada para este proyecto tiene las siguientes características:

- **Sistema operativo Linux:** Gratuito, lo que reduce los costos operativos.
- **Familia de máquinas General Purpose, serie N1:** La opción ideal para cargas de trabajo que no requieren configuraciones especializadas, pero que necesitan un rendimiento equilibrado.
- **8 vCPUs:** Proporciona suficiente poder de procesamiento para manejar las tareas simultáneas del backend, frontend y las conexiones de MQTT.
- **8 GB de RAM:** Memoria adecuada para ejecutar los servicios de manera eficiente sin sobrecargar los recursos.
- **Disco de 100 GB:** Almacenamiento incluido en la instancia para guardar datos temporales y persistentes.

El costo mensual de esta instancia es de aproximadamente $229.80 USD, lo que la convierte en una opción competitiva en términos de costo y rendimiento para el proyecto.

Figura 17.

Calculadora de precios Google Cloud.



Nota. Tomado de Calculadora de precios de Google Cloud. (s.f.). https://cloud.google.com/products/calculator?hl=es_419&dl=CjhDaVExWWpoak5qSmpZeTAzWXpGaExUUTVNVEF0WVdaallpMWpOVEppTUdWbVltTTNZV1lRQVE9PRAIGiQ4QjEwNUFERi0zRjZCLTRCOEItOThDMy00OTNFMUYwNkMwNTI

4.2.1.4 DigitalOcean

DigitalOcean está orientada principalmente a desarrolladores y pequeñas empresas. Sus características más destacadas incluyen:

- **Simplicidad**: Interfaz amigable y fácil de usar.
- **Planes accesibles**: Diseñados para proyectos pequeños y medianos.

- **Desempeño sólido**: Una opción confiable para cargas de trabajo moderadas.

Análisis de costo: Para este proyecto, el servicio ideal de DigitalOcean es el uso de Droplets, que son máquinas virtuales optimizadas para proyectos de desarrollo y aplicaciones web.

DigitalOcean Droplets

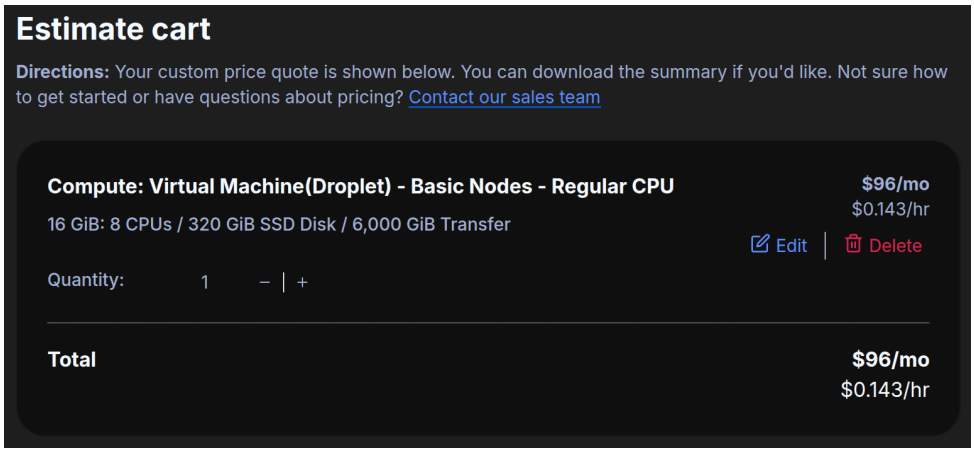
La máquina seleccionada para este proyecto tiene las siguientes características:

- **8 vCPUs:** Adecuada para manejar el procesamiento de las solicitudes del backend, frontend y las conexiones MQTT simultáneas.
- **320 GB de SSD:** Espacio de almacenamiento rápido y eficiente para el almacenamiento de datos y archivos del proyecto.
- **6000 GB de transferencia de datos:** Este amplio límite de transferencia asegura que el tráfico de la aplicación pueda gestionarse sin preocupaciones por exceder el ancho de banda.

El costo mensual para esta máquina virtual es de $96 USD, lo que la convierte en una opción accesible y confiable para proyectos medianos a grandes, con un rendimiento sólido y un costo competitivo.

Figura 18.

Calculadora de precios Digitalocean.



Nota. Tomado de Pricing Calculator | DigitalOcean. (s.f.). https://www.digitalocean.com/pricing/calculator#cart=%5B%7B%22slug%22%3A%22s-8vcpu-16gb%22%2C%22type%22%3A%22droplets%22%2C%22options%22%3A%7B%22type%22%3A%22basic%22%2C%22cpu%22%3A%22regular%22%2C%22blockStorage%22%3A0%7D%2C%22quantity%22%3A1%7D%5D

4.2.1.5 Railway

Railway es una plataforma de computación en la nube que sobresale por su simplicidad y enfoque en despliegues rápidos. Es ideal para startups, proyectos personales y aplicaciones en etapas iniciales, aunque también ofrece opciones para escalabilidad. Sus características más destacadas incluyen:

- **Despliegues rápidos y sencillos:** Configuración mínima y CI/CD integrado con GitHub para implementar proyectos de manera automática.
- **Planes flexibles:** Desde opciones gratuitas (Trial) hasta planes escalables como Pro y Enterprise, adecuados para aplicaciones más grandes.

- **Facilidad de integración:** Compatibilidad con diversos lenguajes y frameworks, lo que agiliza el desarrollo en equipos pequeños o grandes.

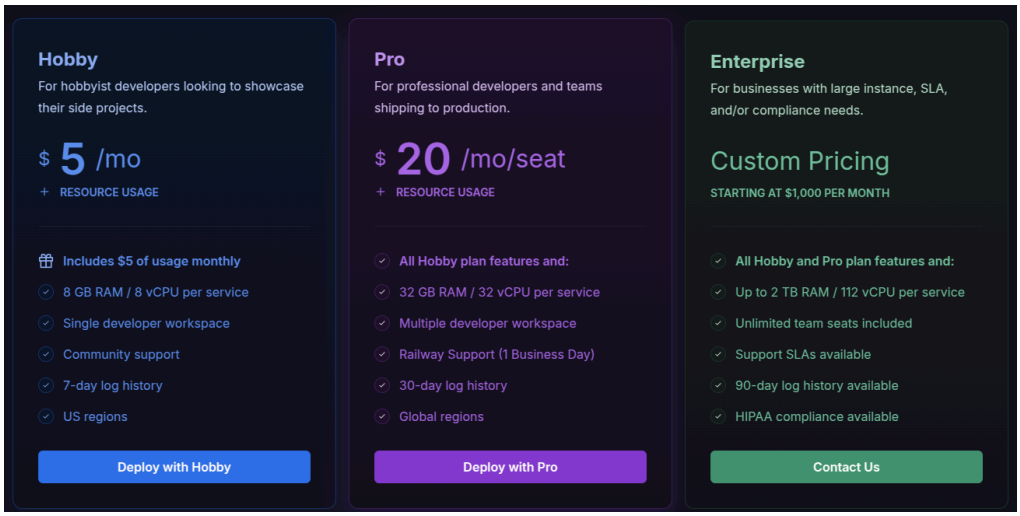
Análisis de costo: Railway ofrece varios planes con diferentes capacidades para ajustarse a las necesidades de distintos proyectos:

- **Trial:** 0.5 GB de RAM, 2 vCPU, 1 GB de almacenamiento efímero y 0.5 GB de almacenamiento en volumen. Ideal para pruebas o proyectos pequeños.
- **Hobby:** 8 GB de RAM, 8 vCPU, 10 GB de almacenamiento efímero y 5 GB de almacenamiento en volumen. Es una excelente opción para proyectos que no requieren una infraestructura demasiado grande, pero que aún necesitan recursos significativos para una carga constante.
- **Pro:** 32 GB de RAM, 32 vCPU, 100 GB de almacenamiento efímero y 50 GB de almacenamiento en volumen. Es ideal para proyectos más grandes y escalables.
- **Enterprise:** 64 GB de RAM, 64 vCPU, 100 GB de almacenamiento efímero y 50 GB de almacenamiento en volumen. Apto para grandes organizaciones con necesidades avanzadas de infraestructura.

El plan **Hobby** de Railway se presenta como una excelente opción, ya que proporciona una capacidad adecuada para la mayoría de los proyectos a un costo razonable, sin necesidad de optar por planes más avanzados como el **Pro** o el **Enterprise**.

Figura 19.

Planes Railway.



Nota. Tomado de Pricing. (s.f.). Railway. https://railway.com/pricing

4.2.1.6 Vercel

Vercel es una plataforma diseñada específicamente para aplicaciones web frontend. Sus principales ventajas incluyen:

- **Gratuito:** Ideal para pequeños proyectos con recursos limitados.
- **Integración con repositorios:** CI/CD integrado con GitHub o GitLab.
- **Optimización para frameworks modernos:** Herramientas avanzadas para aplicaciones basadas en Next.js y otros frameworks populares.

Análisis de costo: Vercel ofrece varios planes con diferentes capacidades para ajustarse a las necesidades de distintos proyectos:

- **Hobby:** Plan gratuito, ideal para proyectos pequeños o personales. Ofrece importación de repositorios, CI/CD automático, cómputo serverless, mitigación de DDoS, firewall

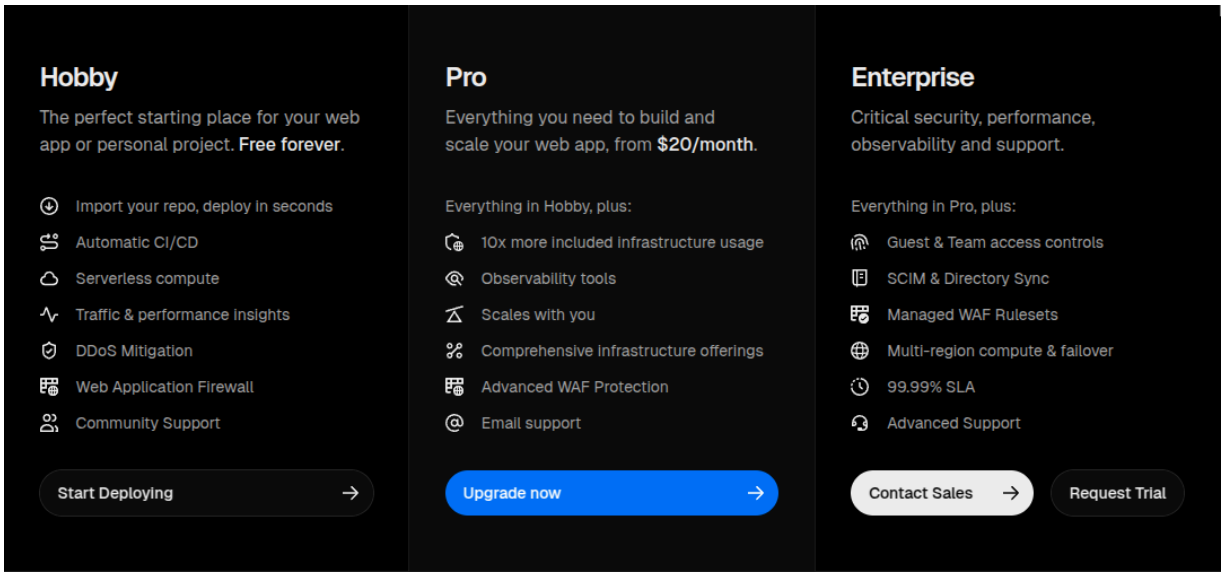
de aplicaciones web y soporte comunitario. Es perfecto para desplegar aplicaciones frontend de manera sencilla y segura.

- **Pro:** Desde $20 al mes. Incluye todo lo del plan Hobby, además de 10 veces más infraestructura, herramientas de observabilidad, escalabilidad y soporte por correo electrónico.
- **Enterprise:** Para empresas grandes, con características avanzadas como controles de acceso, seguridad mejorada, computación multi-región, y un SLA del 99.99%.

El plan Hobby de Vercel es una opción ideal para desarrolladores y proyectos que requieren un despliegue rápido y gratuito de aplicaciones web, sin necesidad de mayores recursos.

Figura 20.

Planes Vercel.



Nota. Tomado de Find a plan to power your projects. (s.f.). Vercel. https://vercel.com/pricing

4.2.1.7 Conclusión del Análisis Comparativo

En base al análisis realizado, se ha seleccionado el plan Hobby de Railway para el backend, la base de datos y MQTT, debido a su capacidad adecuada para manejar la carga del proyecto y su costo accesible de $5 USD al mes. Este plan se ajusta a las necesidades de recursos sin necesidad de optar por planes más grandes, lo que lo convierte en una excelente opción para proyectos pequeños y medianos.

Por otro lado, se ha seleccionado el plan Hobby de Vercel, que es gratuito, para desplegar el frontend desarrollado en Angular. La integración fluida con GitHub y el CI/CD automático son características clave que facilitarán los despliegues rápidos y efectivos.

4.2.2 Comparación de lenguajes para el Backend

En el desarrollo de una plataforma IoT para la detección de alertas tempranas, la selección del lenguaje de programación para el backend es un aspecto crucial para garantizar la eficiencia, escalabilidad y fiabilidad del sistema. Dado que el prototipo debía gestionar múltiples datos en tiempo real provenientes de sensores, el lenguaje backend debía ofrecer tanto un rendimiento óptimo como la capacidad de manejar grandes volúmenes de solicitudes simultáneas. En este análisis, se evaluaron lenguajes de programación como Python, Java, Ruby, Go, PHP y TypeScript, concluyendo con la elección de este último debido a sus características sobresalientes.

Python ha ganado popularidad por su simplicidad y su amplio ecosistema de bibliotecas, lo que lo convierte en una opción ideal para proyectos que requieren rapidez en el desarrollo y análisis de datos. Sin embargo, al tratarse de un lenguaje interpretado, este

presenta limitaciones en términos de rendimiento, lo cual puede ser un inconveniente en aplicaciones IoT que requieren una respuesta en tiempo real. Si bien Python es adecuado para prototipos rápidos, su capacidad para manejar de manera eficiente grandes cantidades de datos simultáneamente es limitada en comparación con otras alternativas más especializadas.

Java, por otro lado, destaca por su robustez y escalabilidad, siendo una opción confiable en proyectos empresariales de gran envergadura. Su capacidad para manejar concurrencia y su rendimiento eficiente son atributos que lo hacen atractivo para aplicaciones IoT. Los desarrolladores del proyecto ya están familiarizados con Java, lo que facilita su adopción en entornos de trabajo que requieren soluciones seguras y bien estructuradas. No obstante, para este tipo de proyectos, donde se requiere mayor agilidad en la creación de prototipos, Java puede resultar más pesado en términos de velocidad de desarrollo comparado con otras opciones más ligeras.

Ruby, reconocido por su enfoque en la productividad del desarrollador y su framework Ruby on Rails, facilita el desarrollo rápido de aplicaciones web. Sin embargo, cuando se trata de gestionar múltiples conexiones simultáneas o manejar concurrencia en tiempo real, Ruby no ofrece el rendimiento adecuado para plataformas IoT. A pesar de su flexibilidad, la falta de herramientas optimizadas para el manejo de datos en tiempo real reduce su viabilidad en proyectos de esta naturaleza.

Por su parte, Go se ha destacado en el desarrollo de sistemas distribuidos y microservicios gracias a su diseño simplificado y eficiente manejo de la concurrencia. Este lenguaje, creado por Google, es ideal para proyectos que requieren alta escalabilidad y rendimiento en tiempo real, cualidades esenciales en plataformas IoT. Sin embargo, la falta de

conocimientos y experiencia en Go por parte de los integrantes del equipo de desarrollo representa un obstáculo significativo para su adopción en este proyecto, lo que lo convierte en una opción menos viable a pesar de sus ventajas técnicas.

PHP, ampliamente utilizado en el desarrollo web, ha sido tradicionalmente elegido para proyectos con alta interacción con bases de datos y gestión de contenidos. Aunque su rendimiento ha mejorado con el tiempo y frameworks como Laravel lo han vuelto más competitivo, PHP presenta limitaciones en aplicaciones IoT que requieren procesamiento eficiente en tiempo real y manejo de una alta concurrencia de conexiones. Su diseño y arquitectura están más orientados hacia aplicaciones web tradicionales que hacia arquitecturas escalables y distribuidas, lo que puede dificultar su capacidad para satisfacer las exigencias de las plataformas IoT.

Finalmente, se evaluó TypeScript, un superset de JavaScript que añade tipado estático opcional, proporcionando un entorno más estructurado y seguro para el desarrollo. Su integración con Node.js y Express.js, junto con su compatibilidad con tecnologías modernas como WebSockets, lo posicionó como una excelente opción para manejar datos en tiempo real en aplicaciones IoT. Aunque para el momento de la selección aún no se había iniciado el desarrollo del proyecto, se consideró que TypeScript ofrecía una combinación de herramientas que permitirían alcanzar los objetivos propuestos. Su sistema de tipado estático y las características avanzadas de su ecosistema brindaban un balance entre velocidad de desarrollo, escalabilidad y mantenibilidad del código, aspectos fundamentales para un proyecto de estas características.

La agilidad en el desarrollo que ofrece TypeScript, en conjunto con su capacidad para escalar de manera eficiente, permitió cumplir con los objetivos del proyecto. Se necesitaba una solución que no sólo ofreciera un rendimiento adecuado, sino que también facilitara la rápida creación de un prototipo funcional. El uso de TypeScript permitió construir una plataforma ágil y adaptable, capaz de gestionar las complejidades inherentes a una plataforma IoT, desde el manejo de múltiples sensores hasta la generación de alertas en tiempo real. En definitiva, TypeScript se eligió no solo por su rendimiento, sino por su capacidad para equilibrar la velocidad de desarrollo con la creación de un código sólido y mantenible, asegurando así el éxito del proyecto.

Figura 21.

Fragmento de código backend.

```
start() {

    //* Middlewares
    this.app.use( express.json() ); // raw
    this.app.use( express.urlencoded({ extended: true })); // x-www-form-urlencoded
    this.app.use( compression() );

    //* Enable CORS
    this.app.use(cors({
        origin: this.frontendOrigin
    }));

    // Swagger
    this.app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerSpecs))

    //* Routes
    this.app.use(this.routes);


    this.app.listen(this.port, () => {
        console.log(`Server running on port ${this.port}`);
    });
}
```

4.2.3 Comparación de bases de datos

La elección de la base de datos adecuada depende de múltiples factores, tales como la estructura de los datos, las necesidades de escalabilidad, el tipo de consultas que se realizarán, y los requisitos de rendimiento. A continuación, se presenta una comparación entre varias bases de datos relacionales y no relacionales, considerando sus características principales y las ventajas que ofrecen.

4.2.3.1 Bases de Datos Relacionales

Las bases de datos relacionales (RDBMS, por sus siglas en inglés) son ideales para aplicaciones que requieren un esquema estructurado y relaciones complejas entre los datos. Estas bases de datos utilizan tablas y claves primarias para organizar y gestionar los datos.

4.2.3.1.1 PostgreSQL

Es una base de datos conocida por su robustez y flexibilidad. Ofrece un alto soporte para tipos de datos complejos y funciones avanzadas. A diferencia de otras bases de datos, PostgreSQL es compatible con datos semiestructurados gracias a su capacidad para manejar JSON y otros tipos de datos no convencionales. Es una opción popular para aplicaciones que requieren transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo que garantiza que los datos siempre se mantengan consistentes. Además, permite realizar consultas complejas y analíticas, lo que la convierte en una excelente opción para aplicaciones que necesitan analizar grandes volúmenes de datos.

4.2.3.1.2 MySQL

Es una base de datos popular en aplicaciones web debido a su simplicidad y facilidad de uso. A pesar de ser menos compleja que PostgreSQL, MySQL ofrece una solución robusta para gestionar datos en aplicaciones de lectura intensiva. Ofrece soporte para transacciones ACID, lo que garantiza que las operaciones de escritura sean seguras y consistentes. MySQL es altamente accesible y tiene una gran comunidad de soporte, lo que la hace ideal para desarrolladores que buscan una opción fácil de integrar con plataformas como PHP, Python y Ruby.

4.2.3.1.3 Microsoft SQL Server

Es una base de datos empresarial diseñada para entornos grandes y complejos. Ofrece transacciones ACID y una serie de características avanzadas como procesamiento analítico en línea (OLAP) y herramientas de inteligencia empresarial (BI). SQL Server se integra muy bien con otras herramientas de Microsoft, lo que lo hace adecuado para empresas que ya utilizan soluciones de Microsoft. Su coste puede ser elevado, ya que requiere licencias comerciales para acceder a su funcionalidad completa.

4.2.3.2 Bases de datos No Relacionales (NoSQL)

Las bases de datos NoSQL son adecuadas para aplicaciones que necesitan manejar grandes volúmenes de datos no estructurados o semiestructurados, y ofrecen una mayor flexibilidad en cuanto a esquema.

4.2.3.2.1 MongoDB

Es una base de datos que destaca por su flexibilidad y escalabilidad. Almacena los datos en formato BSON (similar a JSON), lo que permite almacenar estructuras de datos complejas de manera eficiente. Esta característica la convierte en una opción ideal para aplicaciones que gestionan datos semiestructurados o que requieren cambios frecuentes en el esquema. MongoDB es altamente escalable, permitiendo la expansión horizontal a través de particionamiento (sharding) para manejar grandes volúmenes de datos. A diferencia de las bases de datos relacionales, MongoDB ofrece un rendimiento superior en operaciones de lectura y escritura de alta velocidad, lo cual es esencial para aplicaciones con altos niveles de tráfico.

4.2.3.2.2 Cassandra

Es una base de datos diseñada para manejar grandes volúmenes de datos distribuidos de manera eficiente. Su modelo de datos basado en columnas la hace adecuada para aplicaciones que necesitan escribir y leer datos rápidamente en entornos distribuidos. Cassandra es famosa por su capacidad de escalabilidad horizontal y alta disponibilidad, ya que permite distribuir los datos a través de múltiples nodos sin perder rendimiento. Es ideal para aplicaciones que requieren tolerancia a fallos y una consistencia eventual, lo que significa que los datos pueden no ser consistentes de inmediato pero se garantizará que se sincronicen a largo plazo.

4.2.3.2.3 Redis

Es una base de datos clave-valor en memoria que se utiliza principalmente para almacenar datos temporales, como cachés o colas de mensajes. Su rendimiento es increíblemente rápido debido a que almacena los datos directamente en la memoria RAM, lo que la hace ideal para aplicaciones que requieren acceso a datos en tiempo real. Aunque Redis no está diseñado para ser una base de datos persistente, ofrece un rendimiento sin igual en aplicaciones que necesitan realizar lecturas o escrituras rápidas. Además, soporta varias estructuras de datos complejas como listas, conjuntos y mapas.

4.2.3.3 Resultado del Análisis Comparativo

MongoDB ha sido seleccionada como la base de datos ideal para este proyecto debido a su capacidad para manejar datos generados por sensores en tiempo real, como los de nivel del río, flujo de agua y lluvia. Su formato flexible basado en JSON permite adaptarse fácilmente a nuevas variables sin necesidad de un esquema rígido. Además, ofrece alta disponibilidad y tolerancia a fallos a través de la replicación, lo que asegura la continuidad del sistema en situaciones críticas. Su escalabilidad horizontal también la hace adecuada para el creciente volumen de datos de múltiples estaciones IoT, mientras que su integración con tecnologías como Node.js y Docker facilita el desarrollo y despliegue del sistema. Finalmente, MongoDB permite un procesamiento eficiente tanto de datos históricos como en tiempo real, lo cual es esencial para el análisis y las alertas del sistema.

4.2.4 Comparación de dispositivos y sensores necesarios para el hardware

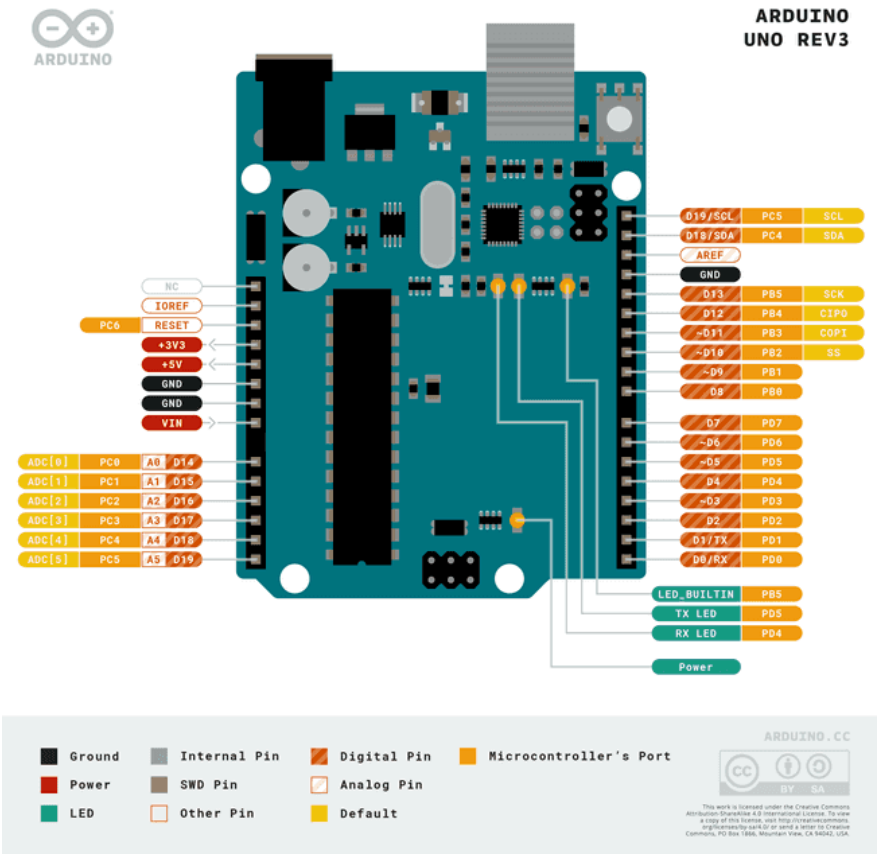
4.2.4.1 Placas de desarrollo

Considerando los dispositivos más utilizados en proyectos de IoT, se evaluaron diversas placas de desarrollo, entre ellas Arduino Uno, DOIT Esp32 DevKit v1 y Raspberry Pi Pico.

- Arduino Uno: Basado en el microcontrolador ATmega328P, el Arduino Uno opera con una frecuencia de 16 MHz e incluye 14 pines digitales de entrada/salida, de los cuales 6 soportan modulación por ancho de pulso (PWM); además, cuenta con 6 entradas analógicas, un conector ICSP, un botón de reinicio y un puerto USB para su programación y alimentación (Arduino, n.d.). Su rango de voltaje de trabajo es de 5V, con entradas que varían entre 7 y 12V. Es una de las placas más utilizadas en proyectos de IoT debido a su robustez, amplia documentación y comunidad activa; su costo promedio en el mercado oscila entre los 20 y 25 dólares, haciéndola una opción accesible y confiable para principiantes y expertos en microcontroladores.

Figura 22.

Diagrama de pines Arduino UNO.



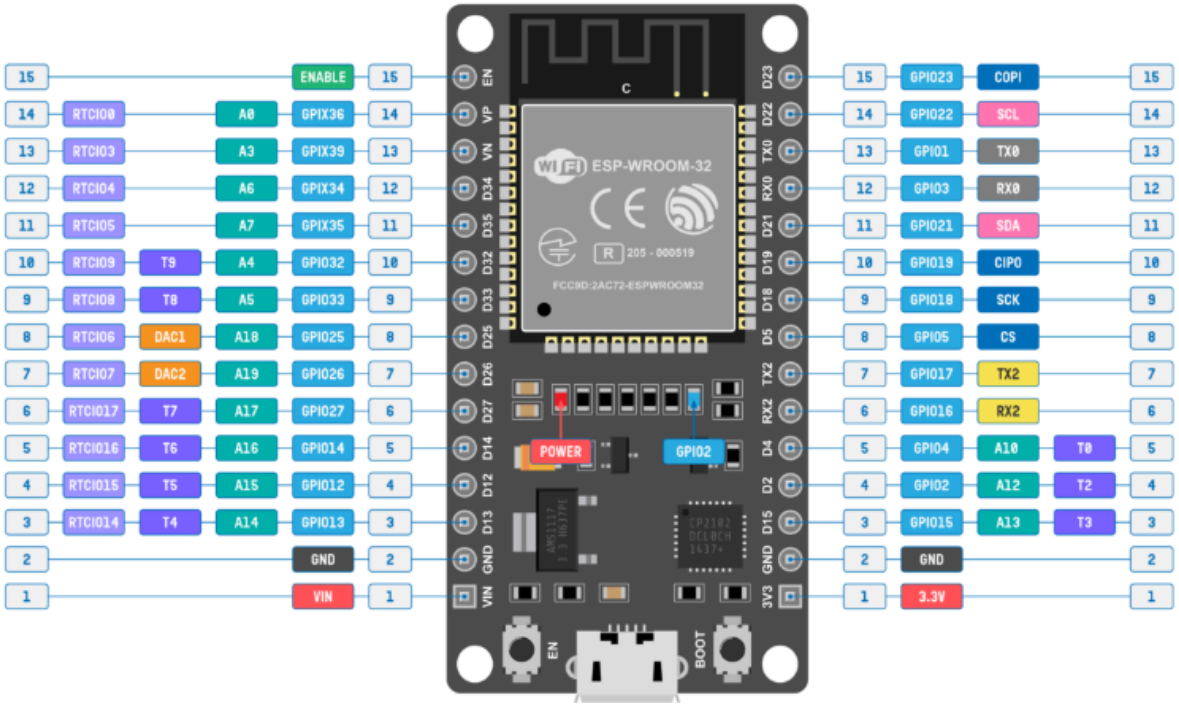
Nota. Tomado de Arduino UNO Rev3 with Long Pins. (s.f.). Docs.Arduino. https://docs.arduino.cc/retired/boards/arduino-uno-rev3-with-long-pins/

- DOIT Esp32 DevKit v1: Esta placa de desarrollo está basada en el popular módulo ESP32-WROOM-32, equipado con un procesador Xtensa LX6 de 32 bits y doble núcleo, operando entre 40 MHz y 160 MHz, pero alcanzando hasta 240 MHz. Soporta temperaturas de operación de -40°C a 125°C y cuenta con 520 KiB de SRAM, 448 KiB de ROM, y 4 MiB de memoria flash integrada. Además, incorpora conectividad Wi-Fi y Bluetooth de 2.4 GHz, 36 pines GPIO, 16 canales ADC de 12 bits, 16 canales PWM, 10 pines táctiles, y múltiples interfaces de comunicación como UART, I2C y

SPI (CircuitState, 2022). Se alimenta con 3.3V y su costo en el mercado oscila entre 5 y 6 dólares.

Figura 23.

Diagrama de pines DOIT ESP32 DevKit V1.



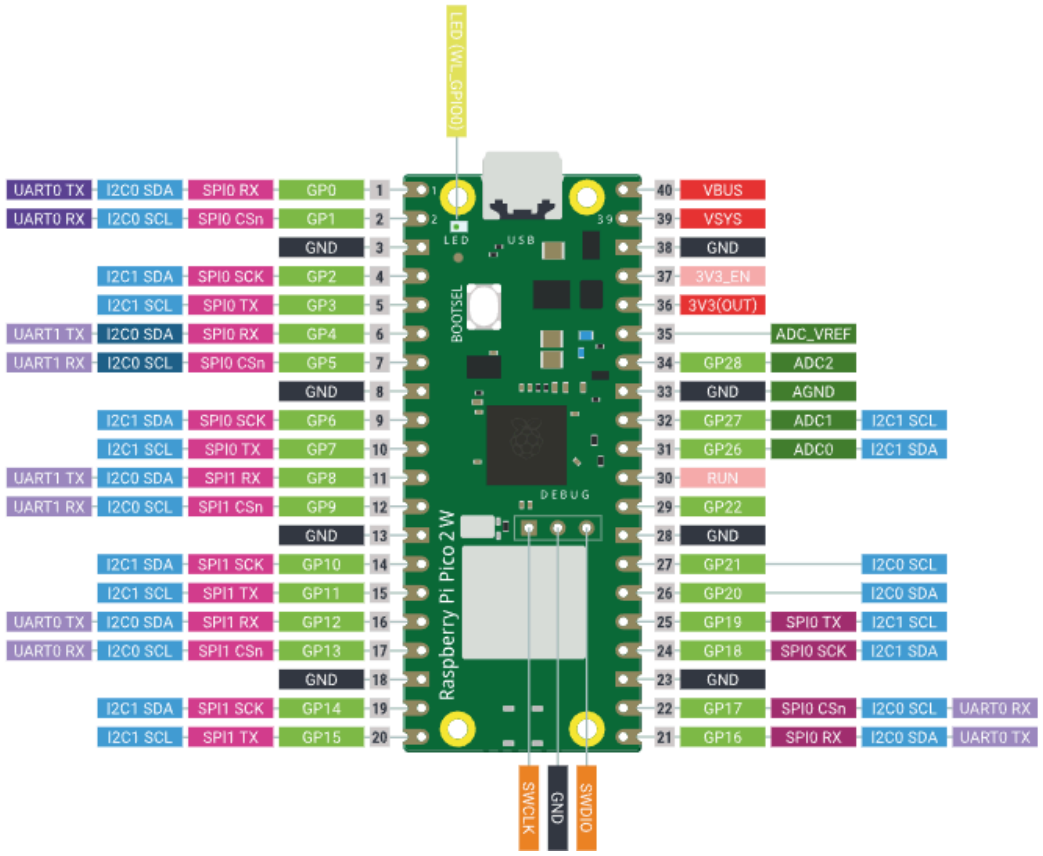
Nota. Tomado de Mohanan, V. (2024, 12 de marzo). DOIT ESP32 DevKit V1 Wi-Fi Development Board - Pinout Diagram & Arduino Reference - CIRCUITSTATE. CIRCUITSTATE Electronics. https://www.circuitstate.com/pinouts/doit-esp32-devkit-v1-wifi-development-board-pinout-diagram-and-reference/.

- Raspberry Pi Pico: Está construido con el microcontrolador RP2040 diseñado por Raspberry Pi, que incluye un procesador dual-core ARM Cortex-M0+ con una velocidad de operación de hasta 133 MHz, 264 KB de RAM interna y soporte para

hasta 16 MB de memoria Flash externa mediante Quad-SPI. Además, cuenta con 26 pines GPIO multifuncionales, de los cuales 3 son entradas analógicas (ADC) con resolución de 12 bits. Este dispositivo admite comunicación mediante dos puertos I2C, SPI, y UART, además de ofrecer 16 canales PWM y 8 máquinas de estado PIO programables para soporte de periféricos personalizados. Su alimentación de trabajo es de 3.3V y soporta un voltaje de entrada entre 1.8 y 5.5V (Raspberry Pi, 2024). El costo promedio del Raspberry Pi Pico varía entre 4 y 6 dólares, dependiendo de la variante adquirida.

Figura 24.

Diagrama de pines Raspberry Pi Pico 2 W.



Nota. Tomado de Pico-Series Microcontrollers - Raspberry Pi Documentation. (s.f.). https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html

Teniendo en cuenta la facilidad de conexión, la amplia disponibilidad de librerías compatibles y el conocimiento previo en el manejo del ESP32, se decidió optar por esta placa como la principal para el desarrollo del proyecto. Además, ya contábamos con unidades ESP32 disponibles, lo que representó una ventaja en términos de costos y logística. Al evaluar las características técnicas, se comprobó que esta placa cumple perfectamente con los requerimientos necesarios para alcanzar los objetivos del proyecto de manera eficiente y confiable.

4.2.4.2 Sensores

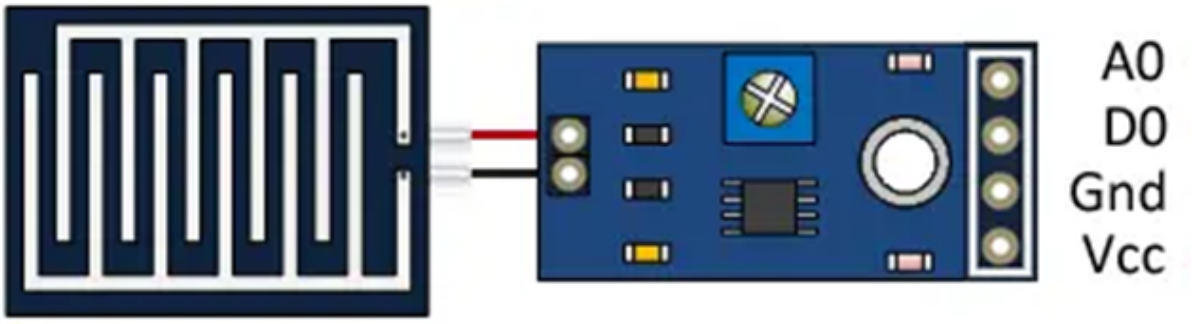
Para el desarrollo del proyecto, se seleccionaron sensores enfocados en medir tres variables clave: lluvia, flujo o caudal de agua y nivel de agua, fundamentales para la monitorización y alerta temprana en la estación IoT. Además, se incluyó una pantalla OLED, que permite visualizar los datos directamente en el lugar donde se ubique la estación, facilitando el monitoreo local de las mediciones en tiempo real.

4.2.4.2.1 Lluvia

- YL-83: Un módulo que detecta la presencia y cantidad de agua sobre su superficie gracias a una serie de pistas conductoras que generan un cambio de voltaje al entrar en contacto con el agua. Este sensor cuenta con un amplificador LM393 que permite entregar una salida analógica proporcional a la cantidad de agua, así como una salida digital ajustable mediante un potenciómetro. Funciona con un rango de voltaje de entrada de 3.3 a 5V.

Figura 25.

Diagrama del sensor YL-83.

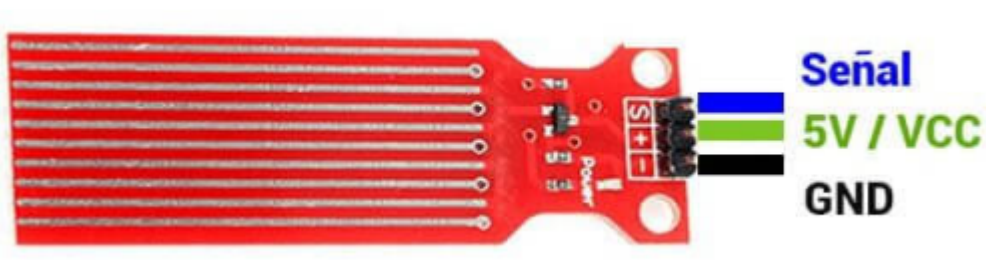


Nota. Tomado de Llamas, L. (2016, 13 de febrero). Detector de lluvia con Arduino y sensor FC-37 o YL-83. Luis Llamas. https://www.luisllamas.es/arduino-lluvia/

- Ard-365: diseñado específicamente para monitorear la presencia de agua en su superficie. Este sensor opera en un rango de voltaje de 3.3V a 5V, con un consumo de corriente máximo de 20 mA. Dispone de una placa de detección de 5x4 cm con pistas de cobre expuestas, que permiten identificar la presencia de agua mediante una salida digital ajustable y una salida analógica proporcional al nivel de humedad detectado. Con un costo aproximado de 6 dólares.

Figura 26.

Diagrama del sensor Ard-365.



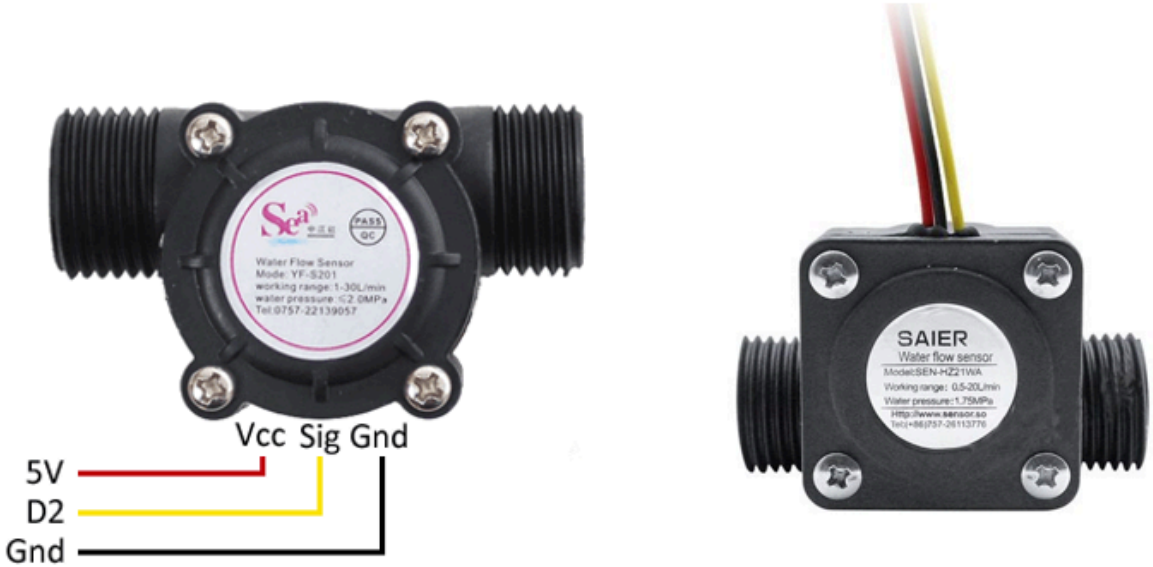
Nota. Tomado de Robot electrónica de Venezuela, automatización, Hobbies, y procesos industriales. (2021, 2 de agosto). Sensor lluvia control de nivel de agua para arduino - robot electrónica. Robot Electrónica. *https://robot.com.ve/product/sensor-lluvia-control-de-nivel-de-agua-para-arduino/*

4.2.4.2.2 Flujo

- Yf-s201 y EN-HZ21WA: Aunque ambos sensores operan en el mismo rango de flujo de 1 a 30 L/min, la principal diferencia radica en que el EN-HZ21WA soporta hasta 1.75 MPa de presión, lo que lo hace más robusto para entornos industriales, aunque su costo es significativamente más elevado (15-20 dólares frente a los 5-7 dólares del YF-S201). En términos de consumo, el YF-S201 tiene un consumo promedio de 10 mA a 5V, mientras que el EN-HZ21WA opera con una alimentación más versátil de 5 a 24V, ofreciendo mayor precisión en entornos de alta presión

Figura 27.

Diagrama del sensor Yf-s201 y EN-HZ21WA.



Nota. Imagen combinada y modificada a partir de:

1. Giraldo, S. A. C. (2023, 18 de septiembre). Sensor de Flujo YF-S201: Medición de Caudal con Arduino. Control Automático Educación. https://controlautomaticoeducacion.com/sistemas-embebidos/arduino/sensor-de-flujo-yf-s201-medicion-de-caudal-con-arduino/

2. Sourcewell Devices Pvt Ltd. (2022, 19 de septiembre). Saier SEN-HZ21WA Water Flow Sensor - SourceWell Devices. Sourcewell Devices. https://sourcewell.in/product/sen-hz21wa/

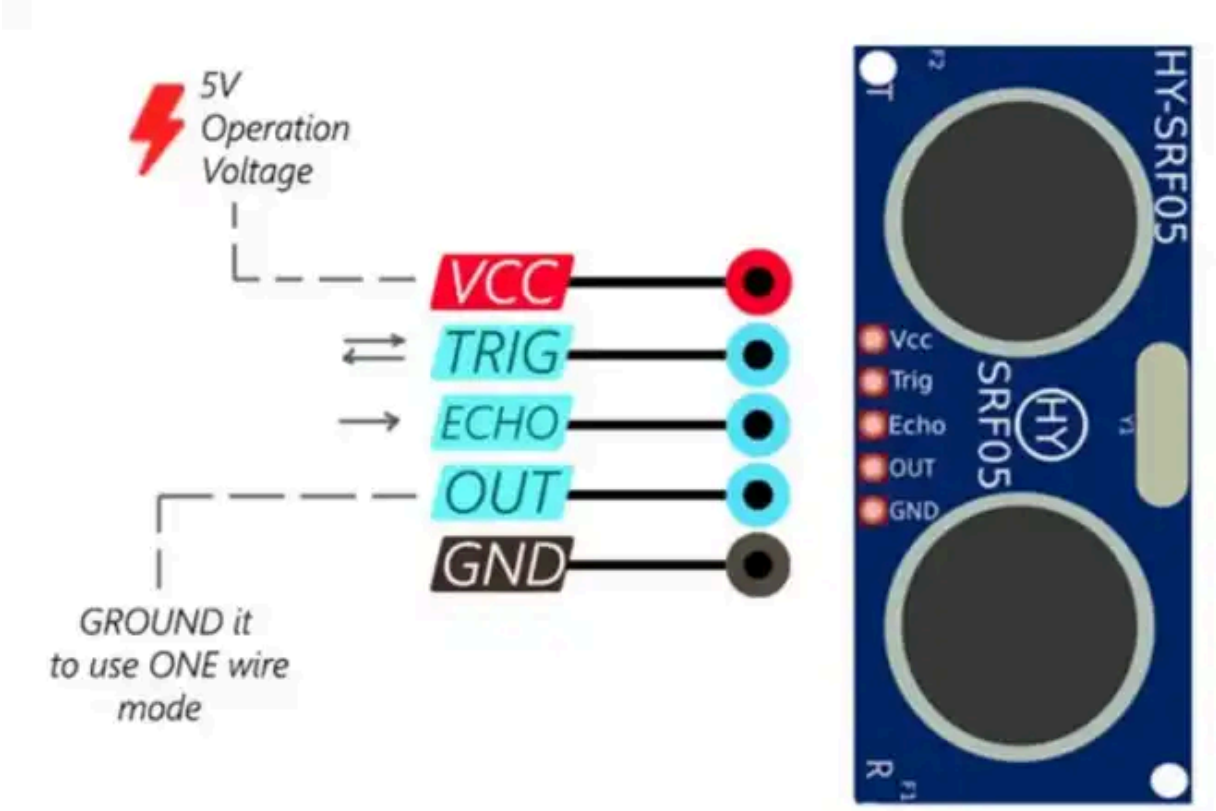
4.2.4.2.3 Nivel

- HC-SR04: Este sensor ultrasónico funciona emitiendo ondas ultrasónicas a 40 kHz a través de su transductor emisor, las cuales rebotan en un objeto y son captadas por el

transductor receptor. Permitiendo determinar distancias en un rango de 2 cm a 400 cm, con una precisión de ± 3 mm. Opera con un voltaje de 5V y un consumo promedio de corriente de 15 mA, con un costo aproximado de 4 dólares.

Figura 28.

Diagrama del sensor HC-SR04.



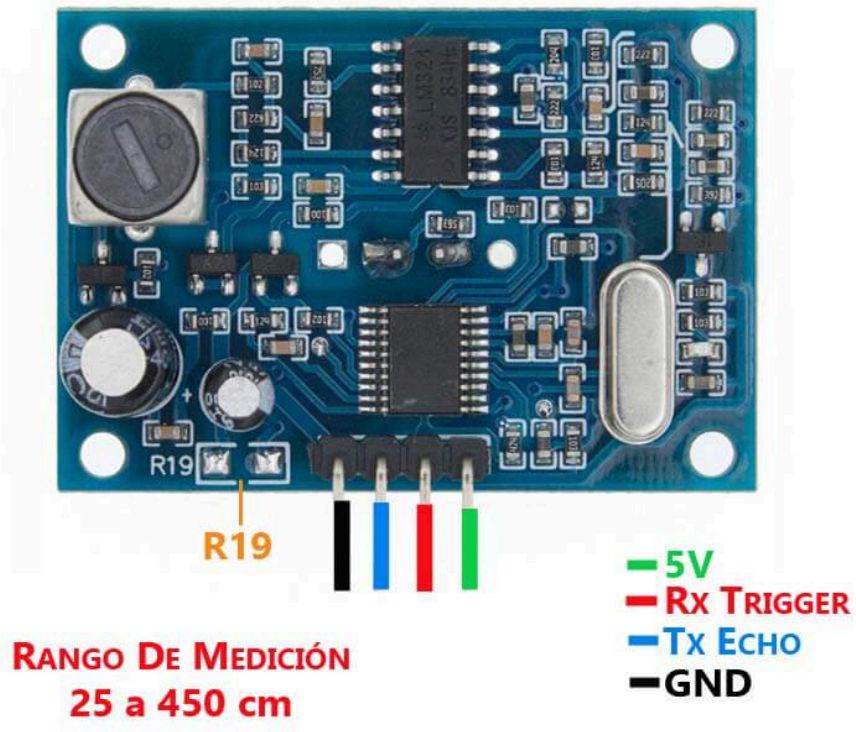
Nota. Tomado de Controla tu PC con el sensor ultrasónico HC-SR04 y Arduino. (2020, 13 de agosto). Descubrearduino.com. https://descubrearduino.com/controla-tu-pc-con-el-sensor-ultrasonico-hc-sr04-y-arduino/

- JSN-SR04T: Este sensor ultrasónico funciona emitiendo ondas ultrasónicas a 40 kHz a través de su transductor emisor impermeable.Permite determinar distancias en un

rango de 25 cm a 450 cm, con una precisión de ± 2 mm. Opera con un voltaje de 5V y un consumo promedio de corriente de 30 mA, con un costo aproximado de 10 dólares.

Figura 29.

Diagrama del sensor JSN-SR04T.



Nota. Nota. Tomado de Sensor Ultrasónico Contra El Agua JSN-SR04T (s.f.). UNIT Electronics. https://uelectronics.com/producto/sensor-ultrasonico-jns-sr04t/

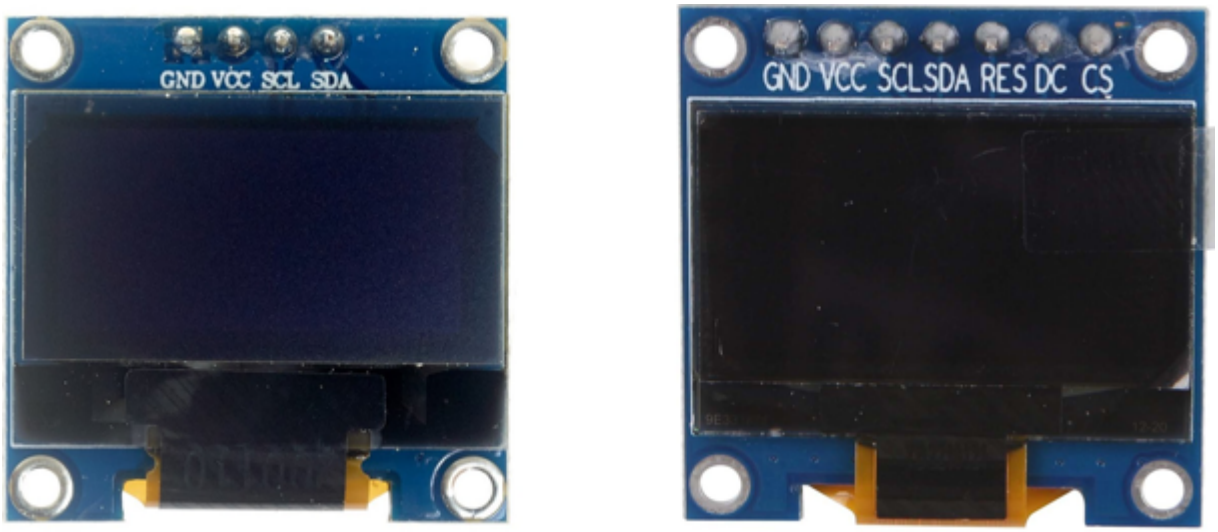
4.2.4.2.4 Pantalla OLED

- OLED SSD1306 (I2C) y OLED SSD1306 (SPI) :Aunque ambas pantallas OLED tienen un tamaño similar, la principal diferencia radica en su interfaz de comunicación. La primera pantalla utiliza el protocolo I2C, lo que requiere menos pines de conexión (VCC, GND, SCL, SDA) y la hace más sencilla de integrar; mientras que la segunda

emplea el protocolo SPI, que ofrece mayor velocidad de transmisión, pero a costa de utilizar más pines (VCC, GND, D0, D1, RES, DC, CS). Ambas pantallas operan con un voltaje de entrada de 3.3 a 5V, consumen aproximadamente 20 mA, y tienen un costo promedio en el mercado de 5 a 7 dólares.

Figura 30.

Diagrama de OLED SSD1306 (I2C) y OLED SSD1306 (SPI).



Nota. Imagen combinada y modificada a partir de:

1. Display Oled 0.96" I2C 128*64 SSD1306. (s.f). Naylamp Mechatronics - Perú. https://naylampmechatronics.com/oled/850-display-oled-096-i2c-12864-ssd1306.html,

2. Controla tu PC con el sensor ultrasónico HC-SR04 y Arduino. (2020, 13 de agosto). Descubrearduino.com. https://descubrearduino.com/controla-tu-pc-con-el-sensor-ultrasonico-hc-sr04-y-arduino/

4.3 Diseño de Hardware y Software

4.3.1 Definición de requerimientos funcionales, no funcionales y roles

4.3.1.1 Roles

Para el sistema, se han definido tres roles principales: Super Admin, Admin y Usuario.

Tabla 1.

Descripción de roles del sistema.

Rol	Descripción	Permisos
Super Admin	*Acceso total a todos los módulos y control sobre la plataforma.*	*Gestionar y añadir super administradores, acceder a todos los módulos y configuraciones del sistema.*
Admin	*Acceso a la mayoría de los módulos con restricciones en la gestión de usuarios. Puede modificar usuarios con su mismo rol o inferior.*	*Acceder al panel de administración de usuarios, gestionar usuarios con el mismo rol o inferior, pero no añadir/modificar super administradores ni acceder a la configuración global de usuarios.*
Usuario	*Acceso limitado al módulo de usuario, con permisos para ver y gestionar sus propios datos.*	*Ver estaciones meteorológicas visibles para su rol y gestionar sus propios datos. No tiene acceso a otros módulos del sistema.*

4.3.1.2 Requerimientos funcionales

4.3.1.2.1 Listado de los requerimientos funcionales

- **RF1:** El sistema debe ser capaz de recopilar información en tiempo real desde sensores conectados. Estos datos se procesarán para evaluar la situación del cauce y generar alertas tempranas cuando sea necesario.
- **RF2:** La plataforma debe permitir a los administradores visualizar en tiempo real las mediciones de las variables obtenidas desde los sensores IoT, así como consultar registros históricos.
- **RF3:** El sistema debe generar y enviar alertas automáticas en tiempo real a través de canales establecidos cuando se detecten valores anómalos en las mediciones.
- **RF4:** Proveer un panel accesible mediante autenticación segura, donde los administradores puedan gestionar sensores, revisar datos históricos, configurar umbrales de alertas y administrar permisos de usuarios.
- **RF5:** El sistema debe permitir el registro de usuarios, solicitando datos básicos como nombre, correo electrónico y contraseña.
- **RF6:** El sistema debe permitir a los usuarios autenticados suscribirse a una o más estaciones y visualizar dichas estaciones en el mapa.
- **RF7:** El sistema debe permitir a los administradores crear, actualizar y eliminar redes y estaciones.
- **RF8:** El sistema debe permitir a los super administradores crear, actualizar, leer y eliminar administradores y usuarios.

4.3.1.2.2 Especificación de los requerimientos funcionales

Tabla 2.

Requerimiento funcional 1.

ID	*RF1*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Sistema de monitoreo de variables hídricas*				
Complejidad	*Alta*		**Prioridad**	*5*	
Tipo	*Necesario*		**Requerimiento que lo utiliza o especializa**	*RF2, RF3*	
Crítico	*Sí, este requerimiento es fundamental para el funcionamiento del sistema, ya que proporciona los datos base necesarios para generar alertas y realizar el monitoreo.*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Todos los usuarios del sistema.*				
Entrada	*Datos captados por sensores IoT (nivel de agua, velocidad del caudal, lluvia).*		**Salida**	*Valores procesados y almacenados en la base de datos para su visualización y uso en otros subsistemas.*	
Descripción					
El sistema debe ser capaz de recopilar información en tiempo real desde sensores conectados. Estos datos se procesarán para evaluar la situación del cauce y generar alertas tempranas cuando sea necesario.					
Precondición					
1. *Los sensores deben estar conectados y configurados correctamente.* 2. *El sistema debe tener acceso a la red de comunicaciones.*					
Postcondición					
1. *Los datos serán enviados y almacenados en la plataforma de monitoreo.* 2. *Se activarán las alertas en caso de superar umbrales predefinidos.*					
Consideraciones					
Garantizar la precisión de las mediciones mediante calibración regular de los sensores.					
Criterios de aceptación					
El sistema debe recopilar datos en tiempo real de al menos tres variables críticas. Los valores procesados deben estar disponibles para visualización en la plataforma web en menos de 500 segundos.					

Tabla 3.

Requerimiento funcional 2.

ID	*RF2*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Plataforma web para visualización de datos*				
Complejidad	*Alta*		**Prioridad**	*4*	
Tipo	*Necesario*		**Requerimiento que lo utiliza o especializa**	*RF3*	
Crítico	*Sí, este requerimiento es esencial para que los usuarios puedan monitorear las variables recolectadas por los sensores y tomar decisiones en tiempo real.*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Administradores del sistema.*				
Entrada	*Datos procesados de los sensores en la base de datos.*		**Salida**	*Interfaz gráfica con visualización de datos.*	
Descripción					
La plataforma debe permitir a los administradores visualizar en tiempo real las mediciones de las variables obtenidas desde los sensores IoT, así como consultar registros históricos.					
Precondición					
1. *Los sensores deben estar conectados y configurados correctamente.* 2. *El sistema debe tener acceso a la red de comunicaciones.*					
Postcondición					
1. *Los usuarios pueden visualizar las variables en tiempo real mediante la interfaz web.* 2. *Los registros históricos están disponibles para consultas adicionales.*					
Consideraciones					
Garantizar que la interfaz sea intuitiva y responsiva para diferentes dispositivos.					
Criterios de aceptación					
Los usuarios deben poder visualizar todas las variables registradas, los tiempos de carga no deben superar los 500 segundos en condiciones normales de uso y la interfaz debe ser compatible con navegadores modernos y dispositivos móviles.					

Tabla 4.

Requerimiento funcional 3.

ID	*RF3*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Sistema de alerta temprana automatizado*				
Complejidad	*Alta*		**Prioridad**	*5*	
Tipo	*Necesario*		**Requerimiento que lo utiliza o especializa**	*N/A*	
Crítico	*Si, este requerimiento es vital para informar a los usuarios de eventos críticos en tiempo real, garantizando la seguridad y la toma de decisiones rápidas.*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Usuarios del sistema.*				
Entrada	*Valores procesados y almacenados que excedan los umbrales predefinidos.*		**Salida**	*Notificaciones automáticas enviadas a los usuarios registrados.*	
Descripción					
El sistema debe generar y enviar alertas automáticas en tiempo real a través de canales establecidos cuando se detecten valores anómalos en las mediciones.					
Precondición					
1. *Los umbrales de alerta deben estar configurados en el sistema.* 2. *Los usuarios deben estar registrados y vinculados a los canales de comunicación.* 3. *Los sensores deben estar operativos y enviando datos.*					
Postcondición					
Los usuarios reciben notificaciones automáticas con información clara y precisa.					
Consideraciones					
1. *Garantizar que las notificaciones sean enviadas solo cuando los umbrales sean excedidos, evitando falsas alarmas.* 2. *Asegurar que las alertas incluyen detalles suficientes para que los usuarios entiendan la situación y puedan actuar.*					
Criterios de aceptación					
1. *Las alertas deben ser generadas y enviadas automáticamente en menos de 500 segundos tras la detección de un evento crítico.* 2. *Los usuarios deben recibir alertas claras y completas, incluyendo recomendaciones de acción.*					

Tabla 5.

Requerimiento funcional 4.

ID	*RF4*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Panel de administración seguro*				
Complejidad	*Media*		**Prioridad**	*5*	
Tipo	*Necesario*		**Requerimiento que lo utiliza o especializa**	*N/A*	
Crítico	*Si, este requerimiento es clave para garantizar que solo administradores puedan acceder a la gestión del sistema y la configuración de sensores y alertas.*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Administradores del sistema.*				
Entrada	*Credenciales de acceso de administradores.*		**Salida**	*Acceso al panel de administración y opciones de configuración.*	
Descripción					
Proveer un panel accesible mediante autenticación segura, donde los administradores puedan gestionar sensores, revisar datos históricos, configurar umbrales de alertas y administrar permisos de usuarios.					
Precondición					
1. *Los administradores deben estar registrados en el sistema con permisos válidos.* 2. *El sistema debe contar con conexión estable al servidor de autenticación.*					
Postcondición					
El administrador accede al panel de administración y puede realizar configuraciones.					
Consideraciones					
Asegurar una experiencia de usuario intuitiva para facilitar la gestión del sistema.					
Criterios de aceptación					
1. *Los administradores deben poder iniciar sesión y acceder al panel de manera segura.* 2. *El sistema debe prevenir accesos no autorizados mediante mecanismos de seguridad implementados.*					

Tabla 6.

Requerimiento funcional 5.

ID	*RF5*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Registro de usuarios*				
Complejidad	*Media*		**Prioridad**	*4*	
Tipo	*Necesario*		**Requerimiento que lo utiliza o especializa**	*RF6*	
Crítico	*Si*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Todos los usuarios del sistema.*				
Entrada	*Datos básicos del usuario*		**Salida**	*Usuario registrado*	
Descripción					
El sistema debe permitir el registro de usuarios, solicitando datos básicos como nombre, correo electrónico y contraseña.					
Precondición					
1. *El usuario no debe estar registrado previamente en el sistema.* 2. *El formulario de registro debe estar disponible.*					
Postcondición					
1. *El usuario queda registrado en el sistema y puede autenticarse posteriormente.* 2. *Los datos del usuario se almacenan en la base de datos.*					
Consideraciones					
1. *Validar que el correo electrónico proporcionado sea único en el sistema.* 2. *Implementar validaciones en el formulario para evitar datos erróneos o incompletos.* 3. *Utilizar un mecanismo de encriptación para almacenar la contraseña de forma segura.*					
Criterios de aceptación					
1. *El sistema debe validar que el correo electrónico no esté registrado previamente.* 2. *El sistema debe permitir el registro sólo si todos los campos obligatorios están completos y validados.* 3. *La contraseña debe almacenarse de manera segura utilizando un estándar de encriptación reconocido.*					

Tabla 7.

Requerimiento funcional 6.

ID	*RF6*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Suscripción y visualización de estaciones*				
Complejidad	*Alta*		**Prioridad**	*5*	
Tipo	*Necesario*		**Requerimiento que lo utiliza o especializa**	*RF7*	
Crítico	*Si*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Usuarios autenticados del sistema*				
Entrada	*Datos de usuario autenticado*	**Salida**	*Mapa con estaciones suscritas*		
Descripción					
El sistema debe permitir a los usuarios autenticados suscribirse a una o más estaciones y visualizar dichas estaciones en el mapa.					
Precondición					
1. *El usuario debe estar autenticado en el sistema.* 2. *Las estaciones disponibles deben estar registradas y configuradas en el sistema.*					
Postcondición					
1. *El usuario queda suscrito a las estaciones seleccionadas.* 2. *El mapa del usuario muestra las estaciones a las que está suscrito.*					
Consideraciones					
1. *Garantizar la visualización adecuada de las estaciones en el mapa, incluyendo su información básica.* 2. *Asegurar que el usuario pueda gestionar (agregar/eliminar) sus suscripciones a estaciones en cualquier momento.*					
Criterios de aceptación					
1. *El sistema debe permitir que los usuarios seleccionen la suscripción a estaciones.* 2. *Las estaciones suscritas deben ser visibles en el mapa del usuario.* 3. *El sistema debe permitir la gestión de suscripciones de manera sencilla e intuitiva.*					

Tabla 8.

Requerimiento funcional 7.

ID	*RF7*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Gestión de redes y estaciones*				
Complejidad	*Alta*		**Prioridad**	*4*	
Tipo	*Necesario*		**Requerimiento que lo utiliza o especializa**	*N/A*	
Crítico	*Si*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Administradores*				
Entrada	*Datos de redes y estaciones*		**Salida**	*Estaciones gestionadas*	
Descripción					
El sistema debe permitir a los administradores crear, leer, actualizar y eliminar redes y estaciones.					
Precondición					
1. *El administrador debe estar autenticado en el sistema.* 2. *Las redes y estaciones existentes deben estar configuradas correctamente en la base de datos.*					
Postcondición					
1. *Las redes y estaciones creadas, actualizadas o eliminadas son registradas en el sistema.* 2. *La información de los sensores asociados es actualizada en tiempo real.*					
Consideraciones					
1. *Validar que los datos ingresados por el administrador sean completos y correctos.* 2. *Garantizar que las acciones realizadas no afecten el funcionamiento de otras estaciones.*					
Criterios de aceptación					
1. *El sistema debe permitir al administrador gestionar redes y estaciones (crear, leer, actualizar, eliminar) desde el panel de administración.* 2. *Los cambios realizados deben reflejarse inmediatamente en las visualizaciones de los usuarios.*					

Tabla 9.

Requerimiento funcional 8.

ID	*RF8*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Gestión de administradores*				
Complejidad	*Alta*		**Prioridad**	*5*	
Tipo	*Necesario*		**Requerimiento que lo utiliza o especializa**	*N/A*	
Crítico	*Si*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Super administradores*				
Entrada	*Datos de los administradores*		**Salida**	*Administradores gestionados*	
Descripción					

El sistema debe permitir a los super administradores crear, actualizar, leer y eliminar administradores y usuarios.

Precondición
1. *El super administrador debe estar autenticado en el sistema.* 2. *Los administradores y usuarios a gestionar deben estar registrados o ser susceptibles de registro.*
Postcondición
1. *Los cambios realizados sobre los administradores y usuarios (creación, actualización, eliminación) se registran en el sistema.*
Consideraciones
1. *Validar que los datos ingresados para los administradores y usuarios sean completos y correctos.*
Criterios de aceptación
1. *El sistema debe permitir a los super administradores realizar operaciones CRUD (crear, leer, actualizar, eliminar) sobre los administradores y usuarios.* 2. *Los permisos asignados a los administradores deben ser aplicados de manera inmediata.* 3. *Los cambios realizados deben reflejarse sin afectar la funcionalidad del sistema.*

4.3.1.3 Requerimientos no funcionales

4.3.1.3.1 Listado de los requerimientos no funcionales

- **RNF 1:** El software debe ser modular y documentado para facilitar la incorporación de nuevas funcionalidades y correcciones.
- **RNF 2:** La infraestructura debe ser escalable para manejar un incremento en el número de sensores y usuarios sin afectar el rendimiento.
- **RNF 3:** La interfaz del sistema debe ser intuitiva, fácil de usar y capaz de procesar las solicitudes del usuario con tiempos de respuesta menores a 60 segundos para garantizar una experiencia fluida.
- **RNF 4:** El sistema debe almacenar de manera segura los datos recolectados en una infraestructura de nube, garantizando la accesibilidad para los subsistemas, reportes y análisis histórico.

4.3.1.3.2 Especificación de los requerimientos no funcionales

Tabla 10.

Requerimiento no funcional 1.

ID	*RNF 1*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Facilidad de mantenimiento y actualización*				
Complejidad	*Media*		**Prioridad**	*4*	
Tipo	*Deseable*		**Requerimiento que lo utiliza o especializa**	*N/A*	
Crítico	*Si, este requerimiento es fundamental para asegurar la continuidad y escalabilidad del sistema a lo largo del tiempo.*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Desarrolladores del sistema.*				
Entrada	*Cambios solicitados por los desarrolladores o usuarios.*		**Salida**	*Sistema actualizado y documentado.*	
Descripción					
El sistema debe contar con una arquitectura modular y documentación detallada que permita a los desarrolladores realizar cambios, implementar nuevas funcionalidades o corregir errores de forma eficiente.					
Precondición					
1. *Existencia de un sistema de versionado para el control de cambios.* 2. *Acceso a la documentación actualizada del sistema.*					
Postcondición					
1. *Los cambios solicitados han sido implementados correctamente.* 2. *La documentación ha sido actualizada con los cambios realizados.*					
Consideraciones					
1. *Asegurar que la documentación sea clara, precisa y accesible para todos los desarrolladores.* 2. *Implementar un sistema de retroalimentación para mejorar continuamente el proceso de mantenimiento.*					
Criterios de aceptación					
1. *Los desarrolladores deben poder realizar cambios en el sistema sin introducir nuevos errores.* 2. *La documentación debe estar actualizada y reflejar los cambios realizados.* 3. *El sistema debe mantener su funcionalidad principal tras cualquier cambio implementado.*					

Tabla 11.

Requerimiento no funcional 2.

ID	*RNF 2*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Escalabilidad automática.*				
Complejidad	*Alta*		**Prioridad**	*3*	
Tipo	*Deseable*		**Requerimiento que lo utiliza o especializa**	*N/A*	
Crítico	*Si, este requerimiento es esencial para manejar el crecimiento en el volumen de datos y usuarios sin afectar el rendimiento del sistema.*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Desarrolladores del sistema.*				
Entrada	*Cambios en la carga de trabajo o en el número de usuarios.*		**Salida**	*Sistema ajustado para mantener el rendimiento óptimo.*	
Descripción					
El sistema debe ser capaz de adaptarse a incrementos o decrementos en la demanda de recursos, garantizando un rendimiento constante y eficiente.					
Precondición					
1. *Infraestructura configurada para soportar escalabilidad.* 2. *Implementación de herramientas de monitoreo de carga y rendimiento.*					
Postcondición					
1. *El sistema ajusta los recursos necesarios para mantener el rendimiento.* 2. *No se producen interrupciones ni degradaciones significativas en la experiencia del usuario.*					
Consideraciones					
Configurar límites de escalabilidad para evitar costos excesivos o uso ineficiente de recursos.					
Criterios de aceptación					
El rendimiento del sistema no debe degradarse durante los picos de demanda.					

Tabla 12.

Requerimiento no funcional 3.

ID	*RNF 3*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Interfaz y tiempo de respuesta*				
Complejidad	*Media*		**Prioridad**	*4*	
Tipo	*Necesario*		**Requerimiento que lo utiliza o especializa**	*N/A*	
Crítico	*Si, este requerimiento es esencial para garantizar una experiencia de usuario eficiente, evitando tiempos de espera prolongados que puedan desincentivar el uso del sistema.*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Todos los usuarios del sistema.*				
Entrada	*Interacción del usuario con la interfaz web.*		**Salida**	*Respuesta inmediata a las solicitudes realizadas por el usuario.*	
Descripción					
La interfaz del sistema debe ser intuitiva, fácil de usar y capaz de procesar las solicitudes del usuario con tiempos de respuesta menores a 60 segundos para garantizar una experiencia fluida.					
Precondición					
1. *El sistema debe estar operativo y accesible.* 2. *Los datos necesarios para las consultas deben estar previamente almacenados en la base de datos.*					
Postcondición					
1. *Los usuarios interactúan con la interfaz sin interrupciones ni retrasos significativos.* 2. *Las solicitudes son procesadas y mostradas en tiempo real.*					
Consideraciones					
1. *Optimizar consultas y procesos para minimizar tiempos de respuesta.* 2. *Implementar mensajes claros para los usuarios en caso de errores o tiempos de espera prolongados.*					
Criterios de aceptación					
1. *El sistema debe procesar y responder a las solicitudes del usuario en menos de 60 segundos en condiciones normales de operación.* 2. *La interfaz debe ser compatible con dispositivos y navegadores modernos.*					

Tabla 13.

Requerimiento no funcional 4.

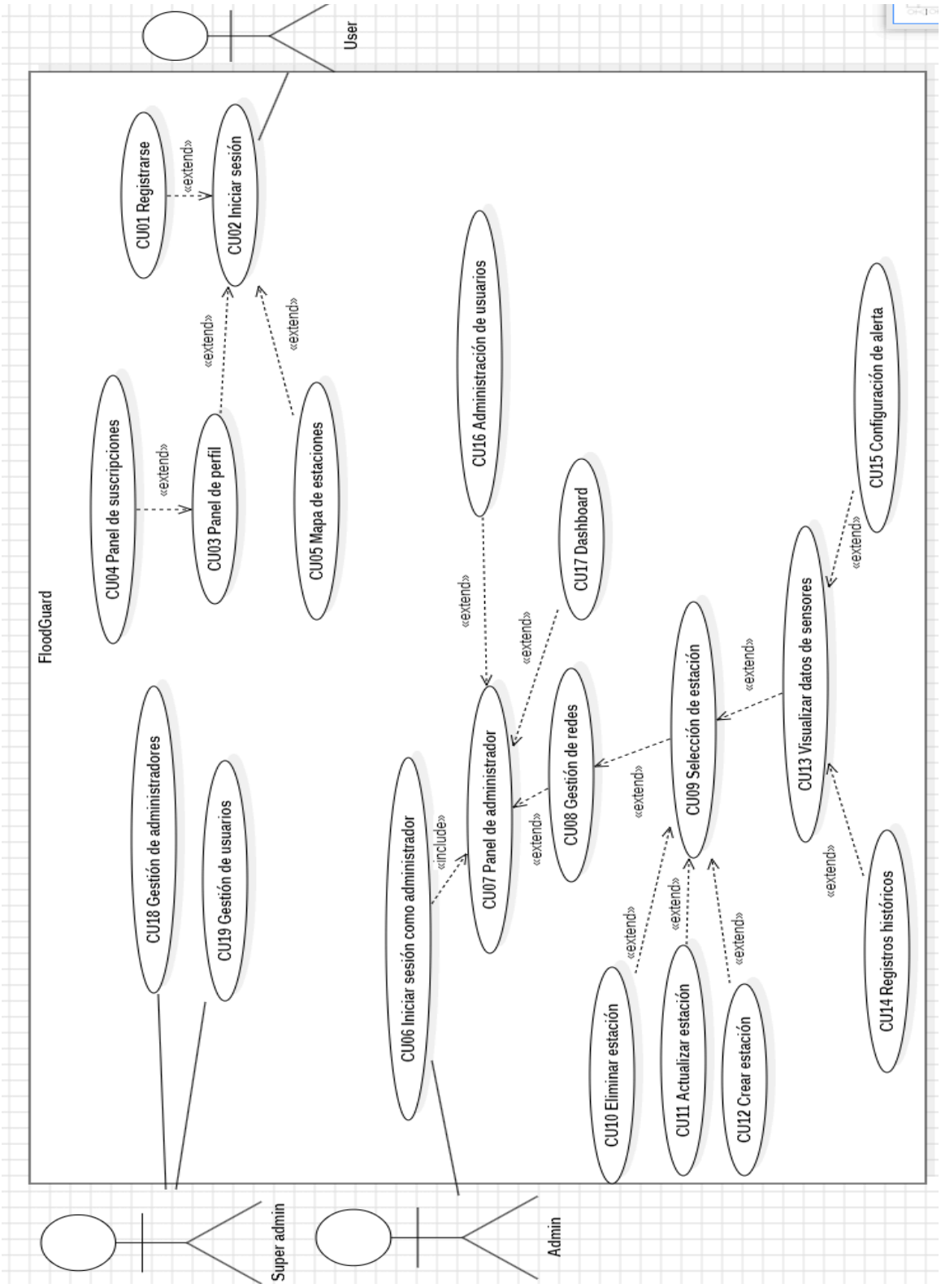
ID	*RNF 4*	**Fuente**	*Equipo de desarrollo*		
Nombre	*Sistema de almacenamiento en la nube*				
Complejidad	*Alta*		**Prioridad**	*4*	
Tipo	*Necesario*		**Requerimiento que lo utiliza o especializa**	*N/A*	
Crítico	*Si, este requerimiento es esencial para garantizar la disponibilidad y seguridad de los datos recolectados por el sistema.*				
Documentos de visualización asociados	*No se requiere*				
Usuarios	*Todos los usuario del sistema*				
Entrada	*Datos procesados desde RF1 y generados por los sensores IoT.*		**Salida**	*Datos almacenados en la nube disponibles para consulta y análisis.*	
Descripción					
El sistema debe almacenar de manera segura los datos recolectados en una infraestructura de nube, garantizando la accesibilidad para los subsistemas, reportes y análisis histórico.					
Precondición					
1. *Configuración de la infraestructura en la nube.* 2. *Conexión activa entre los sensores, el sistema backend y la nube.*					
Postcondición					
1. *Los datos recolectados están almacenados de forma segura en la nube.* 2. *Los datos almacenados son accesibles para su visualización y análisis posterior.*					
Consideraciones					
N/A					
Criterios de aceptación					
Los datos deben estar accesibles desde la plataforma web en menos de 500 segundos.					

4.3.2 Casos de uso

Con el propósito de analizar cómo interactúan los usuarios con el sistema, se construyó un diagrama de casos de uso. Este se basó en los roles previamente establecidos, permitiendo identificar las operaciones que podrán llevar a cabo dentro de la plataforma.

Figura 31.

Diagrama de casos de uso.



4.3.3 Diagrama de procesos

Figura 32.

Diagrama de proceso de envío de datos desde la estación.

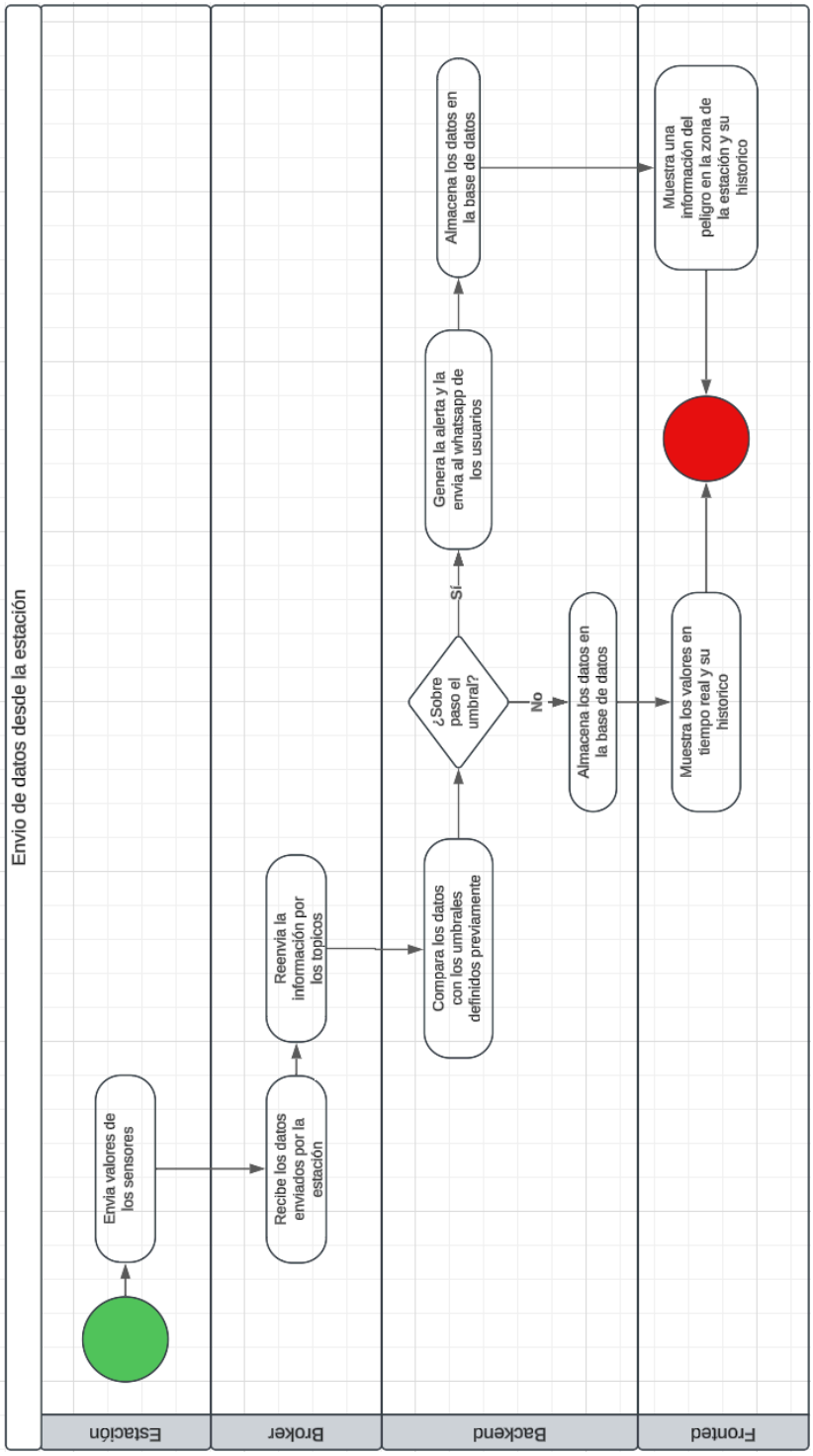


Figura 33.

Diagrama de proceso de suscripción a estación por parte de los usuarios.

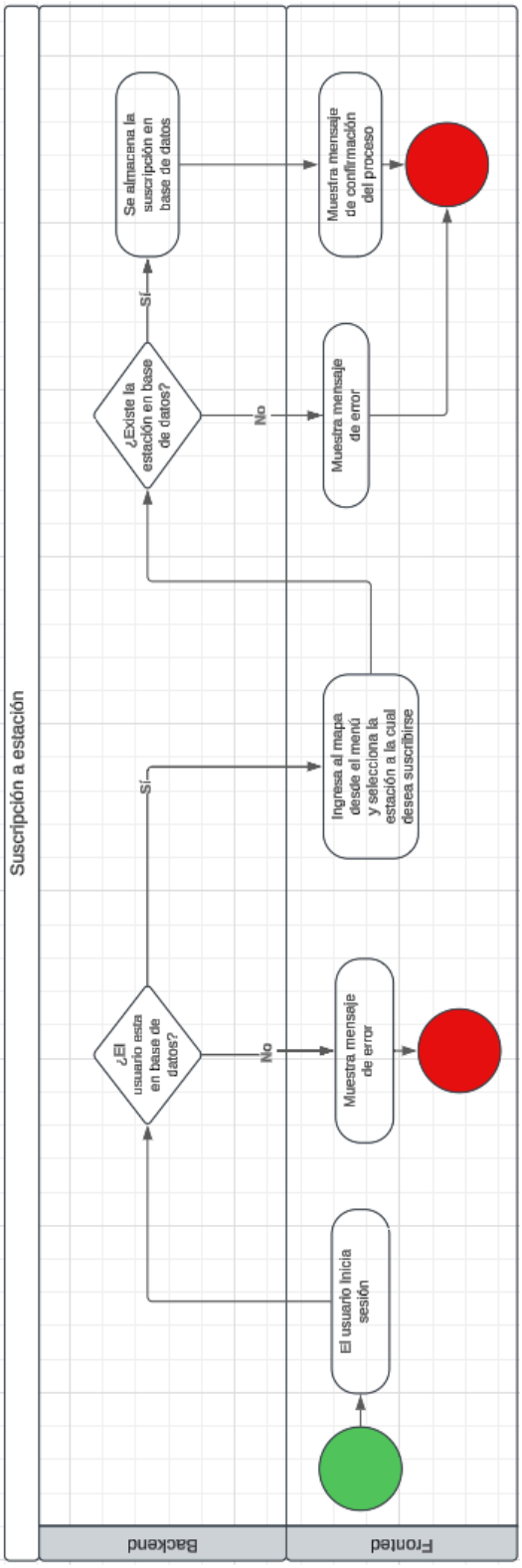


Figura 34.

Diagrama de proceso de eliminación de estación.

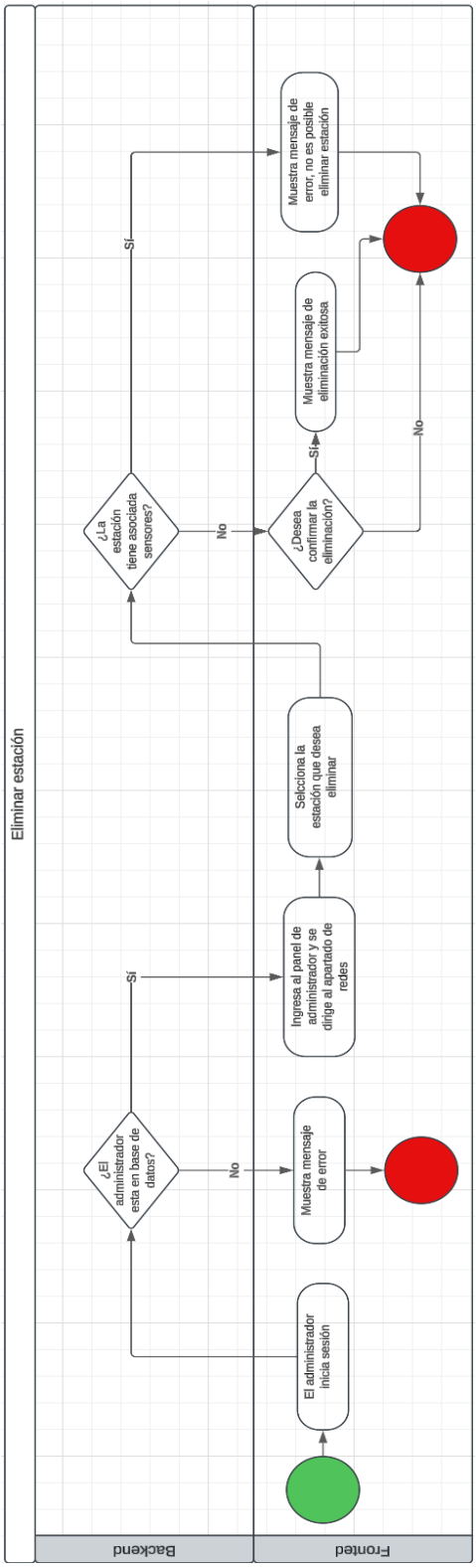
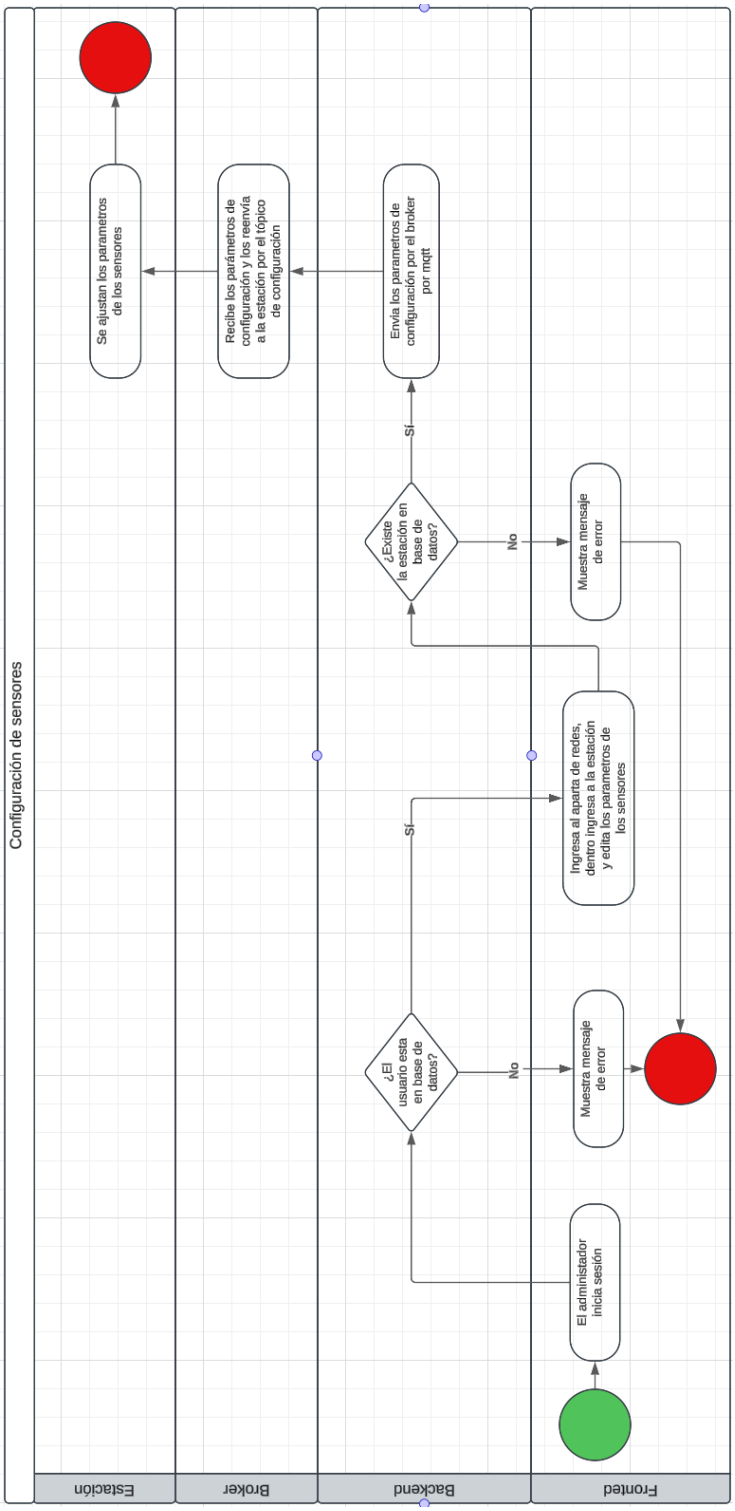


Figura 35.

Diagrama de proceso de configuración de estación.



4.3.4 Modelo de datos

A continuación se presenta el modelo de datos utilizado para la implementación de la base de datos en el sistema. La base de datos fue diseñada utilizando MongoDB.

Para la representación gráfica del modelo, se utilizó la herramienta DBSchema, la cual permite visualizar las colecciones, relaciones y estructuras de los datos de manera clara y organizada. Este diagrama proporciona una visión general de las colecciones, sus atributos principales, y las relaciones entre ellas, facilitando la comprensión del diseño y la lógica de la base de datos.

Figura 36.

Diagrama de base de datos.

4.3.5 Diseño del mockup

Figura 37.

Vista de inicio de sesión.

Figura 38.

Vista de registro.

Figura 39.

Vista de dashboard.

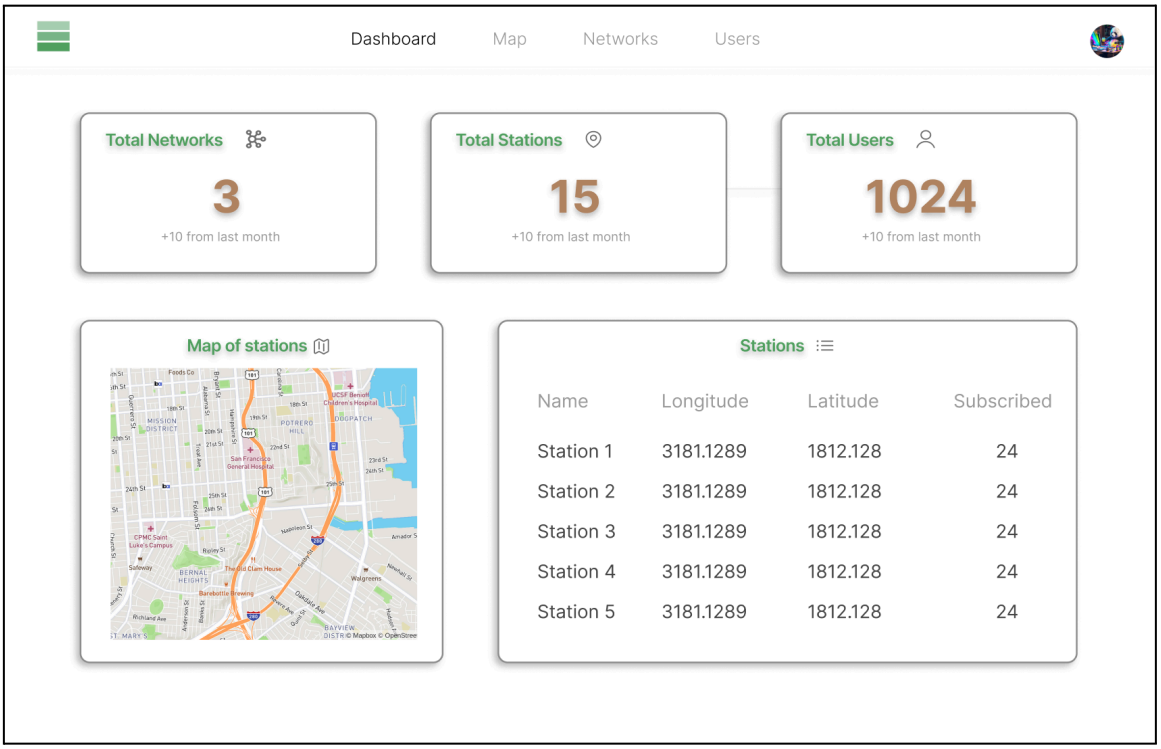


Figura 40.

Vista de ubicación de Redes y Estaciones.

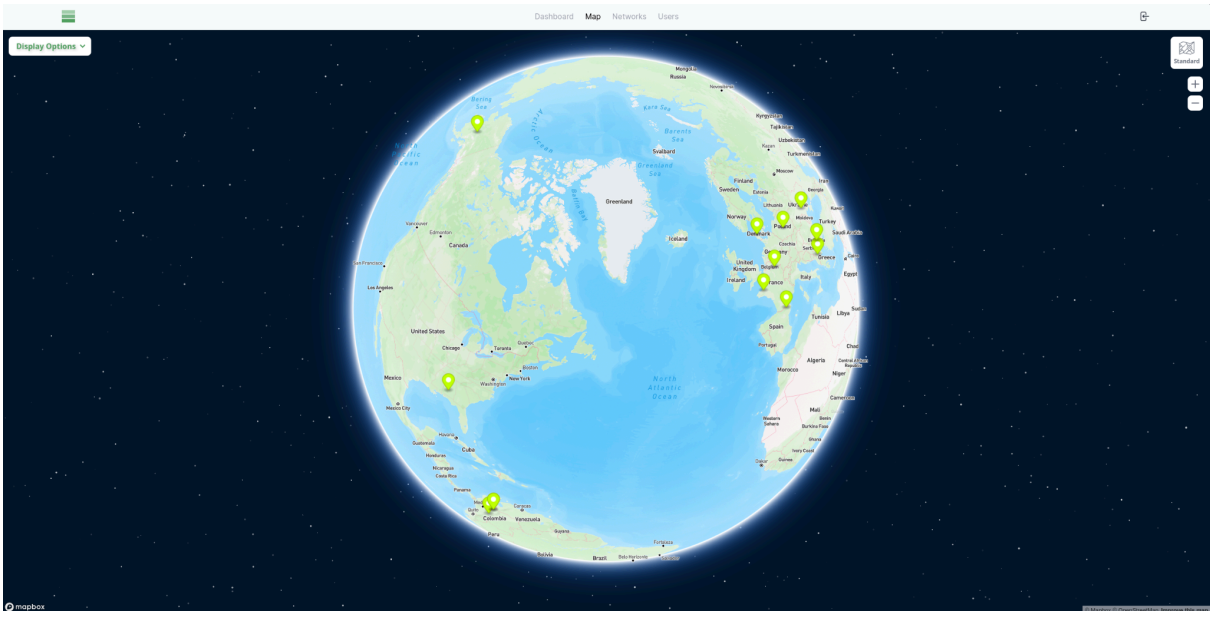


Figura 41.

Vista de información de Redes.

Figura 42.

Vista de información de Estaciones.

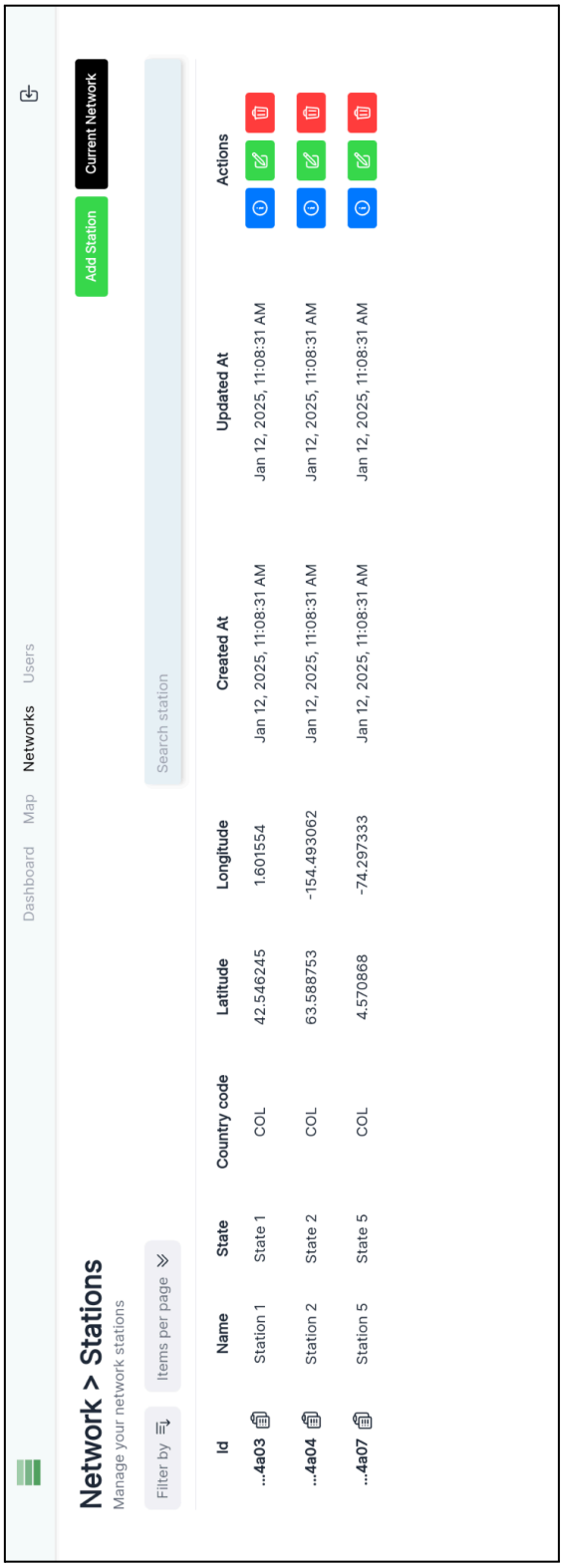


Figura 43.

Vista de información de la Estación.

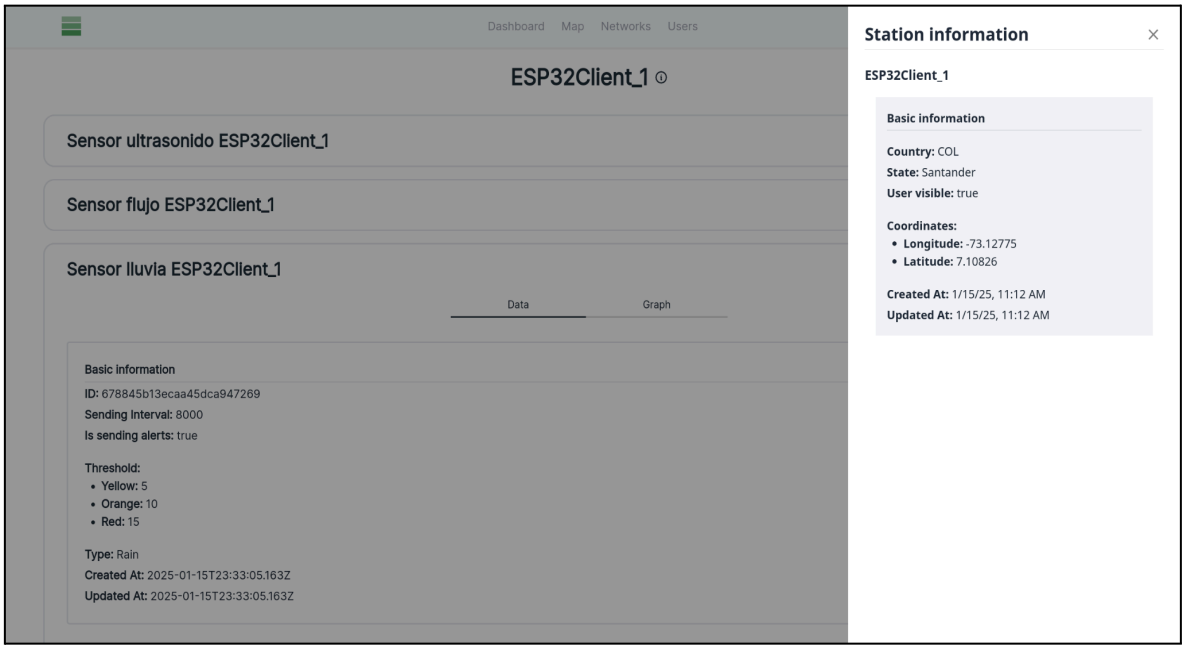


Figura 44.

Vista de sensores de la Estación.

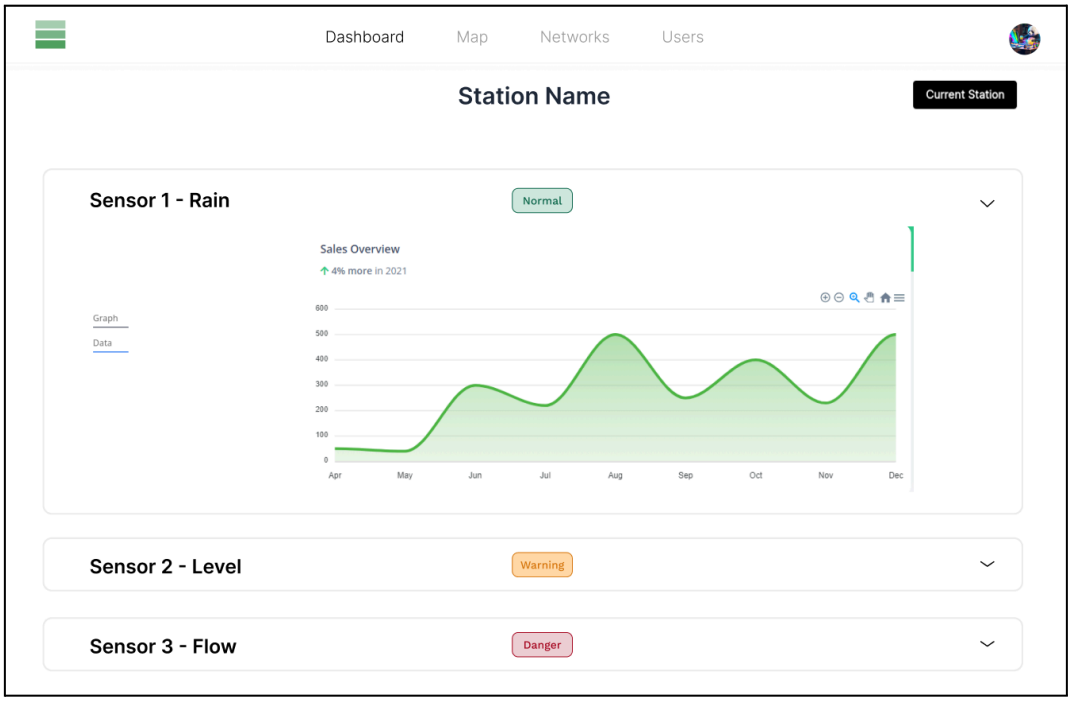


Figura 45.

Vista de administración usuarios.

Figura 46.

Vista de usuario.

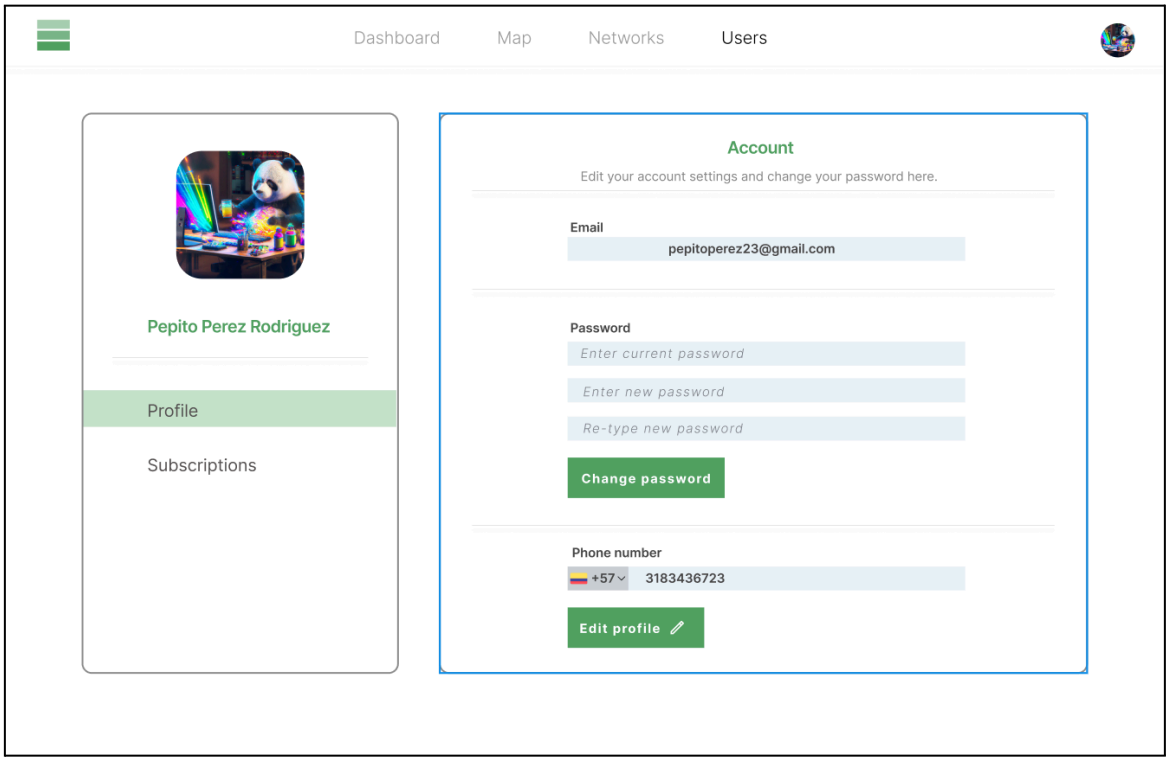
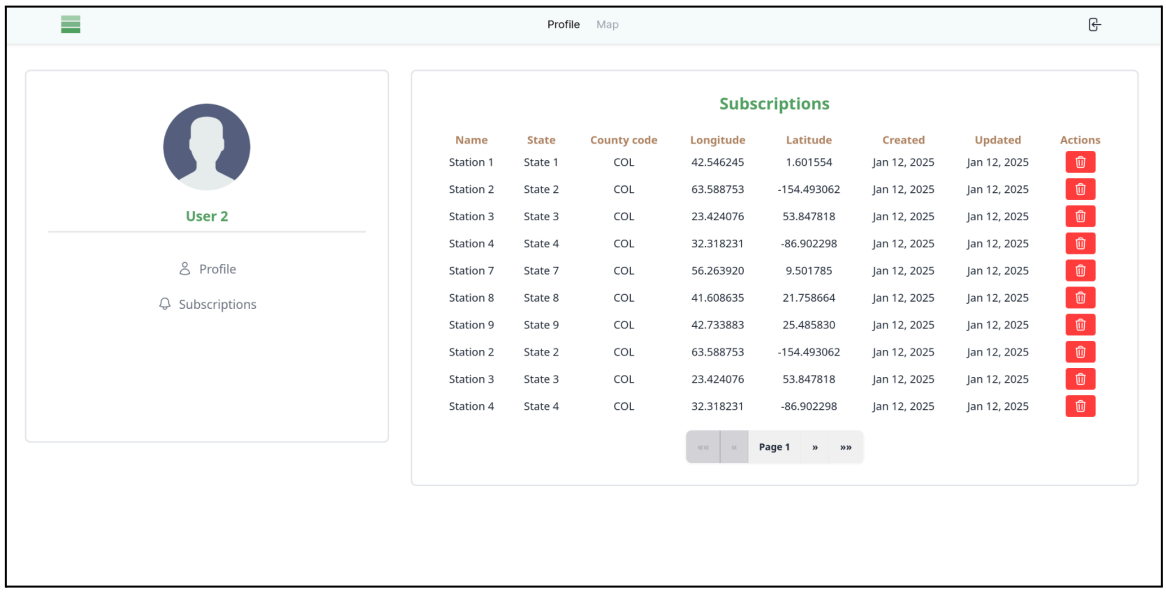


Figura 47.

Vista de subscripciones.



4.3.6 Diseño de la arquitectura

La arquitectura del sistema se basa en un enfoque de microservicios que integra hardware, backend, frontend y servicios externos, diseñada para garantizar un funcionamiento eficiente, escalabilidad y un procesamiento en tiempo real de los datos. A continuación, se describe el diagrama presentado y los componentes principales que lo conforman.

El sistema se compone de varios elementos interconectados. En primer lugar, los clientes, que incluyen tanto usuarios como administradores, interactúan con el sistema a través de un navegador web. Desde esta interfaz, pueden visualizar datos, gestionar estaciones y recibir notificaciones. Esta interacción es gestionada principalmente por dos servidores: el backend, encargado de la lógica central, y el frontend, que proporciona una experiencia de usuario dinámica y accesible.

El backend, desplegado en Railway y desarrollado con Node.js, implementa una arquitectura en capas que asegura modularidad y claridad en su funcionamiento. Los controladores reciben las solicitudes HTTP y delegan la lógica al nivel de servicios, donde se procesan las operaciones más complejas. Este nivel interactúa con la base de datos, que utiliza MongoDB para almacenar datos estructurados como información sobre estaciones, sensores y usuarios. Para mantener la consistencia y claridad en el intercambio de información, se emplean objetos de transferencia de datos (DTO), mientras que las rutas definen las URLs del sistema y su asociación con los controladores mediante el framework Express.

La base de datos MongoDB, esencial para el sistema, garantiza el almacenamiento de información crítica, como los datos provenientes de sensores y configuraciones de usuarios.

La comunicación entre el backend y la base de datos se realiza mediante el protocolo TCP/IP, permitiendo un flujo eficiente de datos.

En cuanto a la comunicación con el hardware, el sistema utiliza el broker EMQX, que gestiona el protocolo MQTT para conectar las estaciones basadas en microcontroladores ESP32 con el backend. Este componente no solo facilita la recepción de datos de los sensores, sino que también permite enviar comandos y actualizaciones a los dispositivos, haciendo posible una integración robusta con el ecosistema IoT.

El frontend, desplegado en Vercel y desarrollado con Angular, actúa como puente visual y funcional entre los usuarios y el sistema. Este servidor organiza la lógica de presentación mediante componentes reutilizables, los cuales se combinan con servicios que gestionan las interacciones con la API REST del backend. Además, las directivas y templates personalizados aseguran una experiencia de usuario altamente interactiva, mientras que el inyector configura las dependencias necesarias para mantener la coherencia en toda la aplicación.

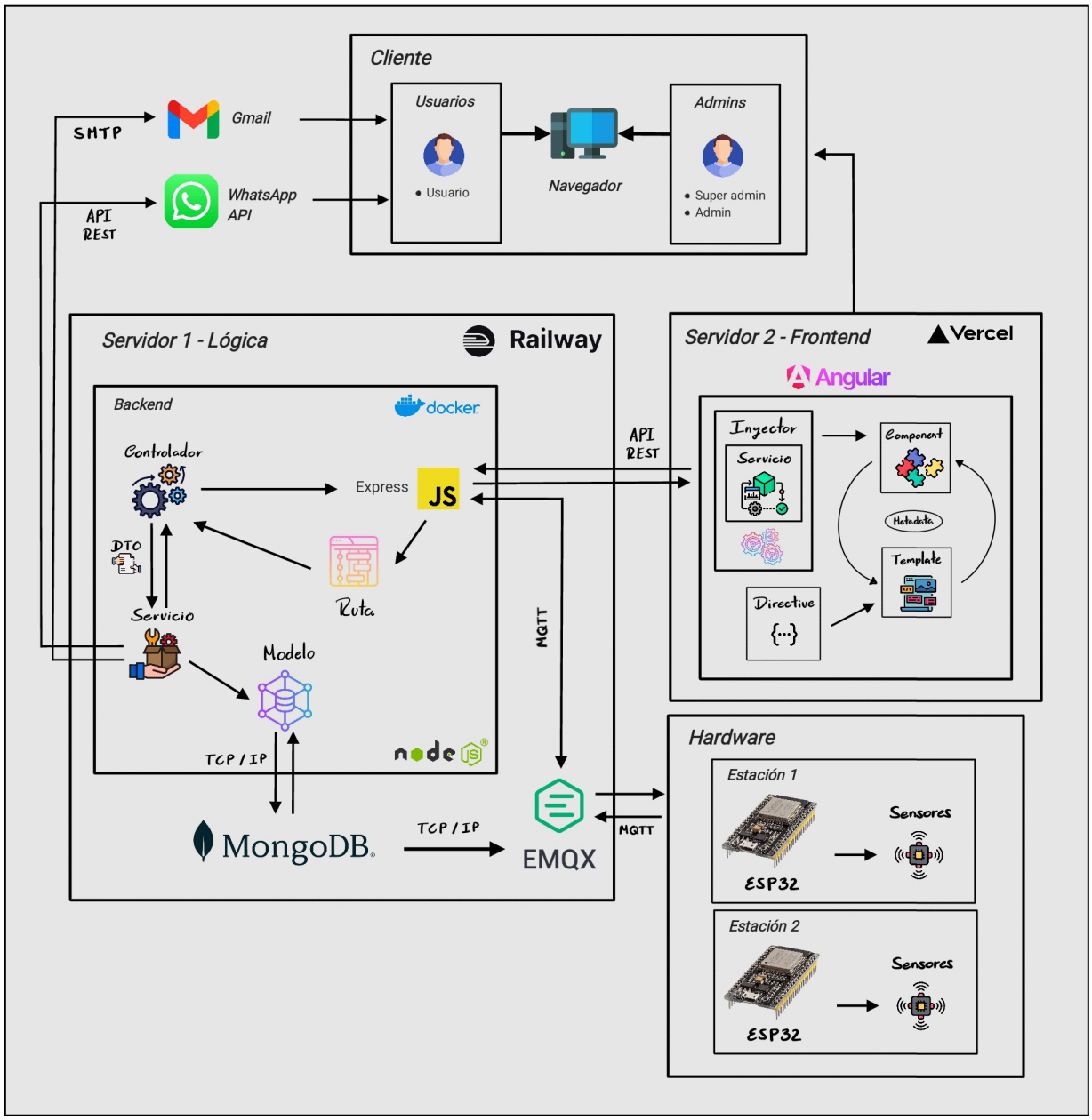
El hardware del sistema incluye estaciones basadas en microcontroladores ESP32, equipados con sensores como ultrasonido, flujo de agua y detección de lluvia. Estas estaciones recopilan datos en tiempo real y los transmiten al broker EMQX mediante MQTT. Una vez procesados, estos datos son almacenados en MongoDB y accesibles a través del frontend mediante consultas directas.

Por último, el sistema integra servicios externos para enviar notificaciones de alerta de manera efectiva. Tanto Gmail como la API de WhatsApp se utilizan para este propósito: Gmail permite enviar notificaciones por correo electrónico sobre alertas relacionadas con las

estaciones, mientras que la API de WhatsApp ofrece la posibilidad de enviar mensajes instantáneos, asegurando que los usuarios reciban información crítica en tiempo real. Estos servicios garantizan una comunicación oportuna y confiable, mejorando significativamente la experiencia del usuario.

Figura 48.

Diagrama arquitectura del sistema.



4.3.7 Control de versiones

El control de versiones fue implementado para gestionar de manera eficiente los cambios realizados en el código fuente durante el desarrollo del sistema. Para ello, se utilizó

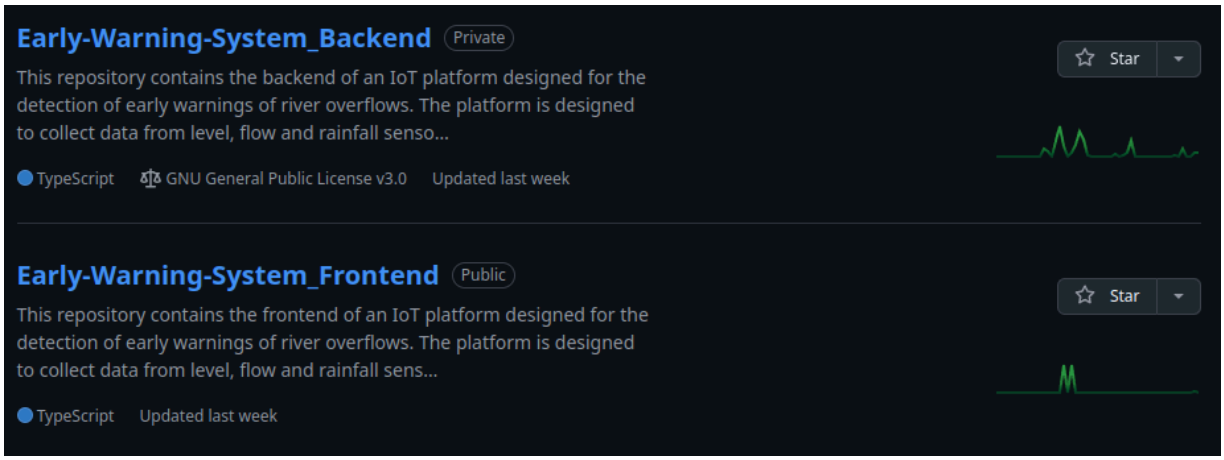
Git, una herramienta que permite registrar el historial de modificaciones, facilitando el seguimiento de cada cambio, la colaboración entre los integrantes del equipo y la resolución de conflictos en el código. Esta práctica asegura que las diferentes etapas del desarrollo queden documentadas, permitiendo revertir cambios no deseados y mantener la integridad del proyecto.

4.3.7.1 Repositorios

El código del sistema se encuentra alojado en un repositorio remoto en GitHub, lo que garantiza accesibilidad, seguridad y una plataforma colaborativa para el equipo de desarrollo.

Figura 49.

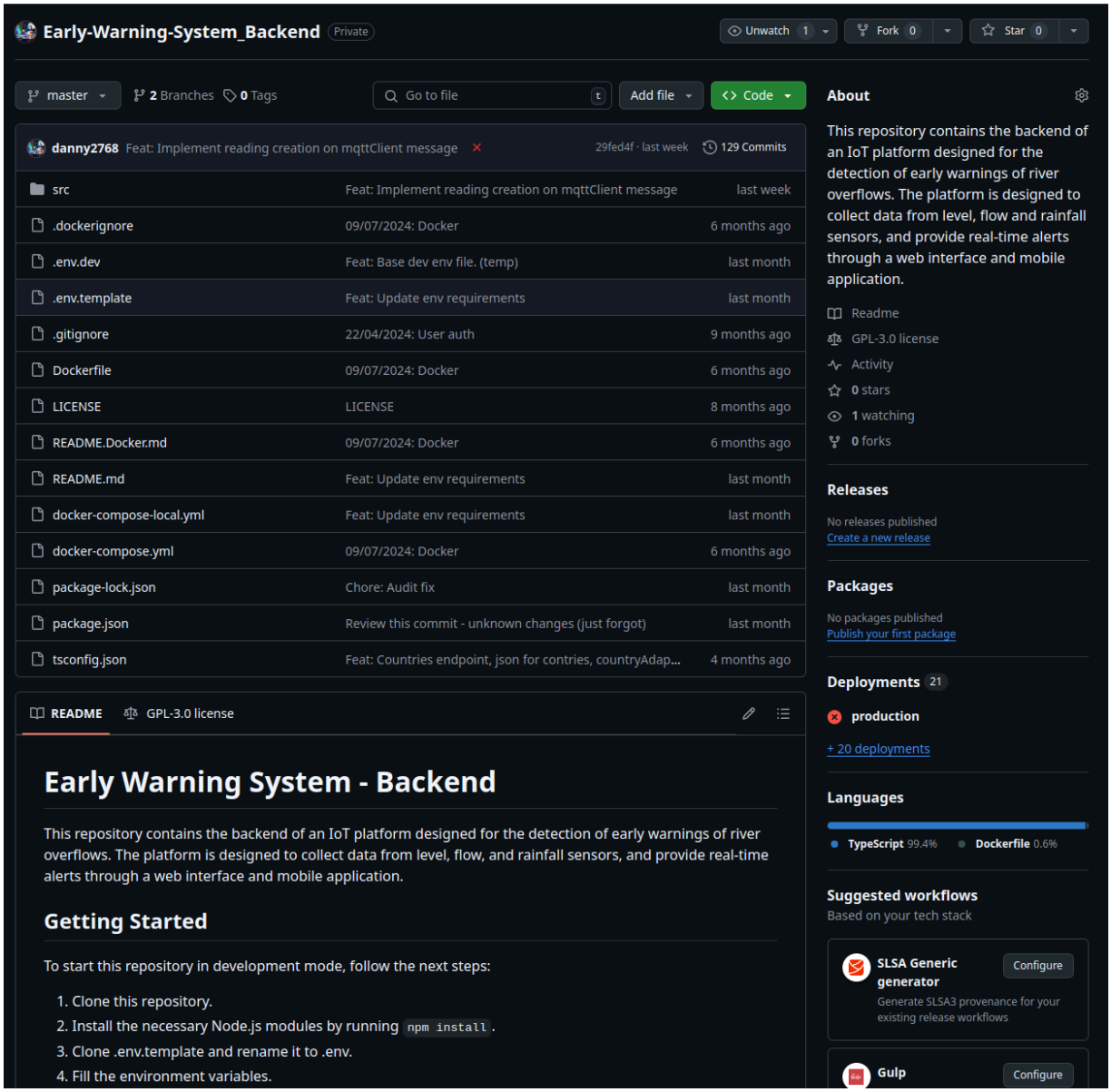
Lista de repositorios del proyecto.



Nota. Tomado de danny2768 - Repositories. (s.f.). GitHub. https://github.com/danny2768?tab=repositories

Figura 50.

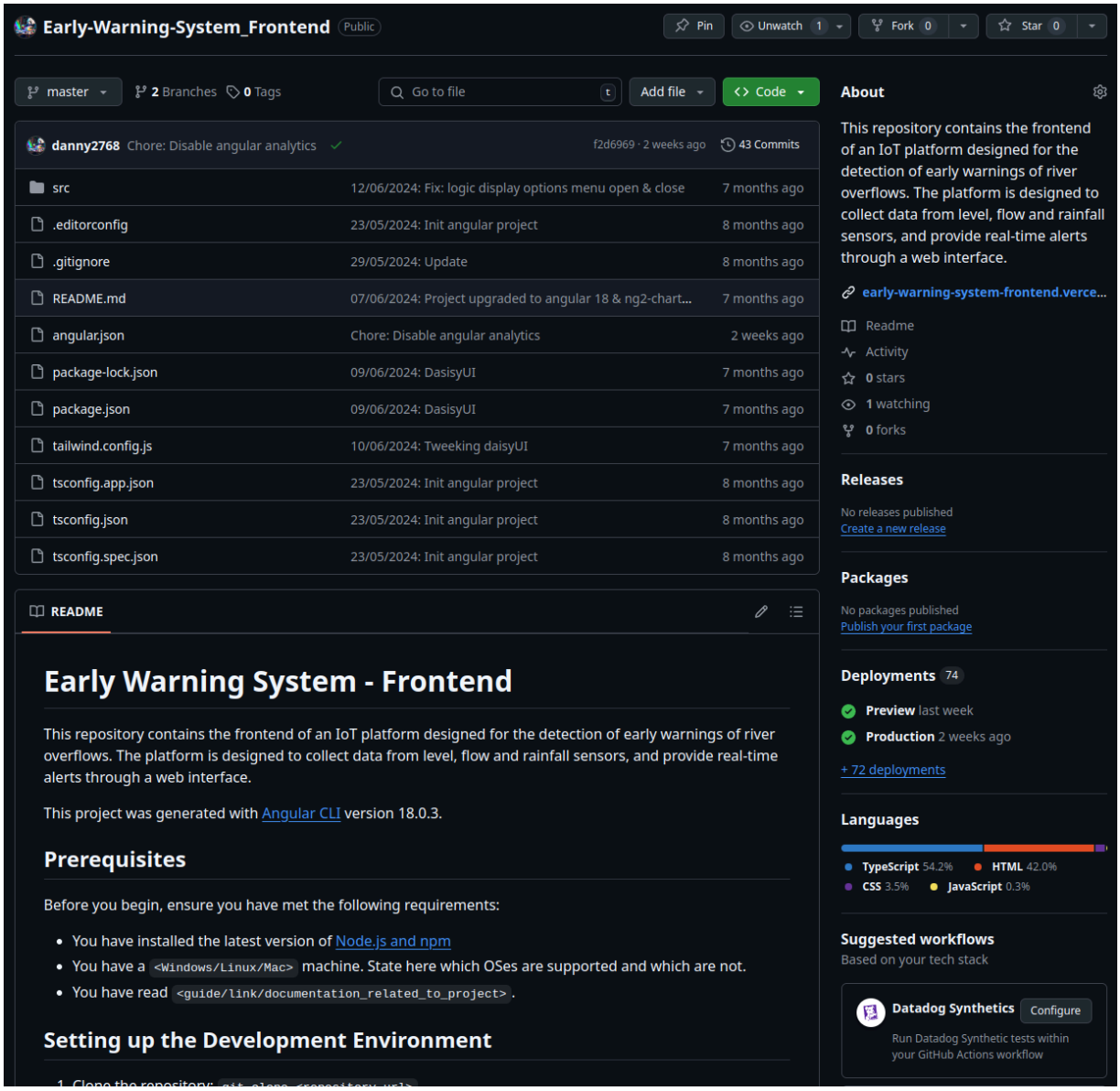
Repositorio del backend.



Nota. Tomado de Danny. (s.f.). GitHub - danny2768/Early-Warning-System_Backend: This repository contains the backend of an IoT platform designed for the detection of early warnings of river overflows. The platform is designed to collect data from level, flow and rainfall sensors, and provide real-time alerts through a web interface. GitHub. https://github.com/danny2768/Early-Warning-System_Frontend

Figura 51.

Repositorio del frontend.



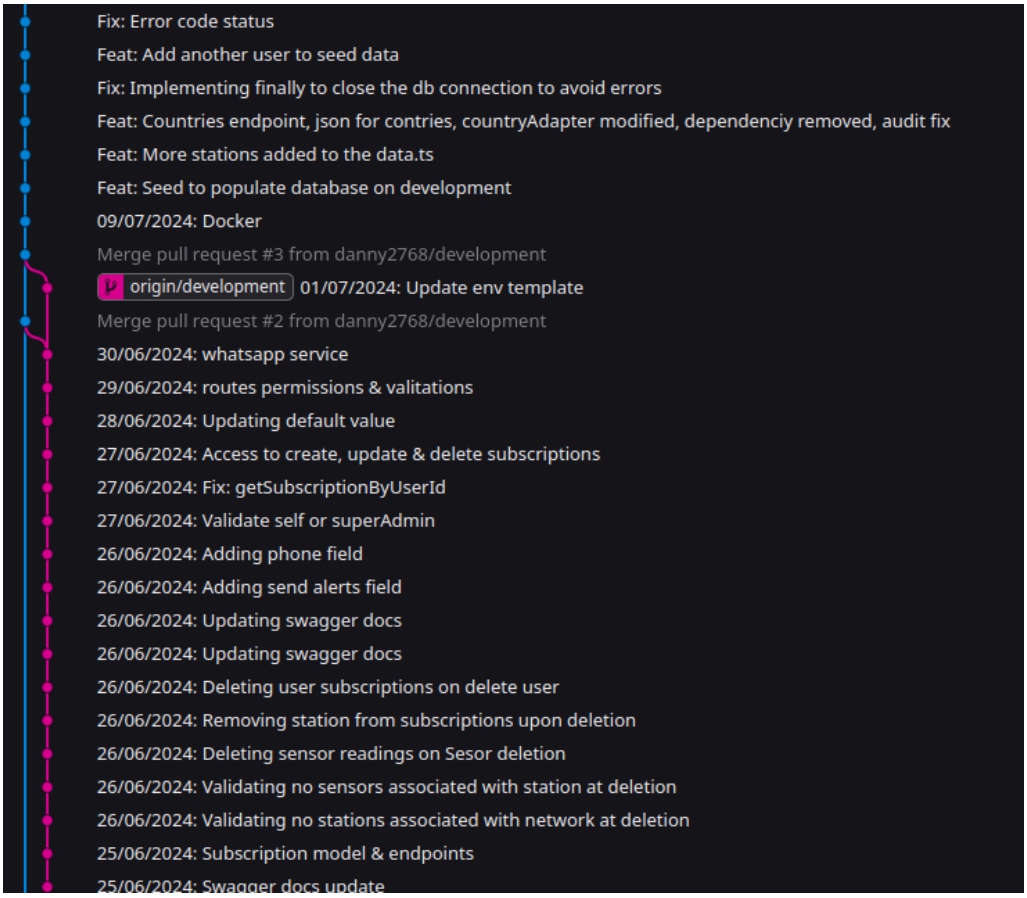
Nota. Tomado de Danny. (s.f.). GitHub - danny2768/Early-Warning-System_Frontend: This repository contains the frontend of an IoT platform designed for the detection of early warnings of river overflows. The platform is designed to collect data from level, flow and rainfall sensors, and provide real-time alerts through a web interface. GitHub. https://github.com/danny2768/Early-Warning-System_Frontend

4.3.7.2 Gráfica de commits

La gráfica de commits evidencia el trabajo continuo y colaborativo realizado por el equipo de desarrollo. Esta gráfica muestra los diferentes puntos en los que se realizaron cambios al código, destacando las contribuciones individuales y el progreso del proyecto.

Figura 52.

Fragmento gráfico de commits.

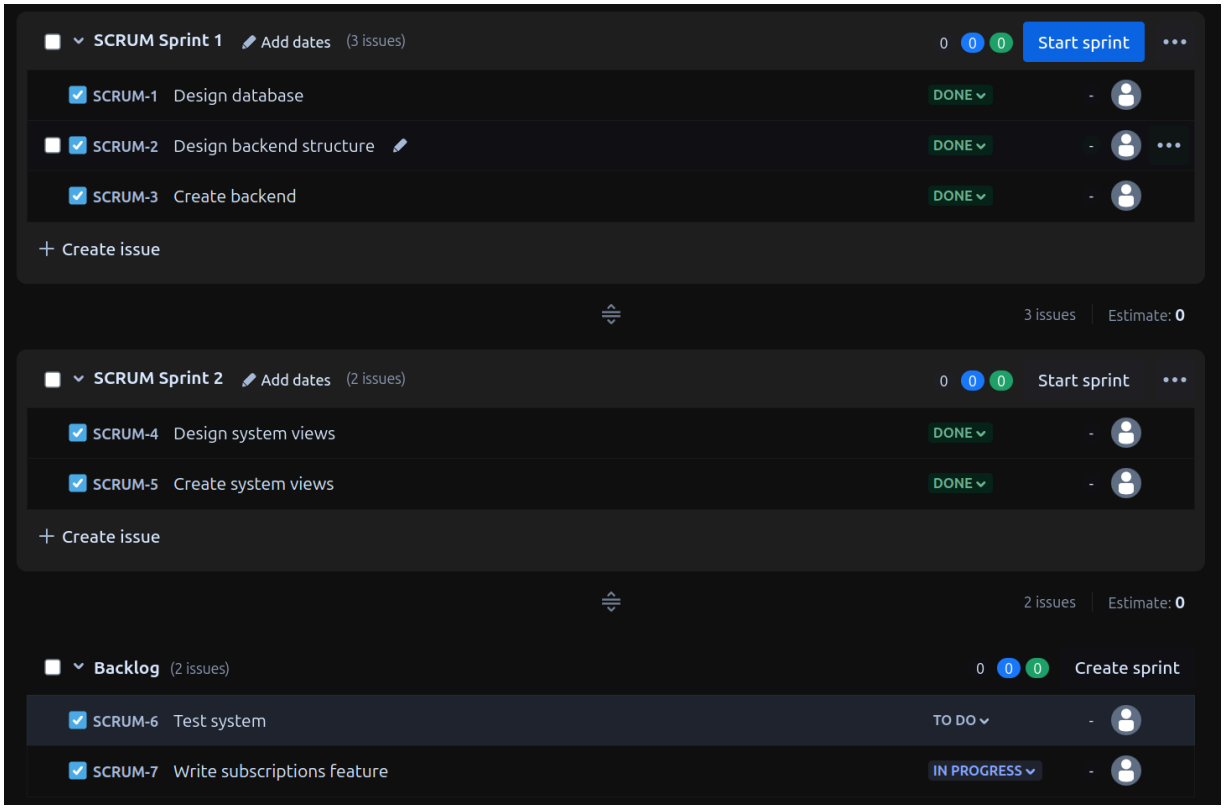


4.3.7.3 Panel de gestión de actividades

El panel de gestión de actividades fue implementado utilizando Jira, una herramienta eficiente para la organización y el seguimiento de proyectos. En este panel, las tareas se organizaron en tres estados principales: Por hacer, En progreso y Completado.

Figura 53.

Fragmento panel de gestión de actividades.



Nota. Elaboración del equipo utilizando Jira.

4.4 Desarrollo del Hardware y Software

El desarrollo del sistema implicó la implementación de un entorno en la nube para alojar los diferentes componentes de software, asegurando su disponibilidad, escalabilidad y correcta interacción con los dispositivos IoT. A continuación, se detallan los pasos seguidos para configurar y desplegar cada elemento.

4.4.1 Creación del servidor y publicación

El sistema utiliza dos plataformas principales para el despliegue de los servidores: Railway para el backend y servicios relacionados, y Vercel para el frontend.

4.4.1.1 Railway

Railway fue la plataforma seleccionada para desplegar la lógica del backend, la base de datos MongoDB y el broker MQTT (EMQX). A continuación, se describe el proceso para cada componente:

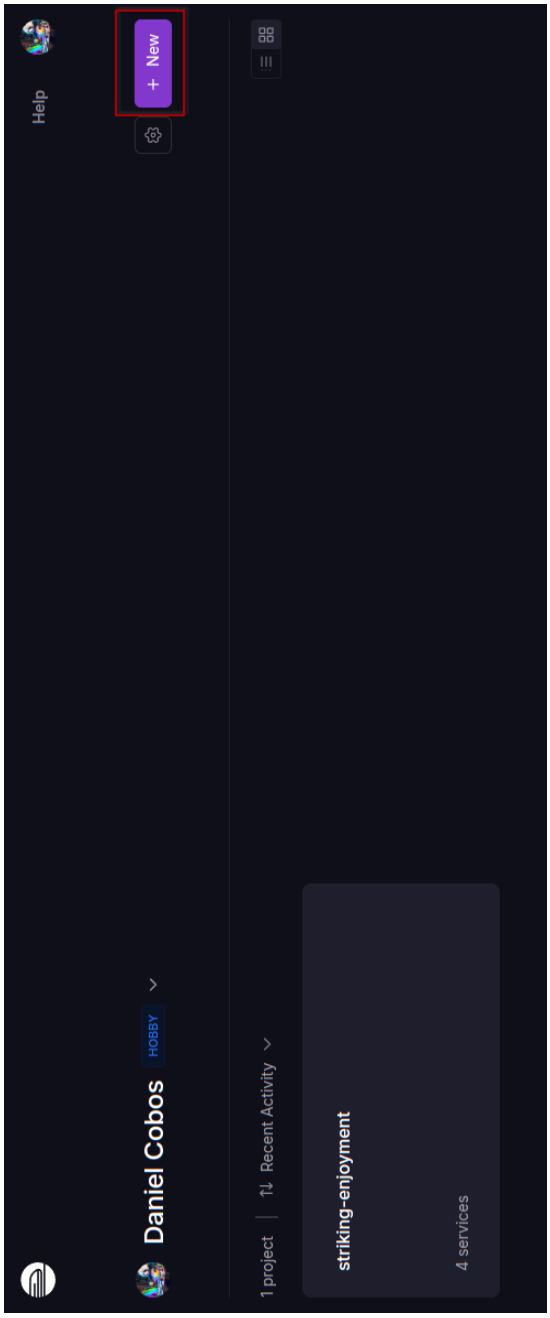
4.4.1.1.1 Creación de un proyecto en Railway

Antes de proceder con el despliegue de servicios específicos, se creó un proyecto en Railway como punto de partida. Este proyecto sirve como un contenedor general para administrar y organizar las instancias y servicios.

- **Acceso al panel de control de Railway:** En este paso, se accede como usuario autenticado al panel de control. Cabe destacar que el usuario está utilizando el plan Hobby, que ofrece una capacidad de 8 vCPU, 8 GB de RAM y 100GB de almacenamiento compartido. Este plan es el mínimo requerido para cumplir con las demandas computacionales del proyecto.
- **Creación de un nuevo proyecto:** En el panel principal se crea un nuevo proyecto:

Figura 54.

Opción crear nuevo proyecto Railway

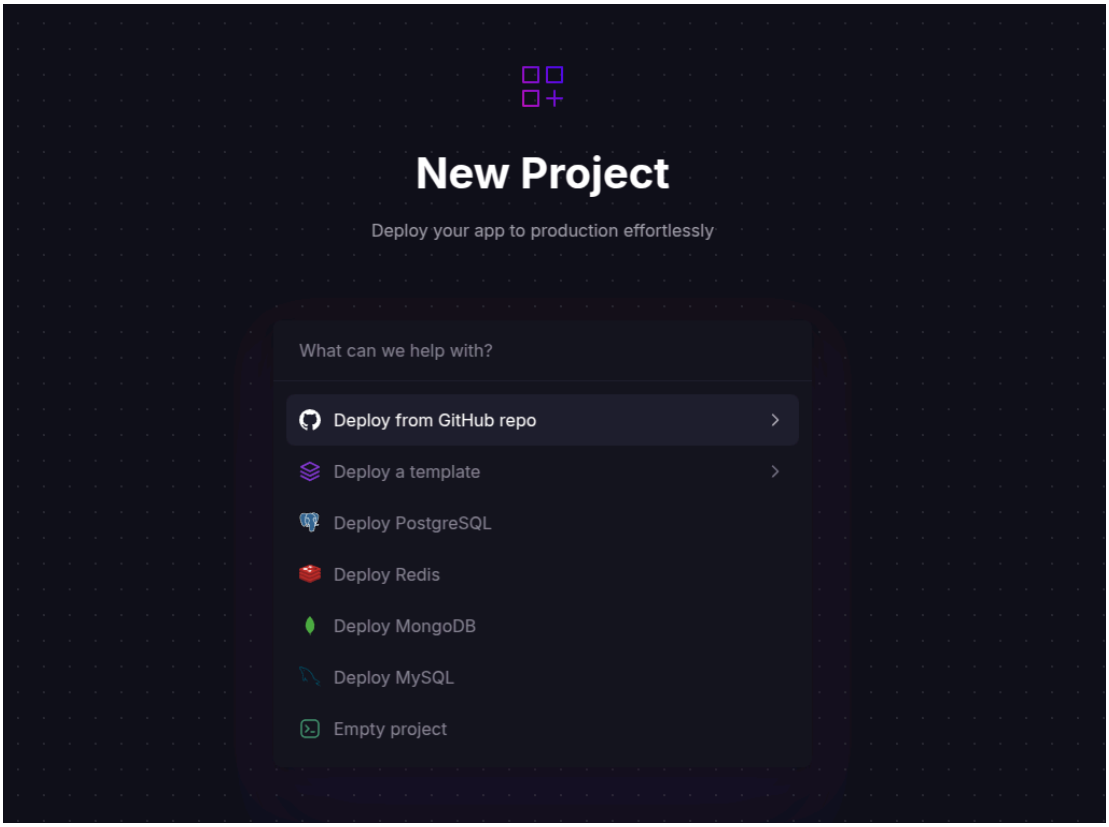


Nota. Tomado de Railway.(s.f.).Railway.

https://railway.com/project/9fabcb9f-2bb3-43fb-aace-03a1976b526c/template?environmentId=16964ced-bebd-43bc-8555-90f241581724.

Figura 55.

Panel creación del nuevo proyecto Railway.



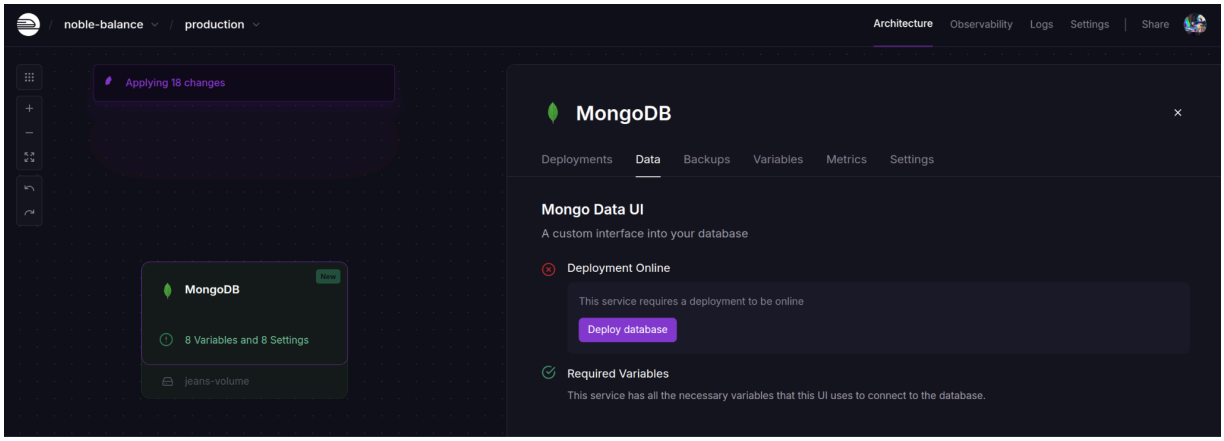
Nota. Tomado de Railway.(s.f.).Railway. *https://railway.com/project/9fabcb9f-2bb3-43fb-aace-03a1976b526c/template?environmentId=16964ced-bebd-43bc-8555-90f241581724*.

4.4.1.1.2 Creación de una instancia de MongoDB

Desde el panel de creación de proyecto se seleccionó la opción ‘Deploy MongoDB’ la cual crea un contenedor que contiene una instancia de MongoDB optimizada para correr en el entorno de Railway.

Figura 56.

Despliegue MongoDB en Railway.



Nota. Tomado de Railway.(s.f.).Railway. https://railway.com/project/9fabcb9f-2bb3-43fb-aace-03a1976b526c/template?environmentId=16964ced-bebd-43bc-8555-90f241581724.

Configuración y acceso

En la sección de Variables de Railway, la plataforma genera automáticamente credenciales seguras para la conexión con la base de datos. Estas credenciales incluyen: MONGO_PUBLIC_URL, MONGO_PASSWORD, MONGO_PORT y MONGO_USER.

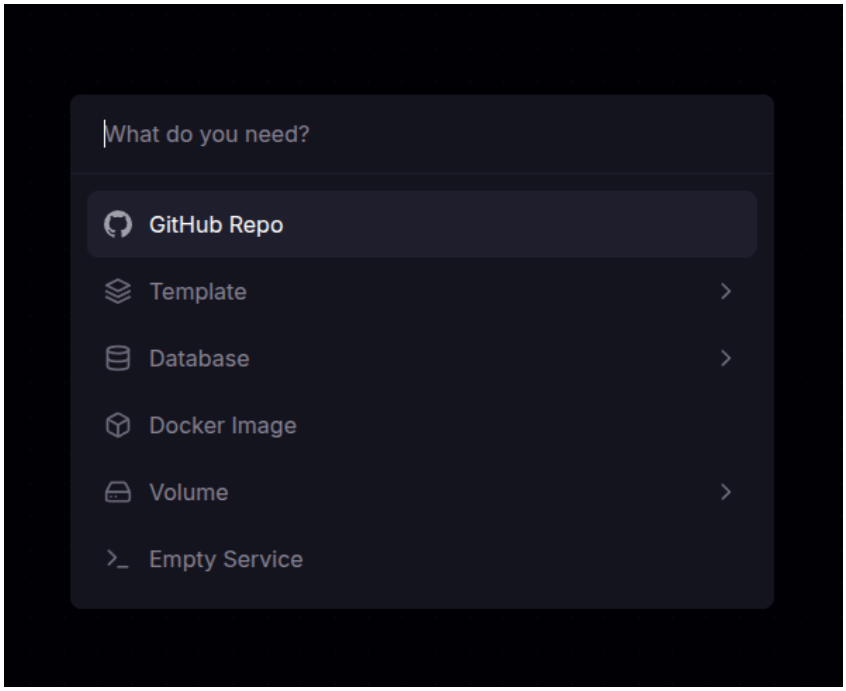
Estas variables pueden ser utilizadas por otros servicios dentro del mismo proyecto para establecer una conexión segura con la base de datos, garantizando así una integración confiable y eficiente entre los componentes del sistema.

4.4.1.1.3 Despliegue del broker MQTT (EMQX)

Desde el panel de proyecto, se seleccionó la opción "New", que permite añadir un nuevo servicio al proyecto. Al hacerlo, la plataforma solicita especificar el tipo de servicio requerido.

Figura 57.

Panel de creación de servicio Railway.

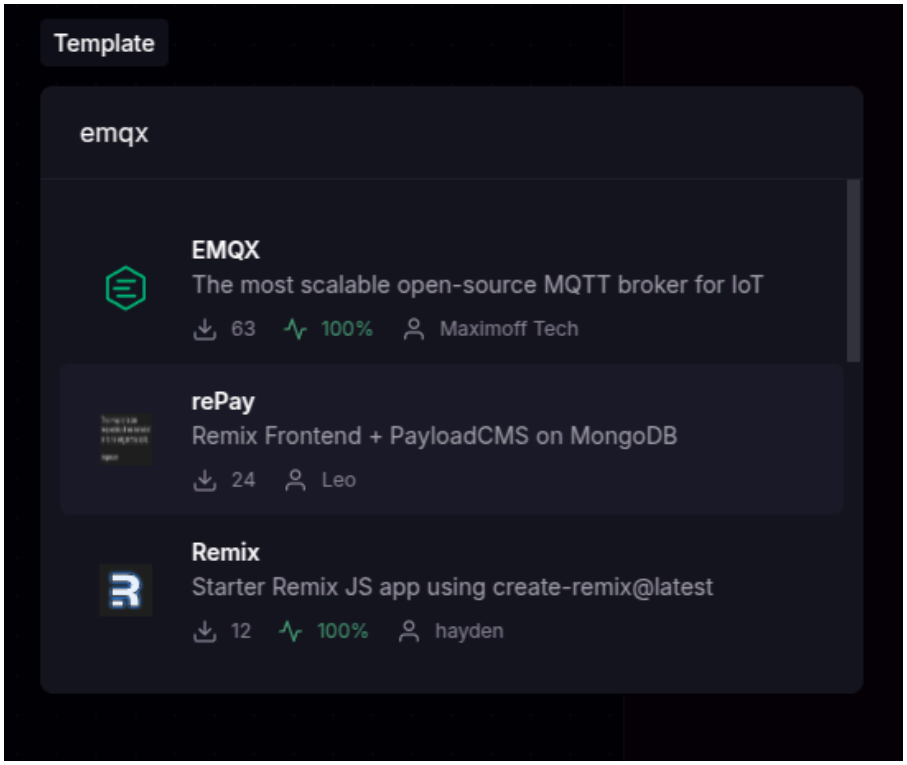


Nota. Tomado de Railway.(s.f.).Railway. https://railway.com/project/9fabcb9f-2bb3-43fb-aace-03a1976b526c/template?environmentId=16964ced-bebd-43bc-8555-90f241581724.

Vamos a seleccionar template y buscamos EMQX:

Figura 58.

Crear un servicio EMQX - Railway.

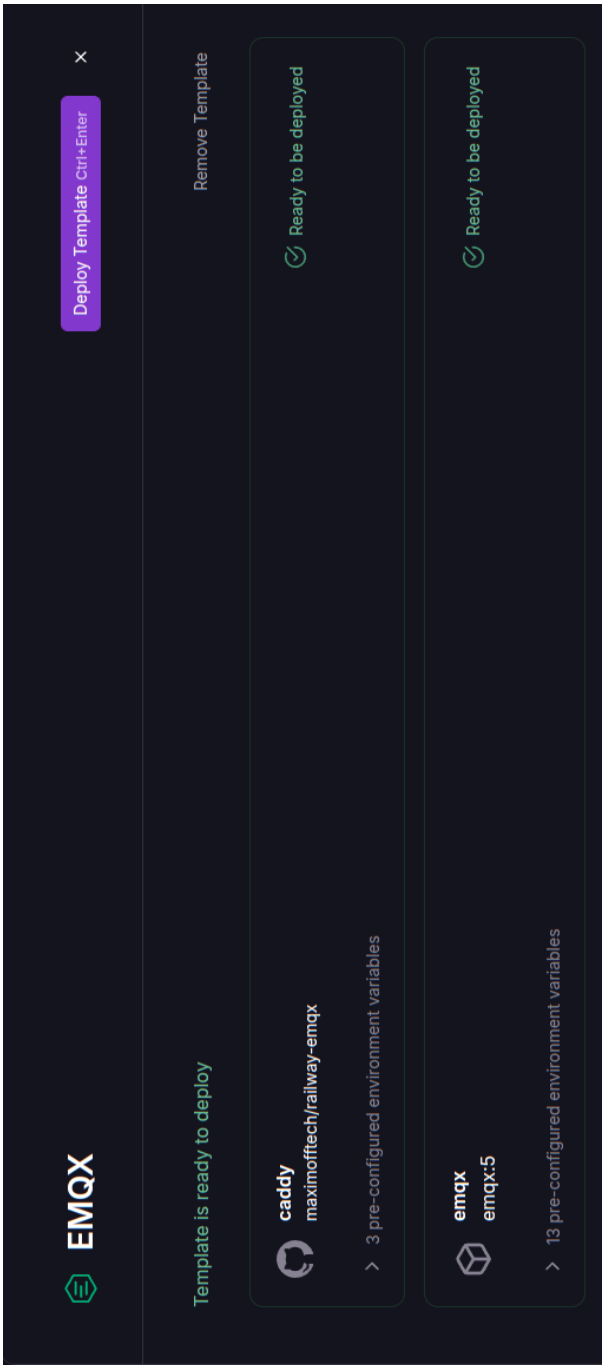


Nota. Tomado de Railway.(s.f.).Railway. https://railway.com/project/9fabcb9f-2bb3-43fb-aace-03a1976b526c/template?environmentId=16964ced-bebd-43bc-8555-90f241581724.

Al seleccionar la opción EMQX, se procede a desplegar las instancias necesarias para el proyecto. La primera instancia corresponde al broker MQTT de EMQX, encargado de gestionar los datos provenientes de los sensores del proyecto. La segunda instancia es Caddy, que se utilizará para desplegar el panel de control o dashboard de EMQX.

Figura 59.

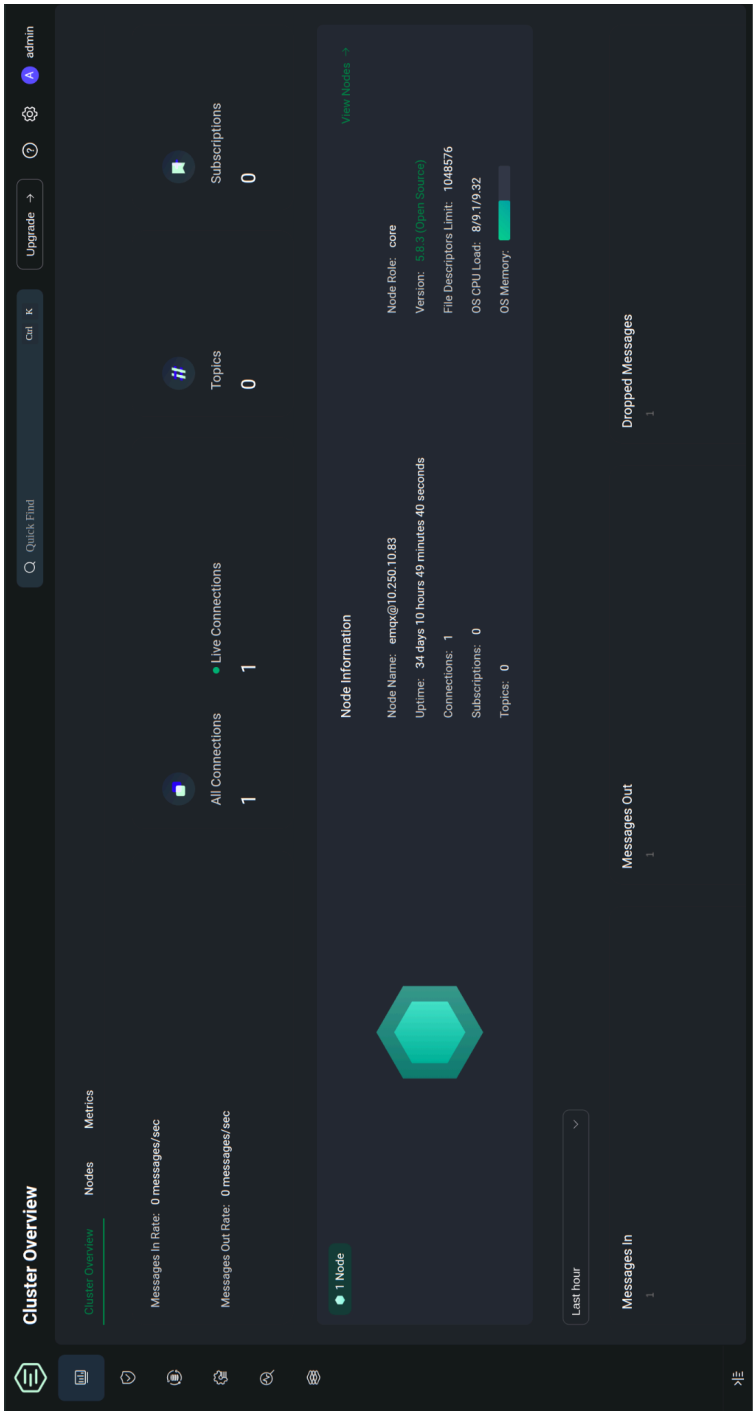
Desplegar servicio EMQX - Railway.



Nota. Tomado de Railway.(s.f.).Railway. https://railway.com/project/9fabcb9f-2bb3-43fb-aace-03a1976b526c/template?environmentId=16964ced-bebd-43bc-8555-90f241581724.

Figura 60.

Dashboard EMQX.



Nota. Tomado de EMQX Dashboard. (s.f.).

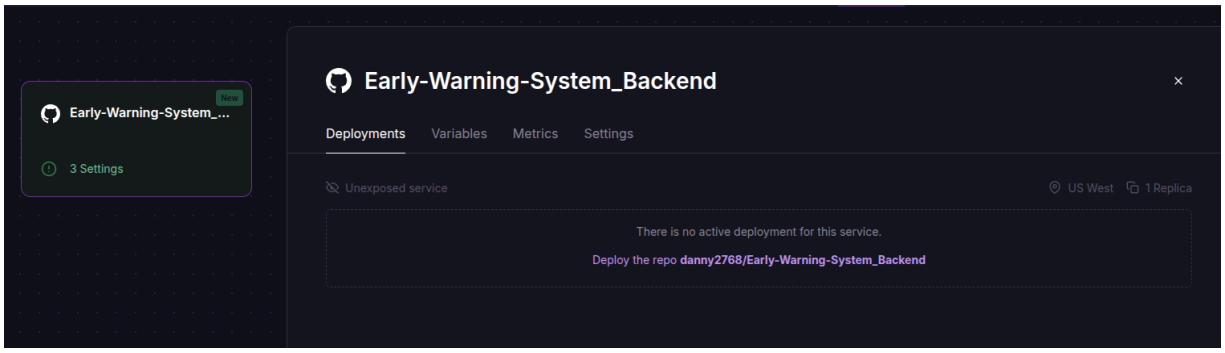
https://caddy-production-dc71.up.railway.app/#/dashboard/overview

4.4.1.1.4 Despliegue del backend desarrollado en Node.js

Para desplegar el backend del proyecto, se seleccionó la opción "Deploy from GitHub Repository" desde el panel de Railway. Se autorizó la integración con GitHub para permitir el acceso a los repositorios. Posteriormente, se seleccionó el repositorio que contiene el código del backend.

Figura 61.

Despliegue backend - Railway.



Nota. Tomado de Railway.(s.f.).Railway. https://railway.com/project/9fabcb9f-2bb3-43fb-aace-03a1976b526c/template?environmentId=16964ced-bebd-43bc-8555-90f241581724.

A continuación, se configuró el despliegue del backend, ya que el proyecto requiere ciertas variables de entorno para su correcto funcionamiento. En la sección "Variables", se ingresaron dichas variables, lo que permitió completar el despliegue del backend de forma exitosa.

El plan Hobby de Railway proporciona URLs con certificados SSL para todos los servicios desplegados, lo que asegura que las conexiones sean seguras y cumplan con los estándares de encriptación. Este paso fue crucial para poder conectar el frontend, el cual está

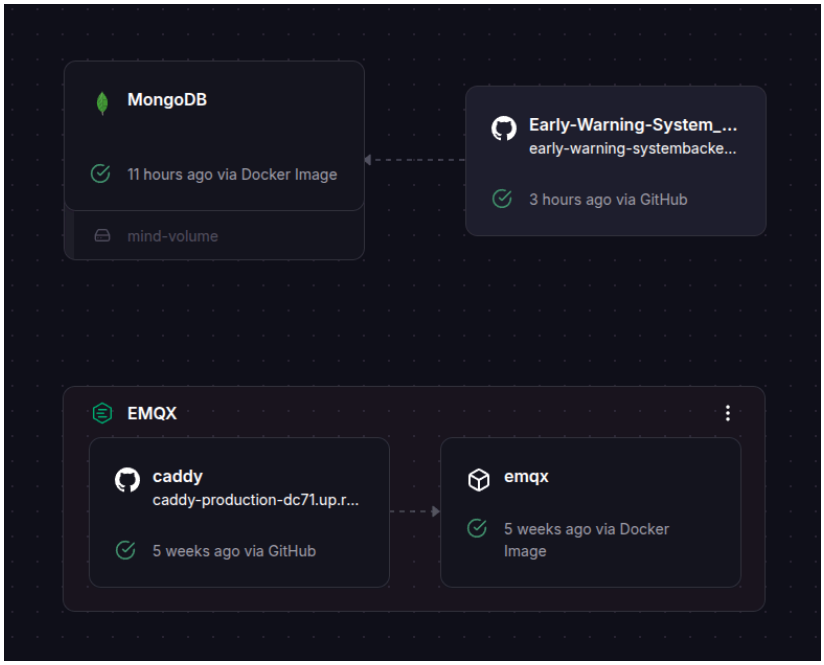
desplegado en otro servidor distinto a Railway, ya que necesitábamos una URL segura para permitir que el frontend interactuara con la API sin problemas. Además, la configuración de SSL permite que se puedan realizar peticiones a la API desde el exterior, como por ejemplo, utilizando herramientas como Postman, lo que facilita la integración con otros servicios y la prueba de la funcionalidad de la API. Para garantizar un correcto funcionamiento del sistema, se configuró esta URL en el apartado de "Settings" del panel de Railway, y se utilizó esta URL como la base para las integraciones con otros servicios, como el frontend y los dispositivos IoT.

4.4.1.1.5 Despliegue exitoso de todos los servicios en Railway

Después de completar el proceso de configuración y despliegue de todos los servicios (base de datos MongoDB, broker MQTT EMQX y backend desarrollado en Node.js), el proyecto mostró un estado completamente operativo en el panel de Railway. La interfaz de Railway permitió monitorear las instancias activas, verificar los logs y realizar ajustes en tiempo real según las necesidades del sistema.

Figura 62.

Despliegue completo - Railway.



Nota. Tomado de Railway.(s.f.).Railway. https://railway.com/project/9fabcb9f-2bb3-43fb-aace-03a1976b526c/template?environmentId=16964ced-bebd-43bc-8555-90f241581724.

4.4.1.2 Vercel

El frontend del sistema, desarrollado en Angular, fue desplegado en Vercel debido a su capacidad para realizar despliegues automáticos y optimizados para aplicaciones web. Los pasos seguidos fueron:

Creación de una cuenta de Vercel y vinculación con Github

El primer paso fue crear una cuenta en Vercel, la cual puede ser registrada usando un correo electrónico o integrando cuentas de servicios como GitHub. Una vez registrada la

cuenta, se procedió a vincularla con GitHub, lo que permitió a Vercel acceder a los repositorios donde se encuentra el código fuente del proyecto. Esta integración facilita el despliegue continuo del frontend cada vez que se realiza un cambio en el repositorio.

Configuración del repositorio de Github

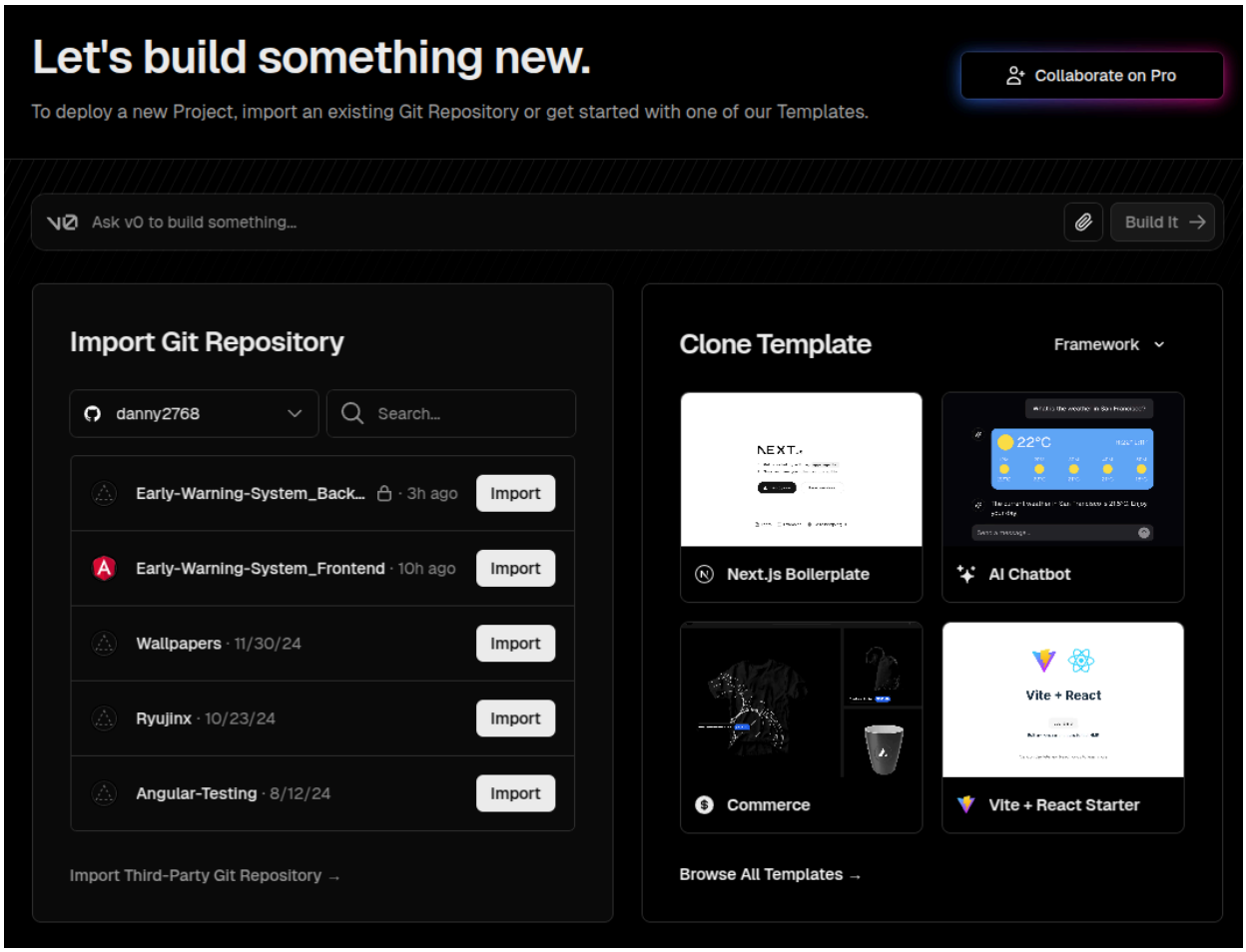
El código fuente del frontend se almacenó en un repositorio de GitHub y se verificó que el proyecto Angular estuviera correctamente configurado para producción mediante el comando ng build, asegurando que los archivos generados estuvieran listos para el despliegue.

Creación del proyecto en Vercel

Para iniciar el despliegue, se creó un nuevo proyecto en Vercel desde el panel de control de la plataforma. Durante la creación del proyecto, se seleccionó la opción de importar el proyecto directamente desde un repositorio de GitHub. Se seleccionó el repositorio que contiene el código del frontend del proyecto y se desplegó.

Figura 63.

Creación nuevo proyecto - Vercel.

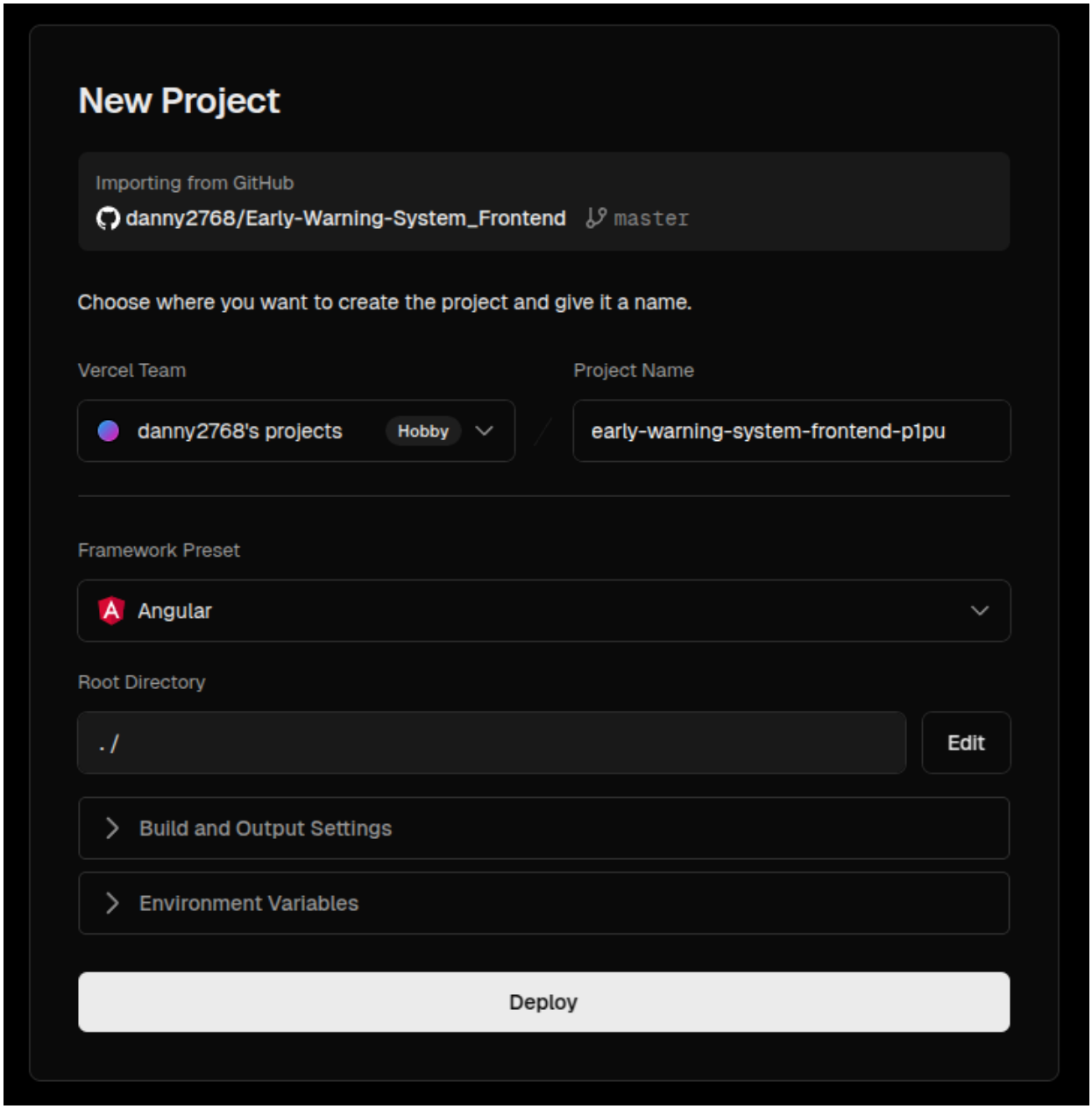


Nota. Adaptado de Dashboard. (s.f.). Vercel.

https://vercel.com/new?teamSlug=danny2768s-project.

Figura 64.

Creación nuevo proyecto - Configuración - Vercel.



Nota. Tomado de Dashboard. (s.f.). Vercel.

https://vercel.com/new?teamSlug=danny2768s-project.

Despliegue automático

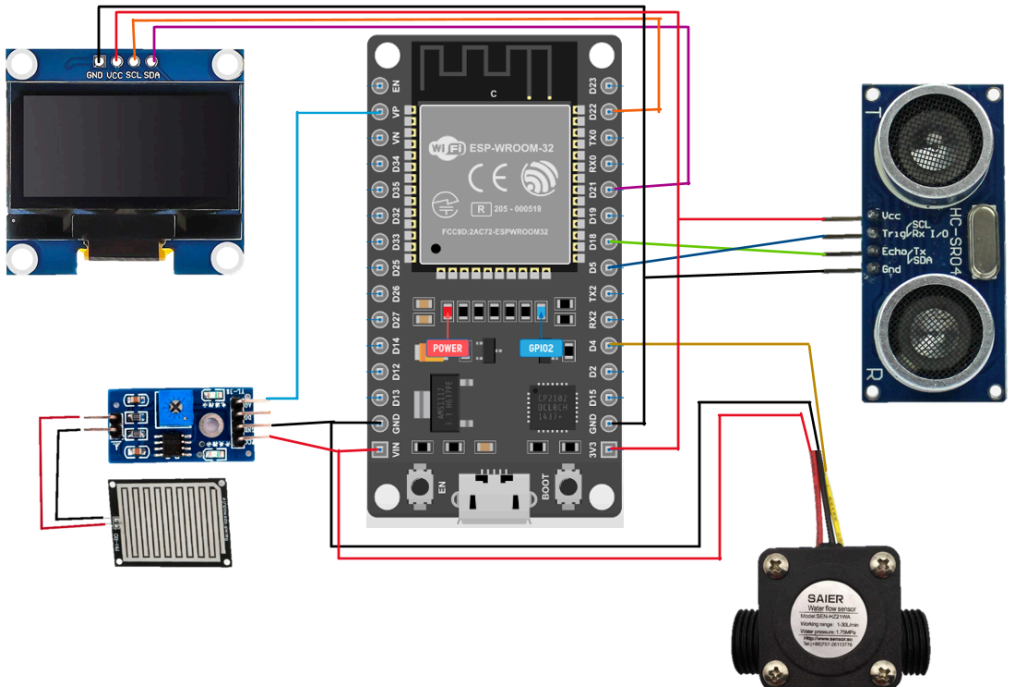
Una vez que la cuenta de Vercel estuvo vinculada con el repositorio de GitHub y se configuró el proyecto, Vercel habilitó el despliegue automático. Al realizar cambios en el repositorio, Vercel detecta los cambios y automáticamente realiza el despliegue de la nueva versión del frontend sin necesidad de intervención manual. Esta integración permite un flujo de trabajo continuo y ágil, con despliegues rápidos y eficientes.

4.4.2 Construcción del hardware

En la implementación del hardware de cada estación, se incorporo la placa de desarrollo DOIT ESP32 DevKit V1, una pantalla OLED SSD1306, un sensor de lluvia YL-83, un sensor para el nivel del agua por medio de ultrasonido HC-SR04 y un sensor de flujo o caudal de agua EN-HZ21WA.

Figura 65.

Creación nuevo proyecto - Configuración - Vercel.



Nota. Imagen combinada y modificada a partir de:

1. Tomado de Mohanan, V. (2024, 12 de marzo). DOIT ESP32 DevKit V1 Wi-Fi Development Board - Pinout Diagram & Arduino Reference - CIRCUITSTATE. CIRCUITSTATE Electronics. https://www.circuitstate.com/pinouts/doit-esp32-devkit-v1-wifi-development-board-pinout-diagram-and-reference/.

2. Sourcewell Devices Pvt Ltd. (2022, 19 de septiembre). Saier SEN-HZ21WA Water Flow Sensor - SourceWell Devices. Sourcewell Devices. https://sourcewell.in/product/sen-hz21wa/

3. Display Oled 0.96" I2C 128*64 SSD1306. (s.f). Naylamp Mechatronics - Perú. https://naylampmechatronics.com/oled/850-display-oled-096-i2c-12864-ssd1306.html,

4. Admin. (2018, 22 mayo). HC-SR04 sensor ultrasonido con atmega-2 - MakerElectronico. MakerElectronico. https://www.makerelectronico.com/hc-sr04-sensor-ultrasonido-atmega/hc-sr04-sensor-ultrasonido-con-atmega-2/

5. Dualtronica. (s. f.). *Sensor de lluvia YL83*. https://dualtronica.com/sensores/107-sensor-de-lluvia.html

La placa de desarrollo DOIT ESP32 DevKit V1 fue programada a través de PlatformIO, utilizando diversas librerías disponibles para enviar los datos suministrados por los sensores al broker EMQX mediante el protocolo MQTT, a través de tópicos de suscripción.

4.4.3 Configuración del Hardware y conexión al broker

Para cada estación, el hardware está configurado para enviar datos en formatos JSON al broker EMQX utilizando el protocolo MQTT. La configuración incluye la dirección del servidor MQTT que para este caso al estar desplegado en railway genera un link “junction.proxy.rlwy.net” y un puerto de acceso autorizado “19243”. El cual apunta al puerto 1883 del broker.

Figura 66.

Creación nuevo proyecto - Configuración - Vercel.

```
// MQTT Broker
const char *mqtt_server = "junction.proxy.rlwy.net";
const int mqtt_port = 19243;
```

Estas estaciones utilizan las librerías PubSubClient y ArduinoJson para facilitar la conexión y publicación de mensajes. El sistema implementa suscripciones a tópicos específicos, como esp32/estacion1/config, para recibir órdenes relacionadas con la configuración de intervalos de publicación de los sensores, a su vez los datos captados por estos son publicados periódicamente en tópicos únicos asignados a cada estación.

Figura 67.

Publicación de datos por tópicos únicos.

```
void sendSensorData(const char *id, float value, const char *type)
{
  StaticJsonDocument<256> doc;
  doc["station"] = "aabbccddee11223344556677";
  JsonArray sensors = doc.createNestedArray("sensors");
  JsonObject sensor = sensors.createNestedObject();
  sensor["id"] = id;
  sensor["value"] = value;
  sensor["type"] = type;

  char buffer[256];
  size_t n = serializeJson(doc, buffer);

  // Mensaje de depuración
  Serial.print("Publicando mensaje: ");
  Serial.println(buffer);

  // Publicar los datos en los tópicos MQTT
  if (client.publish("esp32/estacion2", buffer, n))
  {
    Serial.println("Mensaje publicado exitosamente");
  }
  else
  {
    Serial.println("Error al publicar el mensaje");
  }
}
```

Además, para garantizar la conectividad, el cliente MQTT implementa una rutina de reconexión automática que verifica constantemente el estado de la conexión con el broker y restablece la comunicación en caso de fallos.

Figura 68.

Rutina de reconexión y suscripción a tópico de configuración.

```
void reconnect()
{
  while (!client.connected())
  {
    if (client.connect(clientid, mqtt_user, mqtt_pass))
    {
      Serial.println("Connected to MQTT Broker!");
      client.subscribe("esp32/estacion2/config"); // Suscribirse al tópico de configuración
    }
    else
    {
      Serial.println("Failed to connect, trying again in 5 seconds...");
      delay(5000);
    }
  }
}
```

4.4.4 Desarrollo del backend

El desarrollo del backend del sistema se realizó utilizando Node.js como el entorno de ejecución y TypeScript como lenguaje de programación. La arquitectura empleada sigue principios de Clean Architecture, organizando el código en diferentes capas para mantener la modularidad y facilitar el mantenimiento. Estas capas son: Dominio, Presentación, Data y Config. A continuación, se explica cómo se implementaron los componentes clave dentro de la arquitectura, incluyendo modelos, servicios, DTOs, controladores, middleware y rutas.

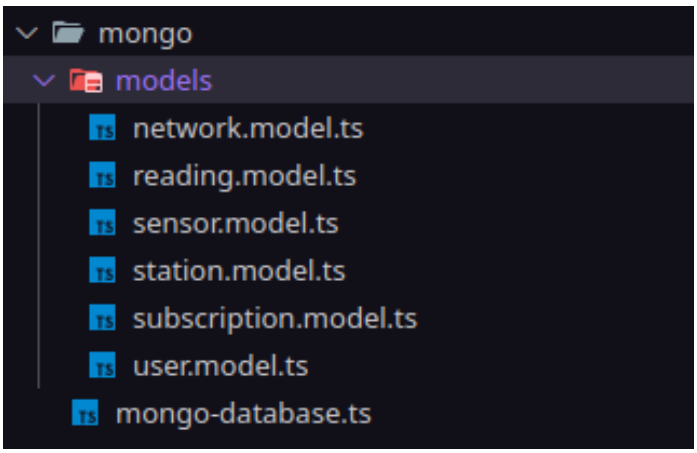
4.4.4.1 Modelos

Los modelos del backend están definidos dentro del directorio data/mongo/models, utilizando Mongoose para interactuar con la base de datos MongoDB. Cada modelo representa una entidad en la base de datos y está asociado con una colección de documentos. Los

modelos son responsables de la validación de datos y la transformación de estos para su almacenamiento.

Figura 69.

Modelos de Mongo.



Cada modelo está vinculado a su respectiva entidad, y algunos de ellos implementan métodos estáticos que permiten convertir las representaciones del modelo en objetos más manejables, facilitando la interacción con la capa de dominio.

Figura 70.

Fragmento de código - Modelo de estación.

```
import mongoose, { Schema } from "mongoose";

const stationSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [ true, 'name is required'],
    },
    city: {
        type: String,
        // required: [ true, 'city is required'],
    },
    state: {
        type: String,
        required: [ true, 'state is required'],
    },
    countryCode:{
        type: String,
        required: [ true, 'countryCode is required'],
    },
    coordinates: {
        longitude: {
            type: Number,
            required: [ true, 'longitude is required'],
        },
        latitude: {
            type: Number,
            required: [ true, 'latitude is required'],
        },
        _id: false,
    },
    networkId: {
        type: Schema.Types.ObjectId,
        required: [ true, 'networkId is required'],
        ref: 'Network',
    },
    isVisibleToUser: {
        type: Boolean,
        required: true,
        default: true,
    },
}, {
    timestamps: true,
});

stationSchema.set('toJSON', {
    virtuals: true,
    versionKey: false,
    transform: function (doc, ret) {
        delete ret._id;
    }
});

export const StationModel = mongoose.model("Station", stationSchema);
```

4.4.4.2 Servicios

Los servicios son responsables de contener la lógica de negocio de la aplicación. En el directorio presentation/services, se encuentran los servicios que ejecutan las operaciones solicitadas por los controladores. Cada servicio interactúa con los modelos de la base de datos y proporciona métodos que permiten realizar operaciones de lectura, escritura, actualización y eliminación (CRUD).

Los servicios del sistema son:

- **Auth Service:** Encargado de la autenticación y autorización de usuarios.
- **Email Service:** Maneja el envío de correos electrónicos, como notificaciones.
- **MQTT Service:** Conecta con el broker MQTT para la recepción y envío de datos en tiempo real.
- **Network Service:** Realiza las operaciones relacionadas con las redes de estaciones.
- **Reading Service:** Gestiona las lecturas de sensores.
- **Sensor Service:** Administra los sensores en las estaciones de monitoreo.
- **Station Service:** Opera sobre las estaciones y sus parámetros.
- **Subscription Service:** Administra las suscripciones a las alertas de río.
- **User Service:** Maneja los usuarios y sus configuraciones.
- **Shared Service:** Encargado de la lógica compartida a través de los servicios.

Figura 71.

Fragmento de código - Servicio de estación.

```
export class StationService {

    constructor(
        private readonly sharedService: SharedService,
    ) {}

    private async validateSensors( sensors: string[] ) {
        sensors.forEach(sensorId => {
            this.sharedService.validateId( sensorId, `sensors contains an invalid id: ${sensorId}` );
        });

        await Promise.all(
            sensors.map( sensorId => this.sharedService.validateSensorById(sensorId) )
        );
    }

    private async validateNetworkId( networkId: string ) {
        this.sharedService.validateId(networkId, 'Invalid networkId'); // Regex validation
        await this.sharedService.validateNetworkById(networkId); // DB validation existance
    }

    public async getStations( paginationDto: PaginationDto ) {
        const { page, limit } = paginationDto;
        try {
            const [ total, stations ] = await Promise.all([
                StationModel.countDocuments(),
                StationModel.find()
                    .skip( (page - 1) * limit )
                    .limit( limit )
            ]);

            const stationsObj = stations.map( station => StationEntity.fromObj(station) );

            const totalPages = Math.ceil( total / limit );

            return {
                pagination: {
                    page: page,
                    limit: limit,
                    totalItems: total,
                    totalPages: totalPages,
                    next: (page < totalPages) ? `/api/stations?page=${page + 1}&limit=${limit}` : null,
                    prev: (page - 1 > 0) ? `/api/stations?page=${page - 1}&limit=${limit}` : null,
                    first: `/api/stations?page=1&limit=${limit}`,
                    last: `/api/stations?page=${totalPages}&limit=${limit}`,
                },
                stations: stationsObj
            }
        } catch (error) {
            if (error instanceof CustomError) throw error;
            throw CustomError.internalServer(`${error}`);
        }
    };

    public async getStationsVisibleToUser( paginationDto: PaginationDto ) {
```

4.4.4.3 Data transfer objects (DTOs)

Los Data transfer objects (DTOs) son utilizados para transferir datos entre las distintas capas de la aplicación. Los DTOs se encuentran en el directorio domain/dtos y son fundamentales para la validación y manipulación de datos dentro de los controladores y servicios. Cada DTO representa una estructura de datos esperada, como las solicitudes de creación o actualización.

Por ejemplo:

- **Create User DTO:** Define los campos necesarios para crear un nuevo usuario.
- **Update User DTO:** Define los campos para actualizar la información de un usuario.
- **Create Network DTO:** Define los datos requeridos para crear una nueva red.
- **Update Network DTO:** Estructura usada para actualizar una red existente.

Estos DTOs proporcionan una capa de validación, asegurando que los datos recibidos en las peticiones sean correctos y completos antes de ser procesados.

Figura 72.

Data Transfer Objects del sistema.

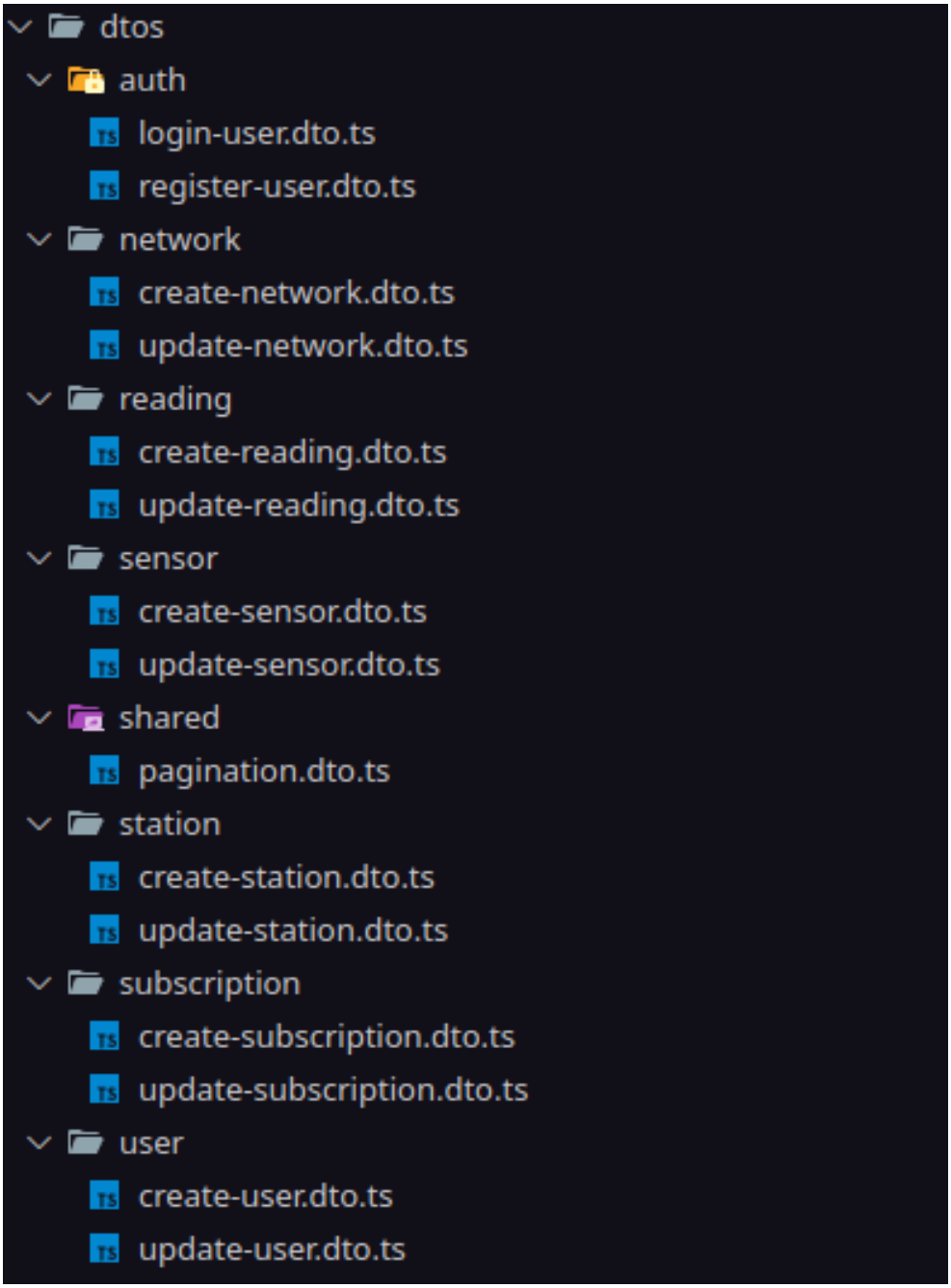


Figura 73.

Fragmento de código - Create sensor DTO.

```
import { Threshold } from "../../interfaces/threshold.interface";
import { SensorType } from "../../interfaces/types";

const SENSOR_TYPES: SensorType[] = ["level", "flow", "rain"];

...
export class CreateSensorDto {

    private constructor(
        public readonly name: string,
        public readonly sensorType: SensorType,
        public readonly sendingInterval: number,
        public readonly stationId: string,
        public readonly sendAlerts: boolean,
        public readonly threshold?: Threshold,
    ) {}

    public static create(object: { [key: string]: any }): [ string?, CreateSensorDto? ] {
        const { name, sensorType, sendingInterval, stationId, sendAlerts, threshold } = object;

        if (!name) return ['Property name is required']
        if (!sensorType) return ['Property sensorType is required']
        if (!SENSOR_TYPES.includes(sensorType)) return ['Property sensorType is invalid']

        // Threshold should only be required if sensorType is level
        if (sensorType === 'level' && !threshold) {
            if (!threshold) return ['Property threshold is required']
            if (typeof threshold !== 'object') return ['Property threshold must be an object']
            if (Object.keys(threshold).length === 0) return ['Property threshold must not be empty']

            const { yellow, orange, red, ...extraThreshold } = threshold;
            if (!yellow) return ['Property threshold.yellow is required']
            if (!orange) return ['Property threshold.orange is required']
            if (!red) return ['Property threshold.red is required']
            if (Object.keys(extraThreshold).length > 0) return [`Property threshold has unexpected fields: ${Object.keys(extraThreshold).join(', ')}`]

            if (typeof yellow !== 'number') return ['Property threshold.yellow must be a number']
            if (typeof orange !== 'number') return ['Property threshold.orange must be a number']
            if (typeof red !== 'number') return ['Property threshold.red must be a number']
        }

        if (!sendingInterval) return ['Property sendingInterval is required']
        if (typeof sendingInterval !== 'number') return ['Property sendingInterval must be a number']

        if (!stationId) return ['Property stationId is required']
        if (typeof stationId !== 'string') return ['stationId property must be a string'];
        if (stationId.trim() === '') return ['stationId property must be a non-empty string'];

        if (!sendAlerts) return ['Property sendAlerts is required']
        if (typeof sendAlerts !== 'boolean') return ['Property sendAlerts must be a boolean']


        return [undefined, new CreateSensorDto( name, sensorType, sendingInterval, stationId, sendAlerts, threshold )];
    }
}
```

4.4.4.4 Controladores

Los controladores son responsables de manejar las peticiones HTTP y coordinar la respuesta adecuada. En el directorio presentation, se encuentran los controladores organizados por módulos del sistema, como usuarios, estaciones, redes, sensores, entre otros.

Cada controlador define las rutas específicas que gestionan las solicitudes de los usuarios. Los controladores reciben las peticiones, validan los datos usando los DTOs, invocan los métodos del servicio correspondiente y finalmente envían una respuesta con la información o el estado de la operación.

Ejemplo de flujo en un controlador:

1. Se recibe una petición en una ruta específica (por ejemplo, POST /create-user).
2. El controlador valida los datos de la petición usando un DTO (por ejemplo, CreateUserDTO).
3. Si los datos son válidos, el controlador invoca el servicio adecuado (por ejemplo, userService.createUser()).
4. El servicio maneja la lógica de negocio, como interactuar con la base de datos.
5. El controlador retorna una respuesta HTTP con el resultado de la operación (éxito o error).

Figura 74.

Fragmento de código - User controller.

```
export class UsersController {

    constructor(
        public readonly userService: UserService
    ) {}

    private handleError = ( error: unknown, res: Response ) => {
        if (error instanceof CustomError) {
            return res.status( error.statusCode ).json({ error: error.message })
        }
        console.log(`${error}`);
        return res.status(500).json({ error: 'Internal server error' })
    };

    public getUsers = ( req: Request, res: Response ) => {
        const { page = 1, limit = 10 } = req.query;
        const [ error, paginationDto ] = PaginationDto.create( +page, +limit );
        if (error) return res.status(400).json({error});

        this.userService.getUsers( paginationDto! )
            .then( users => res.json(users) )
            .catch( error => this.handleError(error, res) );
    };

    public getSelf = (req: Request, res: Response) => {
        const currentUser = req.body.user;

        this.userService.getSelf(currentUser)
            .then(user => res.json(user))
            .catch(error => this.handleError(error, res));
    };

    public getUserById = ( req: Request, res: Response ) => {
        this.userService.getUserById(req.params.id)
            .then( user => res.json(user) )
            .catch( error => this.handleError(error, res) );
    };

    public createUser = ( req: Request, res: Response ) => {
        const [error, createUserDto] = CreateUserDto.create(req.body);
        if (error) return res.status(400).json({error});

        this.userService.createUser( createUserDto! )
            .then( user => res.status(201).json(user) )
            .catch(error => this.handleError(error, res));
    };

    public updateUser = ( req: Request, res: Response ) => {
        const id = req.params.id;
        const [error, updateUserDto] = UpdateUserDto.create({ ...req.body, id});
        if (error) return res.status(400).json({error});

        const currentUserRole = req.body.user.role;

        this.userService.updateUser( updateUserDto!, currentUserRole )
            .then( user => res.json(user) )
            .catch(error => this.handleError(error, res));
    };
```

4.4.4.5 Middleware

El middleware juega un papel crucial en el proceso de manejo de peticiones, ya que se ejecuta entre la solicitud del cliente y la respuesta del servidor. Los middleware son utilizados para tareas como la autenticación, validación de permisos, y manejo de errores.

En el proyecto, uno de los middleware más importantes es el auth.middleware.ts, que verifica si la solicitud contiene un token JWT válido para autenticar al usuario antes de que pueda acceder a las rutas protegidas.

Figura 75.

Fragmento de código - Auth middleware.

```
import { NextFunction, Request, Response } from "express";
import { JwtAdapter } from "../../config";
import { UserModel } from "../../data";
import { UserEntity } from "../../domain";

Daniel Cobos, 4 months ago | 1 author (Daniel Cobos)
export class AuthMiddleware {
    constructor() {}

    private static async extractAndValidateToken(req: Request, expectedRoles: string[] = []) {
        const authorization = req.header('Authorization');
        if (!authorization) throw { status: 401, message: 'Unauthorized. No token provided' };

        if (!authorization.startsWith('Bearer ')) throw { status: 401, message: 'Unauthorized. Invalid Bearer token' };

        const token = authorization.split(' ').at(1) || '';
        const payload = await JwtAdapter.verifyToken<{ id: string, role?: string }>(token);
        if (!payload) throw { status: 401, message: 'Unauthorized. Invalid token' };

        const user = await UserModel.findById(payload.id);
        if (!user) throw { status: 401, message: 'Unauthorized. Invalid token' };

        if (expectedRoles.length && !expectedRoles.some(role => user.role.includes(role))) {
            throw { status: 403, message: 'Forbidden. You dont have permission to access this resource' };
        }

        return user;
    }

    private static async validateTokenWithRole(req: Request, res: Response, next: NextFunction, roles: string[]) {
        try {
            const user = await AuthMiddleware.extractAndValidateToken(req, roles);
            req.body.user = UserEntity.fromObj(user);
            next();
        } catch (error: any) {
            res.status(error.status || 500).json({ error: error.message || 'Internal server error' });
            if (error.status === 500 || !error.status) {
                console.log(error);
            }
        }
    }

    static async validateSelfOrAdminToken(req: Request, res: Response, next: NextFunction) {
        try {
            const user = await AuthMiddleware.extractAndValidateToken(req);
            if (req.params.id === user.id || user.role.includes('ADMIN_ROLE') || user.role.includes('SUPER_ADMIN_ROLE')) {
                req.body.user = UserEntity.fromObj(user);
                next();
            } else {
                throw { status: 403, message: 'Forbidden. You dont have permission to access this resource' };
            }
        } catch (error: any) {
            res.status(error.status || 500).json({ error: error.message || 'Internal server error' });
            if (error.status === 500 || !error.status) {
                console.log(error);
            }
        }
    }
```

4.4.4.6 Rutas

Las rutas definen los puntos de entrada a la aplicación y son configuradas en el directorio presentation/routes.ts y en los módulos específicos (por ejemplo, users/routes.ts, stations/routes.ts). Cada ruta está asociada con un controlador y su método correspondiente.

El flujo de una petición HTTP comienza cuando el cliente hace una solicitud a una de las rutas definidas. El enrutador de la aplicación recibe la solicitud, identifica la ruta correspondiente y redirige la petición al controlador que maneja esa ruta. Una vez procesada, el controlador devuelve una respuesta al cliente.

Las rutas se estructuran de manera que cada módulo del sistema tiene su propio conjunto de rutas. Por ejemplo:

- **/users/routes.ts:** Define las rutas relacionadas con los usuarios.
- **/stations/routes.ts:** Define las rutas para las estaciones de monitoreo.
- **/networks/routes.ts:** Define las rutas para la administración de redes de sensores.

Figura 76.

Fragmento de código - Auth middleware.

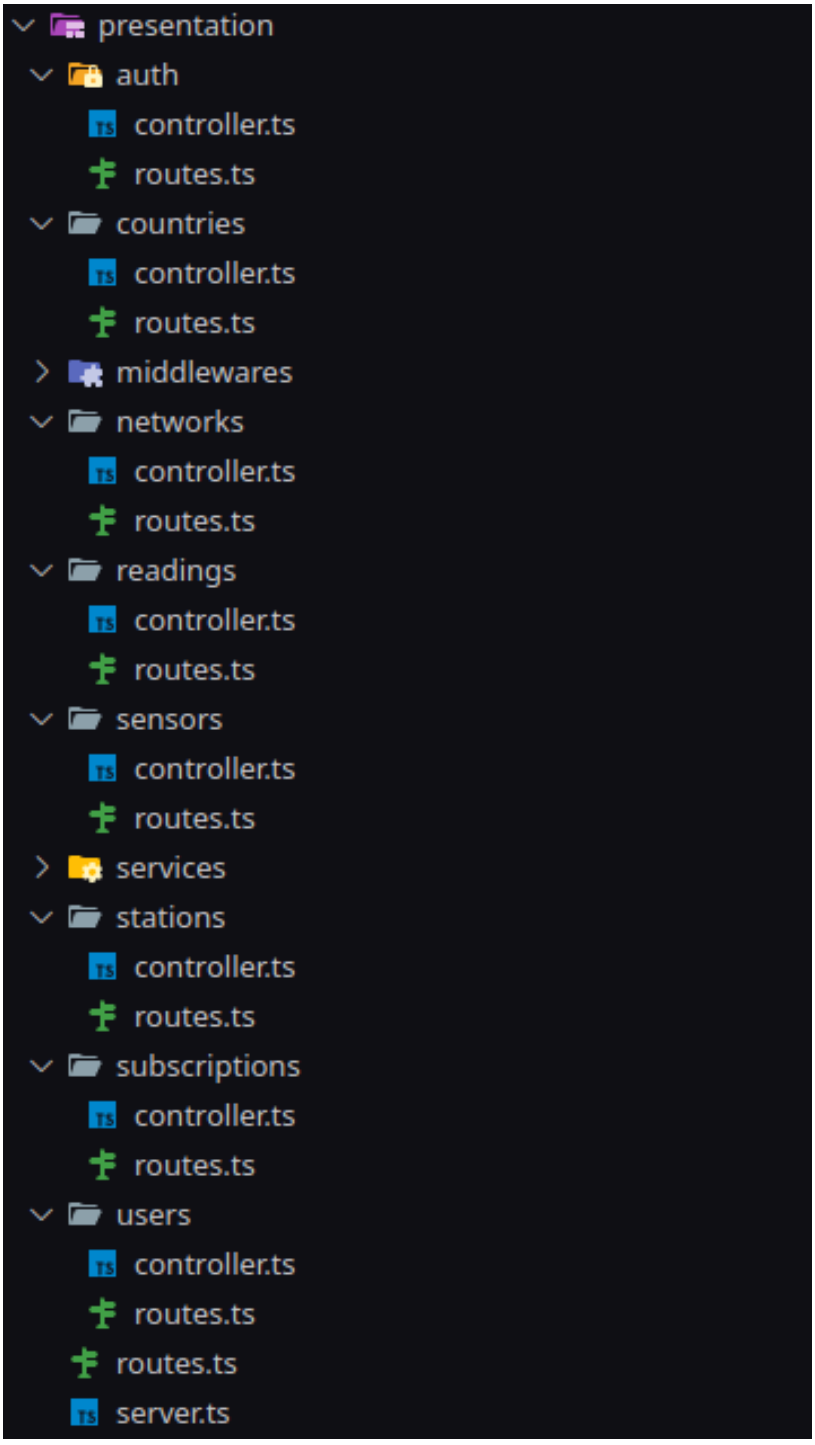


Figura 77.

Fragmento de código - User routes.

```
import { Router } from "express";
import { UsersController } from "./controller";
import { SharedService, UserService } from "../services";
import { AuthService } from '../services/auth.service';
import { EmailService } from '../services/email.service';
import { envs } from "../../config";
import { AuthMiddleware } from "../middlewares/auth.middleware";


Daniel Cobos, 4 months ago | 1 author (Daniel Cobos)
export class UsersRoutes {

    static get routes(): Router {
        const router = Router();
        const sharedService = new SharedService();
        const emailService = new EmailService(
            envs.MAILER_SERVICE,
            envs.MAILER_EMAIL,
            envs.MAILER_SECRET_KEY,
            envs.SEND_EMAIL
        );
        const authService = new AuthService(emailService);
        const userService = new UserService(sharedService, authService);
        const controller = new UsersController( userService );

        router.get("/",       [ AuthMiddleware.validateAdminToken ], controller.getUsers);
        router.get("/self",   [ AuthMiddleware.validateUserToken ], controller.getSelf);
        router.get("/:id",    [ AuthMiddleware.validateSelfOrAdminToken ], controller.getUserById);
        router.post("/",      [ AuthMiddleware.validateAdminToken ], controller.createUser);
        router.put("/:id",    [ AuthMiddleware.validateSelfOrAdminToken ], controller.updateUser);
        router.delete("/:id", [ AuthMiddleware.validateSelfOrSuperAdminToken ], controller.deleteUser);

        return router;
    }
}
```

4.4.4.7 Flujo de ejecución

El flujo de ejecución de una solicitud comienza en el archivo app.ts, que inicializa el servidor y configura las rutas. Cuando una petición es realizada, el enrutador redirige la solicitud al controlador adecuado. Si es necesario, se valida la entrada utilizando DTOs y luego se llama al servicio correspondiente para procesar la solicitud. Finalmente, el controlador devuelve una respuesta al cliente.

Además, la aplicación está diseñada para manejar información en tiempo real mediante MQTT. En el archivo app.ts, se inicializa un cliente MQTT que escucha los datos provenientes del broker MQTT (EMQX) y los procesa en tiempo real.

Figura 78.

Fragmento de código - App.ts.

```
import { envs } from "./config/envs";
import { MongoDatabase } from './data';
import { AppRoutes } from "./presentation/routes";
import { Server } from "./presentation/server";
import { MqttClient } from "./clients/mqtt.client";
import { MqttService } from "./presentation/services/mqtt.service";
import { WhatsappClient } from "./clients/whatsapp.client";
import { WhatsappService } from "./presentation/services/whatsapp.service";

(async() => {
    main();
})();

async function main() {
    // Initialize MongoDB
    await MongoDatabase.connect({
        mongoUrl: envs.MONGO_URL,
        dbName: envs.MONGO_DB_NAME,
    });

    // Initialize MQTT client
    const mqttClient = new MqttClient({
        url: envs.MQTT_BROKER_URL,
        clientId: envs.MQTT_CLIENT_ID,
        username: envs.MQTT_USERNAME,
        password: envs.MQTT_PASSWORD,
    });

    const mailerServiceOptions = {
        mailerService: envs.MAILER_SERVICE,
        mailerEmail: envs.MAILER_EMAIL,
        senderEmailPassword: envs.MAILER_SECRET_KEY,
        postToProvider: true,
    }

    // Initialize MQTT service
    MqttService.initialize(mqttClient, mailerServiceOptions);

    // Initialize whatsapp client
    const whatsappClient = new WhatsappClient({
        apiURL: `https://graph.facebook.com/${envs.WHATSAPP_API_VERSION}/${envs.WHATSAPP_PHONE_NUMBER_ID}/messages`,
        authToken: envs.WHATSAPP_AUTH_TOKEN,
    })

    // Initialize whatsapp service
    WhatsappService.initialize(whatsappClient);

    // Initialize express server
    const server = new Server({
        port: envs.PORT,
        routes: AppRoutes.routes,
        frontendOrigin: envs.FRONTEND_ORIGIN,
    })

    server.start();
}
```

4.4.5 Desarrollo del Frontend

El proyecto está organizado principalmente en módulos, componentes, servicios e interfaces. A continuación se detallan las capas clave y su organización.

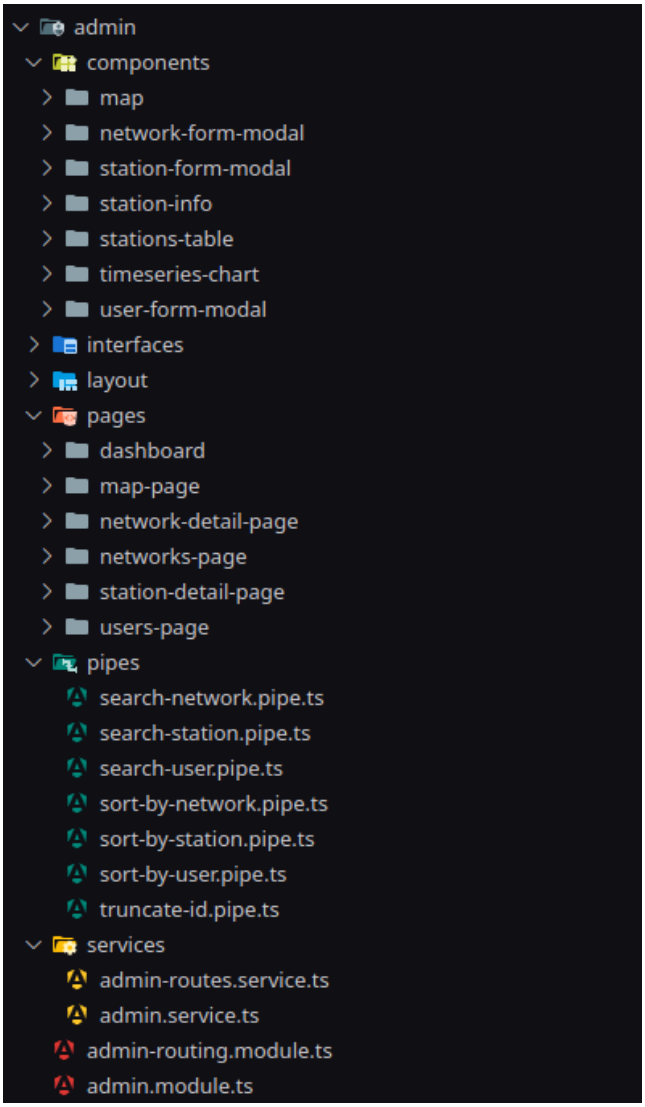
4.4.5.1 Módulos principales

4.4.5.1.1 Admin Module

Está diseñado para la administración del sistema, incluyendo componentes relacionados con la gestión de redes, estaciones y usuarios. Este módulo cuenta con formularios para redes y estaciones, gráficos de series de tiempo, tablas de estaciones, y modales de formulario. Además, dispone de servicios que permiten interactuar con las rutas y los datos de administración, y define las rutas para acceder a páginas administrativas como el dashboard, redes, detalles de estaciones, entre otros.

Figura 79.

Distribución de archivos Admin Module.



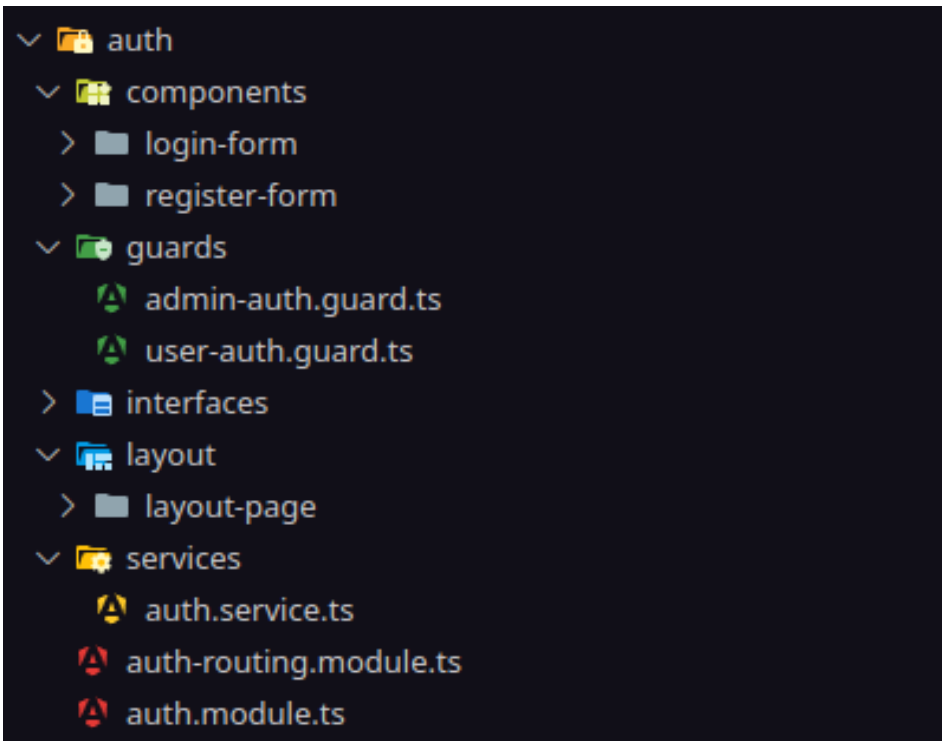
4.4.5.1.2 Auth Module

Este módulo se encarga de la autenticación y autorización del sistema. Sus componentes incluyen formularios de login y registro, mientras que sus servicios proporcionan un sistema de autenticación y protección de rutas mediante guardias (guards). Este módulo

define las rutas necesarias para las páginas de login, registro y aquellas que están protegidas por autenticación.

Figura 80.

Distribución de archivos Auth Module.

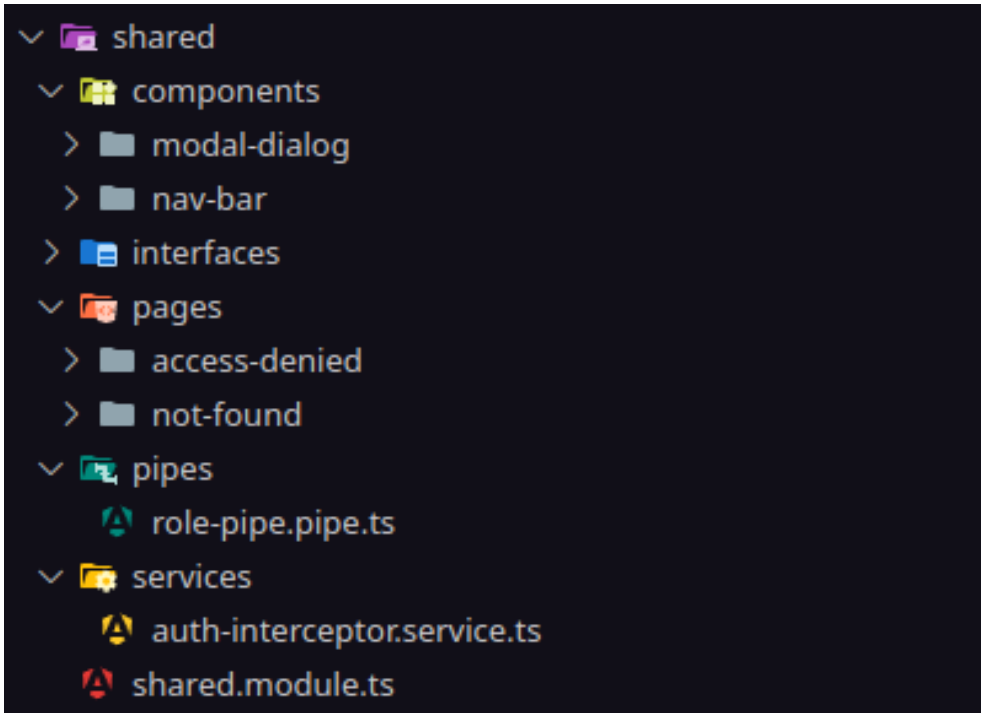


4.4.5.1.3 Shared Module

Es un módulo reutilizable que contiene componentes, interfaces, pipes y servicios que pueden ser utilizados en otras partes del proyecto. Entre sus componentes se encuentran la barra de navegación y los modales de diálogo. Además, incluye pipes para el filtrado y la ordenación de elementos, así como interceptores de autenticación que gestionan los tokens.

Figura 81.

Distribución de archivos Shared Module.

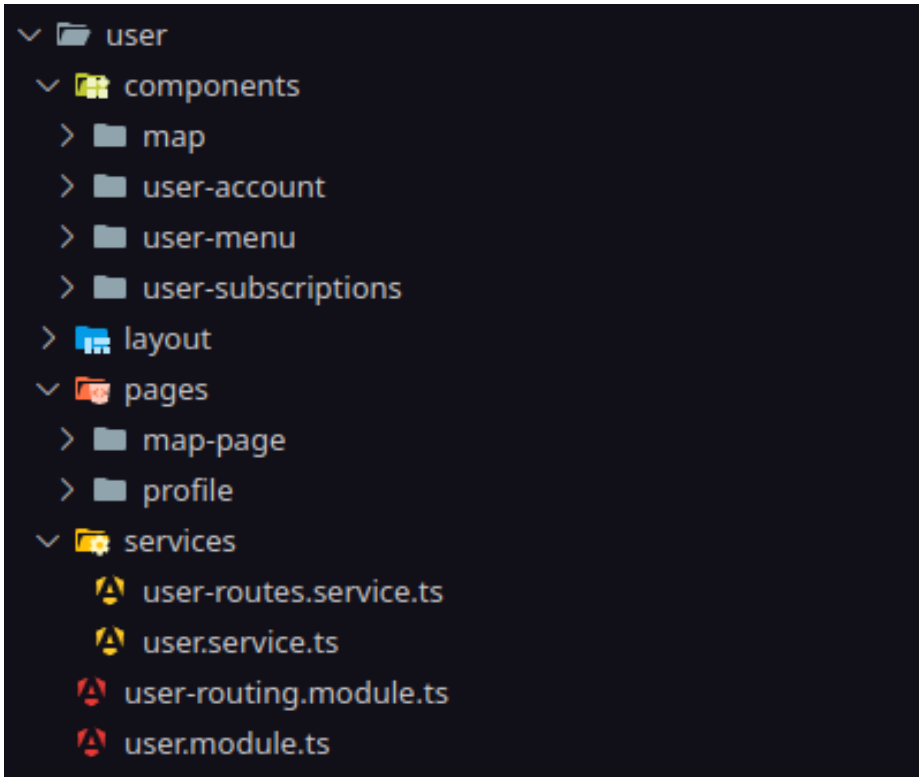


4.4.5.1.4 User Module

Este módulo está orientado a la gestión de la información del usuario y sus suscripciones. Incluye componentes para la visualización de mapas, la gestión de la cuenta de usuario, las suscripciones y el menú del usuario. Sus servicios están enfocados en la gestión de rutas y datos específicos relacionados con el usuario.

Figura 82.

Distribución de archivos User Module.



4.4.5.2 Componentes

Los componentes están organizados en subdirectorios dentro de cada módulo, y se dividen principalmente en páginas y componentes más pequeños. Por ejemplo:

- **Páginas:** Componen las vistas completas para las diferentes funcionalidades de la aplicación, como el mapa, perfil de usuario, redes, y dashboard.
- **Componentes pequeños:** Son reutilizables en diferentes páginas, como los formularios modales (por ejemplo, para agregar usuarios, redes o estaciones) y tablas de datos.

Figura 83.

Fragmento de código - Station form component.

```
@Component({
  selector: 'admin-station-form-modal',
  templateUrl: './station-form-modal.component.html',
  styleUrl: './station-form-modal.component.css',
})
export class StationFormModalComponent implements OnInit, OnDestroy {
  private destroy$ = new Subject<void>();

  @Input({ required: true })
  title!: string;

  @Input({ required: true })
  action!: 'create' | 'update';

  @Input({ required: true })
  networkId!: string;

  @Input()
  station?: Station;

  @Output()
  closeEvent = new EventEmitter<void>();

  public showForm: boolean = true;

  public dialogMessage = {
    title: 'Error',
    description: 'An error has occurred.',
  };

  public myForm: FormGroup = this.fb.group({
    name: ['', [Validators.required]],
    state: ['', [Validators.required]],
    countryCode: ['', [Validators.required]],
    latitude: ['', [Validators.required]],
    longitude: ['', [Validators.required]],
    isVisibleToUser: [true, Validators.required]
  });

  public countries: WritableSignal<Country[]> = signal([]);

  constructor(
    private adminService: AdminService,
    private fb: FormBuilder
  ) {}
```

4.4.5.3 Servicios

Los servicios se utilizan para manejar la lógica de negocio y las interacciones con el backend. Están organizados dentro de cada módulo y proporcionan métodos para acceder a las rutas específicas de la API.

Figura 84.

Fragmento de código - Admin service.

```
@Injectable({
  providedIn: 'root'
})
export class AdminService {

  private baseUrl = environments.baseUrl;

  constructor(
    private http: HttpClient,
  ) { }

  public saveToLocalStorage(key: string, value: any) {
    localStorage.setItem( key, JSON.stringify(value) );
  }

  public getFromLocalStorage(key: string) {
    return JSON.parse( localStorage.getItem(key) || 'null' );
  }

  // # Networks requests
  getNetworks( page: number = 1, limit: number = 10 ) {
    return this.http.get<NetworkResponse>(`${this.baseUrl}/api/networks?page=${page}&limit=${limit}`);
  }

  getNetworkById( id: string ) {
    return this.http.get<Network>(`${this.baseUrl}/api/networks/${id}`);
  }

  createNetwork( network: Network ) {
    return this.http.post<Network>(`${this.baseUrl}/api/networks`, network);
  }

  updateNetwork( network: Network ) {
    return this.http.put<Network>(`${this.baseUrl}/api/networks/${network.id}`, network);
  }

  deleteNetwork( id: string ) {
    return this.http.delete<Network>(`${this.baseUrl}/api/networks/${id}`);
  }

  // # Stations requests
  getStations( page: number = 1, limit: number = 10 ) {
    return this.http.get<StationResponse>(`${this.baseUrl}/api/stations?page=${page}&limit=${limit}`);
  }
```

4.4.5.4 Interfaces

Las interfaces se encuentran dentro de cada módulo y definen la estructura de los datos. Se utilizan para asegurar que los objetos cumplen con una estructura esperada, lo cual facilita la integración y manipulación de datos a lo largo del frontend.

Figura 85.

Fragmento de código - Sensor interface.

```
export interface Sensor {
  id:              string;
  name:            string;
  sensorType:      string;
  sendingInterval: number;
  stationId:       string;
  sendAlerts:      boolean;
  threshold:       Threshold;
  createdAt:       Date;
  updatedAt:       Date;
}

Daniel Cobos, 4 months ago | 1 author (Daniel Cobos)
export interface Threshold {
  yellow: number;
  orange: number;
  red:    number;
}
```

4.4.5.5 Pipes

Los pipes en el proyecto se utilizan para transformar los datos de presentación, como la búsqueda, ordenación y truncado de información, y se aplican en componentes como tablas y formularios.

Figura 86.

Fragmento de código - Search user pipe.

```
@Pipe({
  name: 'searchUser'
})
export class SearchUserPipe implements PipeTransform {

  constructor(
    private datePipe: DatePipe
  ) {}

  transform(users: User[], searchText: string): User[] {
    if (!users || !searchText) {
      return users;
    }
    searchText = searchText.toLowerCase();
    return users.filter(user =>
      user.id.toString().includes(searchText) ||
      user.name.toLowerCase().includes(searchText) ||
      user.email.toLowerCase().includes(searchText) ||
      user.role.join(',').toLowerCase().includes(searchText) ||
      (this.datePipe.transform(user.createdAt, 'MMMM d, y, h:mm:ss a') || '').toLowerCase().includes(searchText) ||
      (this.datePipe.transform(user.updatedAt, 'MMMM d, y, h:mm:ss a') || '').toLowerCase().includes(searchText)
    );
  }
}
```

4.4.6 Desarrollo del Hardware

Para el desarrollo del módulo ESP32 se utilizó el IDE PlatformIO bajo el Framework de Arduino, integrando diversas librerías necesarias para su funcionamiento. Entre estas se encuentran:

- **<WiFi.h>**, encargada de gestionar la conexión WiFi del dispositivo.
- **<PubSubClient.h>,** utilizada para establecer la comunicación mediante el protocolo MQTT con el broker configurado.
- **<ArduinoJson.h>,** para la manipulación y generación de mensajes en formato JSON.
- **<NewPing.h>,** que facilita el control del sensor ultrasónico HC-SR04 para la medición de distancia.

- **<Adafruit_GFX.h> y <Adafruit_SSD1306.h>,** utilizadas para controlar la pantalla OLED y mostrar los datos de los sensores en tiempo real.

El código desarrollado incluye la configuración de los intervalos de medición para cada sensor (distancia, flujo de agua y lluvia) y permite la actualización remota de estos parámetros mediante mensajes MQTT. Una vez finalizado el desarrollo, el código fue cargado al ESP32 mediante el puerto COM utilizando un cable microUSB.

Figura 87.

Fragmento de código - Librerías hardware.

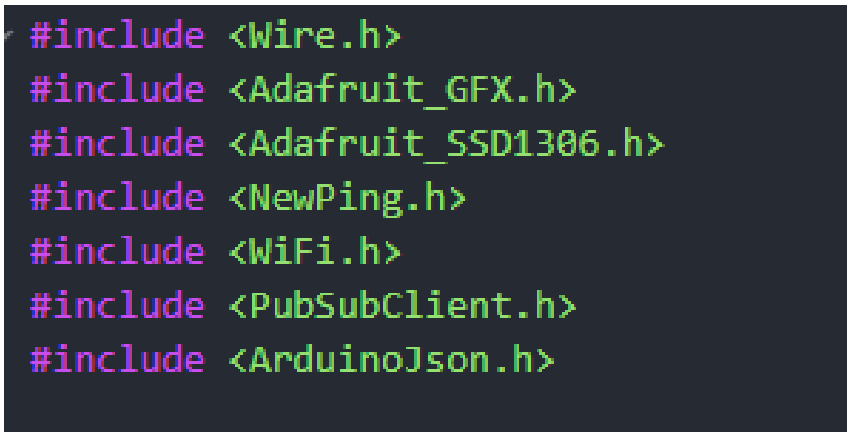


Figura 88.

Fragmento de código hardware.

```
// Definiciones del sensor HC-SR04
#define TRIGGER_PIN 5
#define ECHO_PIN 18
#define MAX_DISTANCE 200 // Máxima distancia a medir (en cm)

// Inicializa el sensor HC-SR04
Tabnine | Edit | Test | Explain | Document
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

// Definiciones para la pantalla OLED
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 32
#define OLED_RESET -1
Tabnine | Edit | Test | Explain | Document
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Definiciones para el sensor de flujo de agua
#define FLOW_SENSOR_PIN 4
volatile uint16_t pulseCount = 0;
float flowRate = 0.0;
unsigned long oldFlowTime = 0;

// Definiciones para el sensor de lluvia MH-RD
#define SENSOR_ANALOG_PIN 36 // Pin analógico

WiFiClient espClient;
Tabnine | Edit | Test | Explain | Document
PubSubClient client(espClient);

// Intervalos de tiempo para cada sensor
unsigned long distInterval = 30000;  // 5 minutos
unsigned long flowInterval = 12000;  // 2 minutos
unsigned long waterInterval = 12000; // 2 minutos

// Temporizadores para cada sensor
unsigned long lastDistTime = 0;
unsigned long lastFlowTime = 0;
unsigned long lastWaterTime = 0;
unsigned long lastDisplayTime = 0; // Para controlar la frecuencia de actualización de la pantalla
```

5 Resultados

De acuerdo con las cinco fases previamente definidas para la realización del proyecto, se llevó a cabo un proceso estructurado que incluyó la obtención, evaluación, interpretación, diseño y aplicación de la información. Cada fase permitió abordar los objetivos planteados de manera estructurada, logrando cumplir con los requerimientos funcionales y no funcionales del sistema. Los resultados obtenidos conforme a las actividades desarrolladas durante estas fases se presentan a continuación.

Tabla 14.

Indagación tecnológica y de requerimientos.

Fase 1: Indagación tecnológica y de requerimientos.	
Estudio e indagación de conceptos fundamentales para el desarrollo de un proyecto de IoT	*cumplida*
Evaluación de tecnologías específicas para la implementación del sistema de monitoreo de ríos	*cumplida*
Revisión de casos de estudio pertinentes sobre la detección de desbordamientos fluviales en entornos reales	*cumplida*
Identificación y selección de herramientas esenciales para el desarrollo del prototipo IoT	*cumplida*

A partir de las actividades desarrolladas durante la Fase 1, se logró recopilar información clave sobre tecnologías relevantes y tendencias aplicables al proyecto. Este análisis inicial permitió establecer los requerimientos fundamentales para el diseño y construcción del prototipo.

Tabla 15.

Estudio y planificación de la arquitectura.

Fase 2: Estudio y planificación de la arquitectura.	
Evaluación de la viabilidad y alcance del proyecto	*cumplida*
Planificación y elaboración del diseño de la plataforma de software específica para el sistema de monitoreo de ríos	*cumplida*
Selección de los indicadores más importantes para evaluar y dar seguimiento al proyecto de alerta temprana	*cumplida*
Definición detallada de la arquitectura necesaria para el prototipo IoT, abarcando dispositivos, conectividad, infraestructura y aplicación	*cumplida*

Durante la Fase 2, se llevó a cabo la planificación de la arquitectura del prototipo IoT, definiendo la estructura necesaria para su desarrollo e integración efectiva.

Tabla 16.

Creación del prototipo y de la plataforma.

Fase 3: Creación del prototipo y de la plataforma.	
Diseño y desarrollo de la interfaz de usuario para el prototipo de alertas tempranas de desbordamientos de ríos	*cumplida*
Integración de los sensores y componentes de hardware en el prototipo IoT según el diseño establecido	*cumplida*
Ensamblaje del prototipo a escala siguiendo las especificaciones previas y los requisitos definidos	*cumplida*
Configuración de la red de conectividad de los dispositivos y su vinculación con la plataforma web de alertas tempranas	*cumplida*

Con el cumplimiento de la fase 3, se desarrolló el prototipo y la plataforma tecnológica, integrando los sensores y componentes de hardware con el diseño establecido, asegurando su conectividad con la red y la plataforma web para una operación eficiente.

Figura 89.

Estaciones funcionales.

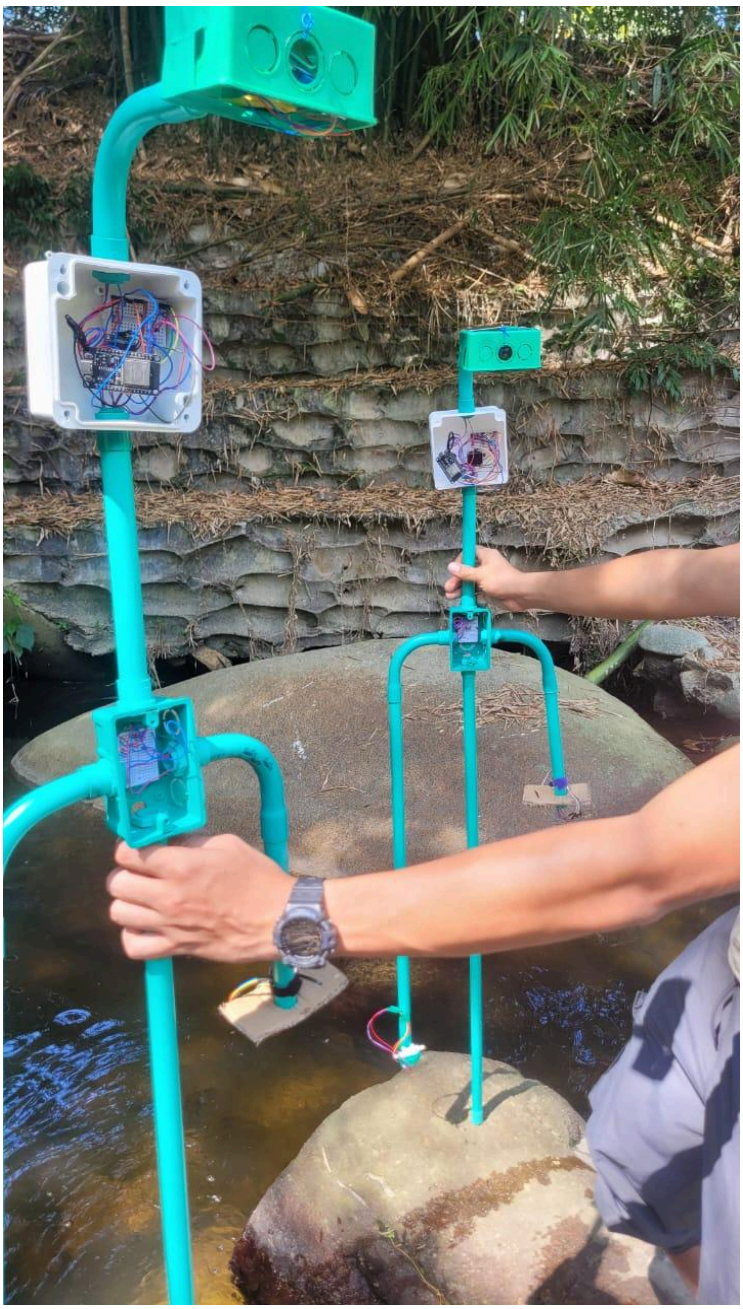


Figura 90.

Componentes electrónicos - Estación funcional.

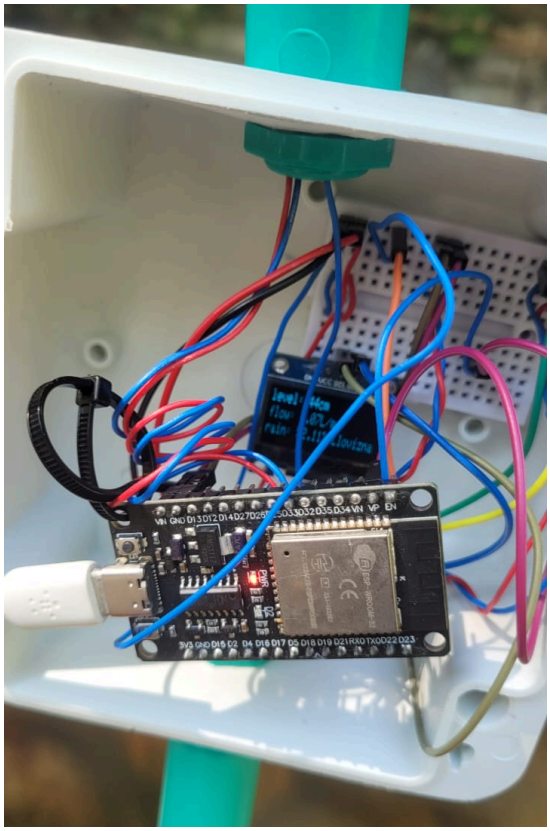


Tabla 17.

Validación y verificación del prototipo.

Fase 4: Validación y verificación del prototipo.	
Implementación del almacenamiento de datos en la nube y su interacción con el prototipo	*cumplida*
Validación de la coherencia y la ejecución del sistema de alertas tempranas	*cumplida*
Validación del prototipo a través de pruebas funcionales, unitarias, de rendimiento y otras pruebas relevantes	*cumplida*
Análisis de los resultados de las pruebas realizadas para identificar posibles problemas y ajustes necesarios	*cumplida*

Se realizó la validación y verificación del prototipo. Además, se analizaron los resultados para identificar posibles mejoras y realizar los ajustes necesarios.

Figura 91.

Validación de datos del prototipo estación 1 en campo.

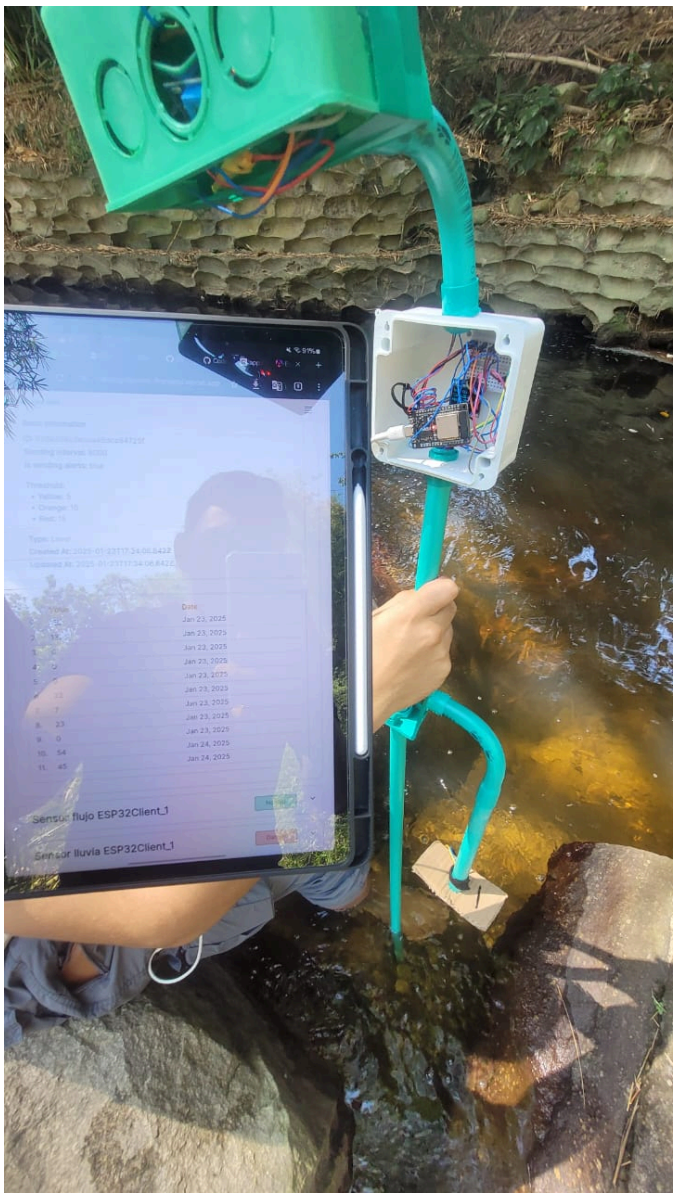


Figura 92.

Validación de datos del prototipo estación 2 en campo.

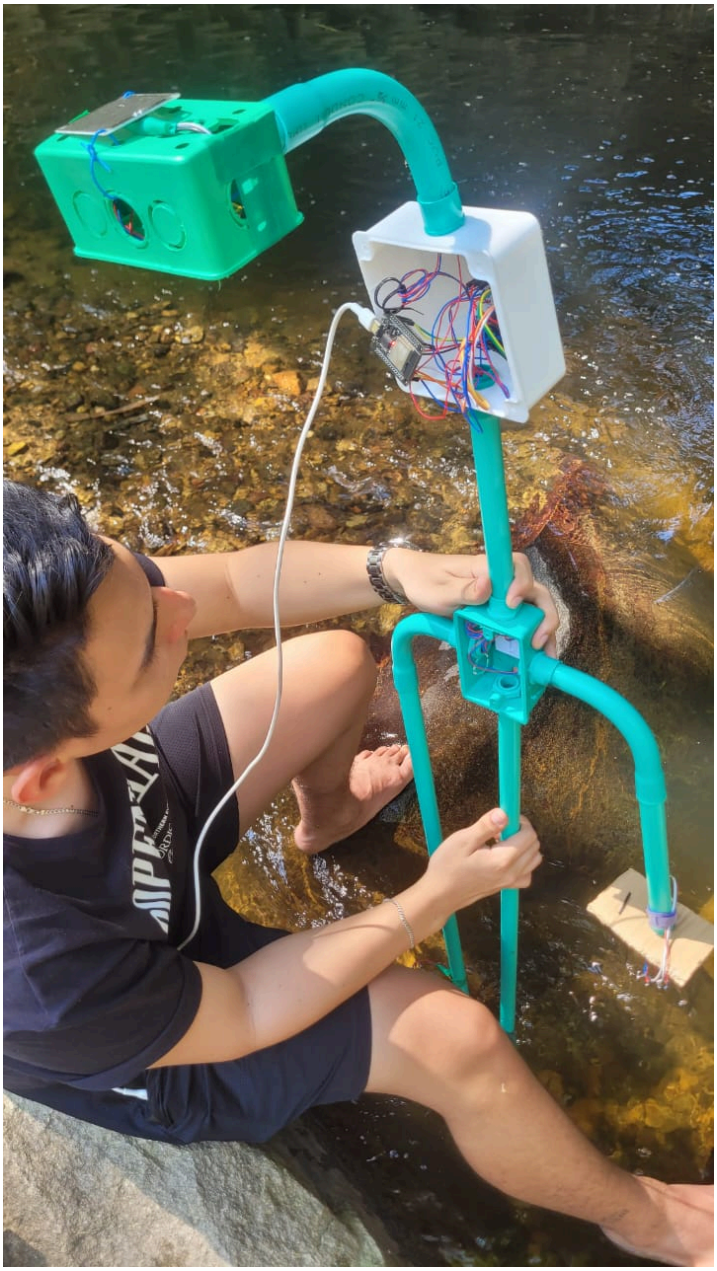
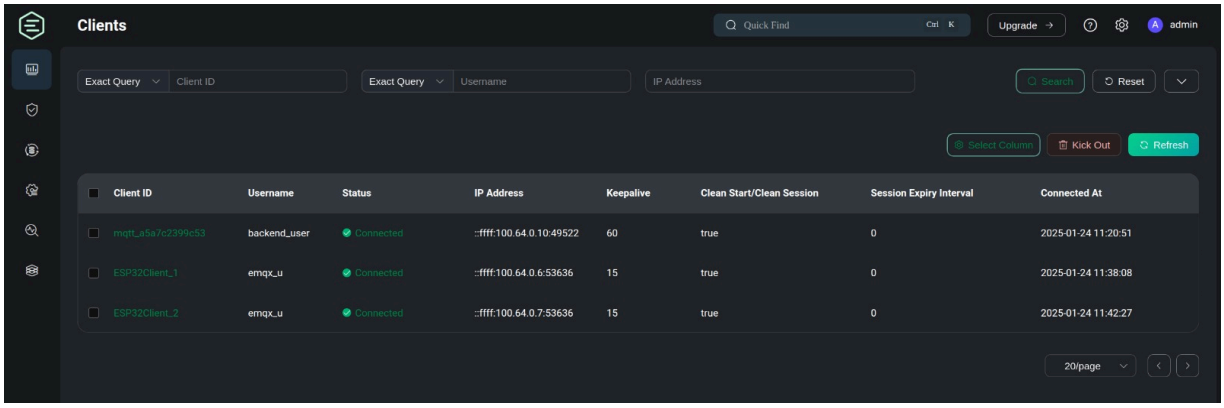


Figura 93.

Prueba de envío de datos al sistema.

Figura 94.

Dashboard EMQX - Clientes conectados.



Nota. Tomado de EMQX Dashboard. (s.f.).

https://caddy-production-dc71.up.railway.app/#/dashboard/overview

Tabla 18.

Presentación del prototipo definitivo.

Fase 5: Presentación del prototipo definitivo.	
Presentación del prototipo definitivo con los ajustes necesarios	*cumplida*

5.1 Evaluación de Requerimientos funcionales y no funcionales

Tabla 19.

Inspección de los requerimientos funcionales y no funcionales.

Requerimiento	**Entrada**	**Salida esperada**	**Criterio de aceptación**	**Cumplido / No cumplido**
RF1	*Datos captados por sensores IoT (nivel de agua, velocidad del caudal, lluvia).*	*Valores procesados y almacenados en la base de datos.*	*El sistema debe recopilar datos en tiempo real de al menos tres variables críticas y procesarlos en menos de 500 segundos.*	*Cumplido*
RF2	*Datos procesados de los sensores en la base de datos.*	*Interfaz gráfica con visualización de datos.*	*Los usuarios deben poder visualizar todas las variables registradas en menos de 500 segundos.*	*Cumplido*
RF3	*Valores procesados y almacenados que excedan los umbrales predefinidos.*	*Notificaciones automáticas enviadas a los usuarios registrados.*	*Las alertas deben ser enviadas automáticamente en menos de 500 segundos tras la detección de eventos críticos.*	*Cumplido*
RF4	*Credenciales de acceso de administradores.*	*Acceso al panel de administración y opciones de configuración.*	*Los administradores deben poder iniciar sesión y acceder al panel de manera segura previniendo accesos no autorizados*	*Cumplido*

RF5	*Datos básicos del usuario.*	*Usuario registrado.*	*El sistema debe validar que el correo electrónico sea único y que todos los campos estén completos.*	*Cumplido*
RF6	*Datos de usuario autenticado.*	*Mapa con estaciones suscritas.*	*Los usuarios deben gestionar sus suscripciones y visualizar las estaciones suscritas de manera intuitiva.*	*Cumplido*
RF7	*Datos de redes y estaciones.*	*Estaciones gestionadas.*	*El sistema debe reflejar cambios en la gestión de redes y estaciones en tiempo real.*	*Cumplido*
RF8	*Datos de los administradores.*	*Administradores gestionados.*	*Los super administradores deben realizar operaciones CRUD sobre administradores y usuarios con permisos aplicados inmediatamente.*	*Cumplido*
RNF1	*Cambios solicitados por desarrolladores o usuarios.*	*Sistema actualizado y documentado.*	*Los desarrolladores deben implementar cambios sin introducir errores, y la documentación debe reflejar actualizaciones.*	*Cumplido*
RNF2	*Cambios en la carga de trabajo o en el número de usuarios.*	*Sistema ajustado para mantener el rendimiento óptimo.*	*El rendimiento del sistema no debe degradarse durante picos de demanda.*	*Cumplido*
RNF3	*Interacción del usuario con la interfaz web.*	*Respuesta inmediata a las solicitudes del usuario.*	*El sistema debe procesar las solicitudes del usuario en menos de 60 segundos.*	*Cumplido*
RNF4	*Datos procesados desde RF1 y generados por sensores IoT.*	*Datos almacenados en la nube disponibles para consulta y análisis.*	*Los datos deben ser accesibles desde la plataforma web en menos de 500 segundos.*	*Cumplido*

6 Conclusiones

El proyecto "Diseño de un prototipo basado en una plataforma IoT para la detección de alertas tempranas ante el desbordamiento de ríos" ha demostrado la viabilidad de integrar tecnologías IoT en la mitigación de riesgos asociados a inundaciones. A través del diseño e implementación de un prototipo funcional, se logró el monitoreo en tiempo real de variables hidrometeorológicas críticas, permitiendo la generación oportuna de alertas para la toma de decisiones preventivas.

Uno de los principales logros del proyecto fue la selección y configuración de sensores capaces de medir eficazmente el nivel del agua, el caudal y la presencia de precipitaciones, datos fundamentales para la detección temprana de eventos de desbordamiento. Asimismo, la infraestructura de comunicación basada en el protocolo MQTT y la computación en la nube permitieron la transmisión confiable y eficiente de la información captada por el sistema, garantizando su disponibilidad para análisis y respuesta inmediata.

En cuanto a la estructura de la base de datos, su diseño resultó adecuado para el almacenamiento y procesamiento de la información capturada por los sensores. Se logró una gestión eficiente de los datos, asegurando su integridad y disponibilidad en la nube. La integración con la plataforma web se llevó a cabo sin inconvenientes, permitiendo a los usuarios recibir alertas automáticas ante eventos críticos.

La arquitectura del sistema, basada en una combinación de tecnologías en la nube y dispositivos IoT, demostró ser eficiente y escalable. Durante las pruebas experimentales realizadas en un entorno real en el municipio de Floridablanca, Santander, se validó la capacidad del sistema para operar de manera estable, identificar cambios abruptos en las

condiciones hidrometeorológicas y emitir alertas con la rapidez necesaria para la toma de decisiones oportunas. La validación en campo confirmó la robustez del diseño y su potencial aplicabilidad en otros entornos vulnerables a desbordamientos.

En relación con los casos de uso y objetivos planteados, el sistema cumplió con los requerimientos funcionales y no funcionales establecidos. Se verificó que los usuarios pudieran visualizar las mediciones en tiempo real, recibir notificaciones automáticas ante eventos críticos y gestionar sus estaciones de monitoreo de manera intuitiva. Además, se implementaron medidas de seguridad adecuadas para el acceso y manipulación de datos, garantizando la confiabilidad del sistema.

El proyecto también ha evidenciado la importancia de la escalabilidad y sostenibilidad en el diseño de sistemas de alerta temprana. Se identificaron oportunidades de mejora, como la integración de sensores de mayor precisión, la optimización del consumo energético mediante energía solar y el desarrollo de algoritmos avanzados de predicción basados en inteligencia artificial. Estas mejoras permitirían aumentar la fiabilidad del sistema y su aplicabilidad en contextos más amplios.

Finalmente, el impacto potencial de este prototipo se extiende más allá de la prevención de desbordamientos, ya que su estructura modular podría adaptarse a otros escenarios de monitoreo ambiental, como la detección de incendios forestales o la gestión de recursos hídricos. En este sentido, el proyecto no solo representa un aporte tecnológico en la reducción de desastres naturales, sino que también sienta las bases para futuras investigaciones y desarrollos en la gestión del riesgo y la protección del medio ambiente.

7 Recomendaciones

Para mejorar la conectividad de las estaciones, se recomienda evaluar alternativas de comunicación que complementen la red WiFi. Una opción viable es la integración de módulos de SIM Card que permitan enviar mensajes SMS en caso de fallas en la conectividad WiFi, asegurando la transmisión de alertas críticas. Adicionalmente, la implementación de redes de transmisión como LoRa podría ofrecer una mayor cobertura y confiabilidad en áreas remotas o de difícil acceso.

En cuanto a la precisión y robustez del sistema, se sugiere considerar la incorporación de sensores de mayor calidad para variables críticas como el nivel del agua y la velocidad del caudal. Esto permitiría obtener datos más exactos y minimizar posibles errores en la detección de alertas tempranas. Asimismo, sería beneficioso explorar la utilización de algoritmos avanzados de inteligencia artificial para el análisis predictivo de datos, lo que podría aumentar significativamente la capacidad del sistema para anticipar desbordamientos.

Desde la perspectiva de la plataforma web, sería recomendable implementar funcionalidades adicionales que mejoren la experiencia del usuario. Entre estas, se destacan la personalización de alertas y configuraciones por parte de los usuarios, y la posibilidad de que los administradores gestionen estaciones y dispositivos desde una aplicación móvil dedicada. Esto facilitaría la instalación y configuración de las estaciones, especialmente en situaciones donde el acceso a la plataforma desde dispositivos móviles sea necesario.

Finalmente, se recomienda realizar un monitoreo continuo del prototipo en entornos reales para evaluar su desempeño a largo plazo. Esto permitirá identificar áreas de mejora, optimizar recursos y explorar la posibilidad de ampliar el sistema a otras regiones vulnerables

en Colombia, contribuyendo así a una gestión más eficiente de los riesgos asociados al desbordamiento de ríos.

Referencias

¿Qué es una API de RESTful? - Explicación de API de RESTful - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/restful-api/

¿Qué es una aplicación web? - Explicación de las aplicaciones web - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/web-application/

¿Qué es una base de datos? - Explicación de las bases de datos en la nube - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/database/

Angular. (n.d.). Angular. https://angular.dev/overview

Arduino UNO Rev3 with Long Pins | Arduino Documentation. (n.d.). https://docs.arduino.cc/retired/boards/arduino-uno-rev3-with-long-pins/

Arua, U. F. (2024, June 18). The role of the CI/CD pipeline in cloud computing. *Civo.com*. https://www.civo.com/blog/the-role-of-the-ci-cd-pipeline-in-cloud-computing

auth0.com. (n.d.). *JWT.IO - JSON Web Tokens Introduction*. JSON Web Tokens - jwt.io. https://jwt.io/introduction

Ávila, Á., Guerrero, F. C., Escobar, Y. C., & Justino, F. (2019). Recent precipitation trends and floods in the Colombian Andes. *Water, 11*(2), 379. https://doi.org/10.3390/w11020379

Codecademy. (n.d.). *What is REST?* Codecademy. https://www.codecademy.com/article/what-is-rest

Codecademy. (n.d.-a). *What is CRUD?* Codecademy. https://www.codecademy.com/article/what-is-crud

Computing, S. (2024, 16 julio). *Edge computing vs. fog computing vs. cloud computing*. Scale Computing.

https://www.scalecomputing.com/resources/edge-computing-vs-fog-computing-vs-cloud-computing

Conceptos básicos sobre bases de datos - Soporte técnico de Microsoft. (n.d.). https://support.microsoft.com/es-es/topic/conceptos-b%C3%A1sicos-sobre-bases-de-datos-a849ac16-07c7-4a31-9948-3c8c94a7c204#:~:text=Una%20base%20de%20datos%20es,productos%2C%20pedidos%20u%20otras%20cosas.

Contenedores de Docker | ¿Qué es Docker? | AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/es/docker/

Contributor, T. (2019, March 1). *clean architecture*. WhatIs. https://www.techtarget.com/whatis/definition/clean-architecture

Cytron Technologies. (2013). *HC-SR04 Ultrasonic Sensor: User's Manual* (Version 1.0). Cytron Technologies Sdn. Bhd. https://web.eece.maine.edu/~zhu/book/lab/HC-SR04%20User%20Manual.pdf

Docker: Accelerated Container Application Development. (2024, July 8). Docker. https://www.docker.com/

EMQ Technologies Inc. (n.d.). *EMQX: the #1 MQTT platform for IoT, IIoT and connected cars*. www.emqx.com. https://www.emqx.com/en

Express - Node.js web application framework. (n.d.). https://expressjs.com/

Fitzgibbons, L. (2018, June 6). *DRY principle*. WhatIs. https://www.techtarget.com/whatis/definition/DRY-principle

Fleisch, E. (2010). *What is the internet of things? When things add value* (Auto-ID Labs White Paper WP-BIZAPP-053). Auto-ID Lab St. Gallen, Switzerland. https://cocoa.ethz.ch/downloads/2014/06/None_AUTOIDLABS-WP-BIZAPP-53.pdf

Front End vs Back End - Difference Between Application Development - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/compare/the-difference-between-frontend-and-backend/#:~:text=The%20backend%20is%20the%20data,application%20data%20for%20your%20users.

Gupta, J. N. D., & Sharma, S. K. (2002). Globalization and information management strategy. In M. Khosrow-Pour (Ed.), Encyclopedia of information systems (pp. 475-487). https://www.sciencedirect.com/referencework/9780122272400/encyclopedia-of-information-systems

Hamdaqa, M., & Tahvildari, L. (2012). Cloud Computing Uncovered: A Research Landscape. En *Advances in computers* (pp. 41-85). https://doi.org/10.1016/b978-0-12-396535-6.00002-8

Hashemi-Pour, C., & Churchville, F. (2024, April 30). *user interface (UI)*. App Architecture. https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI

Ibm. (2024, August 19). API. *What is an API (application programming interface)?* https://www.ibm.com/topics/api

Ibm. (2025, February 11). ¿Qué es una API REST? *IBM*. https://www.ibm.com/mx-es/topics/rest-apis

International Telecommunication Union. (2005). *ITU internet reports 2005: The internet of things* (7th ed.). https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf

Kochilakis, G., Poursanidis, D., Chrysoulakis, N., Varella, V., Kotroni, V., Eftychidis, G., Lagouvardos, K., Papathanasiou, C., Karavokyros, G., Aivazoglou, M., Makropoulos, C., & Mimikou, M. (2016). A web based DSS for the management of floods and

wildfires (FLIRE) in urban and periurban areas. *Environmental Modelling & Software*, *86*, 111-115. https://doi.org/10.1016/j.envsoft.2016.09.016

Krichen, M., Abdalzaher, M. S., Elwekeil, M., & Fouda, M. M. (2023). Managing natural disasters: An analysis of technological advancements, opportunities, and challenges. *Internet Of Things And Cyber-Physical Systems*, *4*, 99-109. https://doi.org/10.1016/j.iotcps.2023.09.002

López, D., & Maya, E. (2017). *Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones* Web. Séptima Conferencia de Directores de Tecnología de Información, TICAL. http://138.59.13.30/bitstream/10786/1277/1/93%20Arquitectura%20de%20Software%20basada%20en%20Microservicios%20para%20Desarrollo%20de%20Aplicaciones%20Web.pdf

Loza Peralta, C. V. (2020). *Arquitectura de Software Basada en Microservicios para el Uso en Dispositivos de Internet de las Cosas* (Tesis de pregrado). Universidad Nacional de San Agustín, Arequipa, Perú. https://repositorio.ulasalle.edu.pe/handle/20.500.12953/100

Ltd, R. P. (n.d.). *Buy a Raspberry Pi pico – Raspberry Pi*. Raspberry Pi. https://www.raspberrypi.com/products/raspberry-pi-pico/

Mamani Rodríguez, Z., Del Pino Rodríguez, L., & Gonzales Suarez, J. C. (2020). *Arquitectura basada en Microservicios y DevOps para una ingeniería de software continúa*. *Industrial Data*, 23(2), 141-149. Universidad Nacional Mayor de San Marcos. https://doi.org/10.15381/idata.v23i2.17278

Mattos, T. S., Oliveira, P. T. S., De Souza Bruno, L., Carvalho, G. A., Pereira, R. B., Crivellaro, L. L., Lucas, M. C., & Roy, T. (2022). Towards reducing flood risk

disasters in a tropical urban basin by the development of flood alert web application. *Environmental Modelling & Software*, *151*, 105367. https://doi.org/10.1016/j.envsoft.2022.105367

Mohanan, V. (2024, March 12). *DOIT ESP32 DevKit V1 Wi-Fi Development Board - Pinout Diagram & Arduino Reference - CIRCUITSTATE*. CIRCUITSTATE Electronics. https://www.circuitstate.com/pinouts/doit-esp32-devkit-v1-wifi-development-board-pinout-diagram-and-reference/

Mora, S. L. (2022, October 4). *¿Qué son las Single-Page Application (SPA)? El desarrollo elegido por Gmail y LinkedIn*. DIGITAL55. https://digital55.com/blog/que-son-single-page-application-spa-desarrollo-elegido-por-gmail-linkedin/

NestJS - A progressive Node.js framework. (n.d.). NestJS - a Progressive Node.js Framework. https://nestjs.com/

Next.Js by Vercel - the React framework. (n.d.). https://nextjs.org/

PostgreSQL: about. (n.d.). The PostgreSQL Global Development Group. https://www.postgresql.org/about/

Railway. (n.d.). Railway. https://railway.app/

React. (n.d.). https://react.dev/

Responsive design - Learn web development | MDN. (2024, July 25). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design

RXJS. (n.d.). https://rxjs.dev/

SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. (2024, December 19). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Glossary/SPA

Spring boot. (n.d.). Spring Boot. https://spring.io/projects/spring-boot

Sufiyan, T. (2024, July 9). *What is Node.js? A complete guide for developers*. Simplilearn.com. https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs

Superior, E. F. P. (n.d.). *Aplicaciones web: qué son, tipos y ventajas*. ESIC. https://www.esic.edu/rethink/tecnologia/que-son-las-aplicaciones-web-c

The WebSocket API (WebSockets) - Web APIs | MDN. (2024, September 16). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

United Nations International Strategy for Disaster Reduction. (2012). *Terminology*. Geneva: UNISDR. http://www.unisdr.org/we/inform/terminology

Vue.js. (n.d.). The Progressive JavaScript Framework | Vue.js. https://vuejs.org/

What are Microservices? | AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/microservices/

What is a CI/CD pipeline? (n.d.). https://www.redhat.com/en/topics/devops/what-cicd-pipeline

What is a Container? | Docker. (n.d.). Docker. https://www.docker.com/resources/what-container/

What is a Daemon? How Does Lenovo's Daemon Technology Work? | Lenovo US. (2023, May 28). https://www.lenovo.com/us/en/glossary/what-is-a-daemon/

What is a Framework? - Framework in Programming and Engineering Explained - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/framework/

What is a REST API? (n.d.). https://www.redhat.com/en/topics/api/what-is-a-rest-api

What is a webhook? (n.d.). https://www.redhat.com/en/topics/automation/what-is-a-webhook

what-is-aws. (n.d.-b). [Video]. Amazon Web Services, Inc. https://aws.amazon.com/what-is-aws/

What is Heroku | Heroku. (n.d.). https://www.heroku.com/what

What is HTTP? | Cloudflare. (n.d.). https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/

What is HTTPS? | Cloudflare. (n.d.). https://www.cloudflare.com/learning/ssl/what-is-https/

What is Java? - Java Programming Language Explained - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/java/

What is JavaScript? - Learn web development | MDN. (2024, August 3). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

What is MQTT? - MQTT Protocol Explained - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/mqtt/

What is SQL? - Structured Query Language (SQL) Explained - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/sql/

What is User Experience (UX) Design? (2024, September 3). The Interaction Design Foundation. https://www.interaction-design.org/literature/topics/ux-design

Which NoSQL database is right for you? (n.d.). [Video]. Amazon Web Services, Inc. https://aws.amazon.com/nosql/

Wigmore, I. (2016, December 15). *single-page application (SPA)*. WhatIs. https://www.techtarget.com/whatis/definition/single-page-application-SPA#:~:text=%20single-page%20application%20

Wikipedia contributors. (2001, November 9). *Programming language*. Wikipedia. https://en.wikipedia.org/wiki/Programming_language

Wikipedia contributors. (2001b, November 10). *C (programming language)*. Wikipedia. https://en.wikipedia.org/wiki/C_(programming_language)

Wikipedia contributors. (2024, September 23). *Microsoft Azure*. Wikipedia. https://en.wikipedia.org/wiki/Microsoft_Azure

Wikipedia contributors. (2024a, July 15). *Vercel*. Wikipedia. https://en.wikipedia.org/wiki/Vercel

Wikipedia contributors. (2024b, August 5). *MongoDB*. Wikipedia. https://en.wikipedia.org/wiki/MongoDB

Wikipedia contributors. (2024b, September 25). *Google Cloud Platform*. Wikipedia. https://en.wikipedia.org/wiki/Google_Cloud_Platform

Wikipedia contributors. (2024c, August 23). *General-purpose input/output*. Wikipedia. https://en.wikipedia.org/wiki/General-purpose_input/output

Wikipedia contributors. (2024c, September 17). *Front-end web development*. Wikipedia. https://en.wikipedia.org/wiki/Front-end_web_development

Wikipedia contributors. (2024c, September 9). *TypeScript*. Wikipedia. https://en.wikipedia.org/wiki/TypeScript

Wikipedia contributors. (2024e, September 23). *C* + +. Wikipedia. https://en.wikipedia.org/wiki/C%2B%2B

World Meteorological Organization. (2023). *State of the global climate 2022* (WMO-No. 1316). Geneva. https://library.wmo.int/idurl/4/66214

Yasar, K. (2024, May 14). *Docker image*. IT Operations. https://www.techtarget.com/searchitoperations/definition/Docker-image

"Docker Engine." (2024, November 26). Docker Documentation. https://docs.docker.com/engine/

"Dockerfile reference." (2024, September 10). Docker Documentation. https://docs.docker.com/reference/dockerfile/#:~:text=A%20Dockerfile%20is%20a%20text,line%20to%20assemble%20an%20image.

"Part 7: Use docker compose." (2024, September 3). Docker Documentation. https://docs.docker.com/get-started/workshop/08_using_compose/#:~:text=Docker%20Compose%20is%20a%20tool,or%20tear%20it%20all%20down.

Apéndices

Apéndice A

Tabla de datos recolectados realizando la validación y verificación del prototipo.

Reading id	Reading value	Reading createdAt	Sensor name	sensor type	Sensor threshold yellow	Sensor threshold orange	Sensor threshold red	Sensor sending interval	Sensor station ID
6792d2bcc056974b524d66b2	132	23/01/2025 18:37	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6792d2dac056974b524d66c9	132	23/01/2025 18:38	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6792d352c056974b524d673a	22	23/01/2025 18:40	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6792d370c056974b524d6756	7	23/01/2025 18:40	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6792d38ec056974b524d676d	23	23/01/2025 18:41	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6793ab85c0ebc24043547079	54	24/01/2025 10:02	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6793aba3c0ebc240435470a1	45	24/01/2025 10:02	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6793abc1c0ebc240435470da	43	24/01/2025 10:03	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6793abdfc0ebc24043547100	44	24/01/2025 10:03	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6793abfdc0ebc240435471d	43	24/01/2025 10:04	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6793c1f19f3e2c463742e797	70	24/01/2025 11:38	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6793c22d9f3e2c463742e7ac	154	24/01/2025 11:39	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
6793c24b9f3e2c463742e7c3	154	24/01/2025 11:39	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
67c53b3631df9097c1a91e0f	104	3/03/2025 0:16	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5
67c53bae31df9097c1a91eb8	63	3/03/2025 0:18	Sensor ultrasonido ESP32Client_1	level	30	20	10	8000	66455fd1903474beccdb69b5

6792d2dac0569 74b524d66ce	0	23/01/2025 18:38	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d2e6c0569 74b524d66d5	0	23/01/2025 18:38	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d2f2c05697 4b524d66dd	0	23/01/2025 18:38	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d2fec05697 4b524d66ec	0	23/01/2025 18:38	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d30ac0569 74b524d66f1	0	23/01/2025 18:38	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d316c0569 74b524d6700	0	23/01/2025 18:39	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d322c0569 74b524d6705	0	23/01/2025 18:39	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d32ec0569 74b524d670d	0	23/01/2025 18:39	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d33ac0569 74b524d672a	0	23/01/2025 18:39	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d346c0569 74b524d6732	0	23/01/2025 18:39	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d352c0569 74b524d6741	0	23/01/2025 18:40	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d35ec0569 74b524d6746	0	23/01/2025 18:40	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d36ac0569 74b524d674e	0	23/01/2025 18:40	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d376c0569 74b524d675a	0	23/01/2025 18:40	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d382c0569 74b524d6765	0	23/01/2025 18:40	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5
6792d38ec0569 74b524d6774	0	23/01/2025 18:41	Sensor flujo ESP32Client_1	flow	5	10	15	8000	66455fd1903474b eccdb69b5

67928110c0569 74b524d63d3	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
67928112c0569 74b524d63d8	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
67928115c0569 74b524d63dd	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
67928117c0569 74b524d63e2	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
67928119c0569 74b524d63e9	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
6792811cc0569 74b524d63f2	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
6792811ec0569 74b524d63fa	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
67928121c0569 74b524d6404	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
67928123c0569 74b524d640a	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
67928125c0569 74b524d6414	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
67928128c0569 74b524d641a	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
6792812ac0569 74b524d6422	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
6792812dc0569 74b524d642a	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
6792812fc05697 4b524d6434	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
67928132c0569 74b524d643a	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5
67928134c0569 74b524d6444	0	23/01/2025 12:49	Sensor lluvia ESP32Client_1	rain	5	10	15	8000	66455fd1903474b eccdb69b5

6793acffc0ebc24043547158	53	24/01/2025 10:08	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
6793ad1dc0ebc24043547175	45	24/01/2025 10:09	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
6793ad3bc0ebc2404354719d	47	24/01/2025 10:09	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
6793ad59c0ebc240435471d4	46	24/01/2025 10:10	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
6793ad77c0ebc240435471fc	46	24/01/2025 10:10	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
6793adccc0ebc2404354721b	50	24/01/2025 10:12	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
6793adebc0ebc24043547247	47	24/01/2025 10:12	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
6793ae08c0ebc2404354726a	46	24/01/2025 10:13	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
67c5369a9423d0549357c174	6	2/03/2025 23:56	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
67c536b89423d0549357c1a1	5	2/03/2025 23:57	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
67c536f49423d0549357c1d1	19	2/03/2025 23:58	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
67c537c69423d0549357c29e	145	3/03/2025 0:01	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
67c5396c31df9097c1a91cfb	129	3/03/2025 0:09	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
67c5398a31df9097c1a91d20	128	3/03/2025 0:09	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
67c539a831df9097c1a91d3c	128	3/03/2025 0:10	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677
67c539c631df9097c1a91d50	129	3/03/2025 0:10	Sensor ultrasonido ESP32Client_2	level	30	20	10	8000	aabbccddee11223344556677

6793ad1ec0ebc2404354717f	0	24/01/2025 10:09	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ad29c0ebc24043547187	0	24/01/2025 10:09	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ad35c0ebc24043547192	0	24/01/2025 10:09	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ad41c0ebc240435471a4	0	24/01/2025 10:09	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ad4dc0ebc240435471af	0	24/01/2025 10:10	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ad59c0ebc240435471de	0	24/01/2025 10:10	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ad65c0ebc240435471e6	0	24/01/2025 10:10	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ad71c0ebc240435471f1	0	24/01/2025 10:10	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ad7dc0ebc24043547203	0	24/01/2025 10:10	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ad89c0ebc2404354720e	0	24/01/2025 10:11	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793adccc0ebc2404354721d	2069197083	24/01/2025 10:12	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793add8c0ebc2404354722e	0	24/01/2025 10:12	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ade4c0ebc24043547239	1704961777	24/01/2025 10:12	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793adf0c0ebc2404354724e	9457209587	24/01/2025 10:12	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793adfcc0ebc2404354725c	8524808884	24/01/2025 10:13	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677
6793ae09c0ebc24043547274	0	24/01/2025 10:13	Sensor flujo ESP32Client_2	flow	5	10	15	8000	aabbccddee11223344556677

67928125c056974b524d6412	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
67928128c056974b524d641c	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
6792812ac056974b524d6424	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
6792812dc056974b524d642c	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
6792812fc056974b524d6432	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
67928132c056974b524d643c	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
67928134c056974b524d6442	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
67928137c056974b524d644a	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
67928139c056974b524d6452	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
6792813cc056974b524d645c	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
6792813ec056974b524d6462	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
67928140c056974b524d646c	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
67928143c056974b524d6474	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
67928145c056974b524d647a	9919413757	23/01/2025 12:49	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
67928148c056974b524d6482	9919413757	23/01/2025 12:50	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677
6792814ac056974b524d648c	9919413757	23/01/2025 12:50	Sensor lluvia ESP32Client_2	rain	5	10	15	8000	aabbccddee11223344556677