UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE CUENCA

CARRERA DE TELECOMUNICACIONES

DESARROLLO DE UN PROTOTIPO IOT PARA LA DETECCIÓN TEMPRANA DE INCENDIOS FORESTALES

Trabajo de titulación previo a la obtención del título de Ingeniero en Telecomunicaciones

AUTORES: JUAN DIEGO ARMIJOS GOERCKE
CRISTHIAN MARCELO GUERRERO ABAD

TUTOR: ING. JUAN PABLO BERMEO, MGST.

Cuenca – Ecuador
2024

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Juan Diego Armijos Goercke con documento de identificación N° 0106908734 y Cristhian Marcelo Guerrero Abad con documento de identificación N° 0105168074; manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Cuenca, 2 de agosto de 2024

Atentamente,

Juan Diego Armijos Goercke

0106908734

Cristhian Marcelo Guerrero Abad

0105168074

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, Juan Diego Armijos Goercke con documento de identificación N° 0106908734 y Cristhian Marcelo Guerrero Abad con documento de identificación N° 0105168074, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico: "Desarrollo de un prototipo IoT para la detección temprana de incendios forestales" el cual ha sido desarrollado para optar por el título de: Ingeniero en Telecomunicaciones, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, 2 de agosto de 2024

Atentamente,

Juan Diego Armijos Goercke

0106908734

Cristhian Marcelo Guerrero Abad

0105168074

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Juan Pablo Bermeo Moyano con documento de identificación N° 0102645884, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DESARROLLO DE UN PROTOTIPO IOT PARA LA DETECCIÓN TEMPRANA DE INCENDIOS FORESTALES, realizado por Juan Diego Armijos Goercke con documento de identificación N° 0106908734 y Cristhian Marcelo Guerrero Abad con documento de identificación N° 0105168074, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Cuenca, 2 de agosto de 2024

Atentamente,

Ing. Juan Pablo Bermeo Moyano, Mgst.
0102645884

AGRADECIMIENTOS

Agradezco a todas las personas que han participado por ayudarme en este proceso, a mis padres Diego y Lornyn, a mi hermano Santiago, a mis queridas abuelitas Lucy y Cuma y a toda mi familia, a mis amigos y sobre todo a mis profesores de esta universidad que sin ellos no hubiera resurgido mi amor por esta carrera.

Juan Diego Armijos Goercke

Quiero agradecer a todas las personas que estuvieron conmigo durante todo este proceso de vida universitaria. A mis padres, Marcelo y Eliana, de manera muy especial a mi abuelita Elvia Peña, quien fue mi mayor apoyo en este proceso. También agradezco a todos los amigos que pude conocer en este tiempo, en especial a mi mejor amiga Ángeles Quezada, quien compartió todo este tiempo a mi lado, siendo un soporte muy importante para mí. Sin ellos, no hubiera podido tener el ánimo para hacer las cosas con amor y afrontar la carrera con el coraje y la valentía que se requiere.

Cristhian Marcelo Guerrero Abad

DEDICATORIAS

Dedicatoria de Juan Diego Armijos Goercke

Dedico este trabajo de titulación a todas las personas que me han ayudado en el proceso de aprendizaje a lo largo de estos años, a todos aquellas personas que conocí y de alguna manera me han ayudado y a todos los que hicieron difícil este camino, sin ustedes esto no sería posible.

Dedicatoria de Cristhian Marcelo Guerrero Abad

Quiero dedicar mi trabajo de titulación a todas las personas que creyeron y confiaron en mí, quienes fueron pilares en este proceso de aprendizaje durante mi vida universitaria. De la misma forma, agradezco a aquellas personas que complicaron mi camino, ya que sin esos conflictos o rechazos, no habría sido posible alcanzar esta meta.

Índice general

Agradecimientos I

Dedicatorias II

Índice General III

Índice de figuras VII

Índice de tablas VIII

Resumen IX

Abstract XI

Antecedentes 1

Justificación 3

Objetivos 4

Objetivo General 4

Objetivos Específicos 4

Introducción 5

1. Capítulo I: Fundamentación Teórica 7

1.1. Conceptos Fundamentales 7

1.1.1. Redes IoT 7

1.1.2. Raspberry Pi Pico 13

1.1.3. ESP32 . . . 15
1.1.4. Sistema de detección temprana . . . 18
1.1.5. LoRa . . . 18
1.1.6. LoraWan . . . 20
1.1.7. Angular . . . 22
1.1.8. Bases de datos . . . 23
1.2. Node-Red . . . 23
1.3. Estudios Relacionados . . . 24

2. Capítulo II: Diseño **27**
2.1. Diseño electrónico del prototipo . . . 28
2.2. Diseño de la carcasa física del prototipo . . . 35
2.3. Diseño de la página web . . . 40
2.4. Diseño de la base de datos . . . 70
2.5. Diseño de Comunicación LoRaWAN y Página Web . . . 80
2.6. Análisis de resultados . . . 81

3. Capítulo III: Validación Experimental **88**
3.1. Construcción del prototipo . . . 88
3.2. Experimentación . . . 89
3.3. Análisis de datos experimentales . . . 93

4. Capítulo IV: Discusión **96**
4.1. Interpretación de los resultados . . . 96
4.2. Comparación con estudios previos . . . 98

5. Capítulo V: Conclusiones y Recomendaciones **100**
5.1. Conclusiones . . . 100
5.2. Recomendaciones . . . 101
5.3. Limitaciones del estudio . . . 102
5.4. Sugerencias para futuras investigaciones . . . 102

Glosario **104**

Referencias **109**

Índice de figuras

1.1. Vista superior de la placa Raspberry Pico 14
1.2. Vista inferior de la placa Raspberry Pico 15
1.3. Distribución de pines placa Raspberry Pi Pico 15
1.4. Vista superior de la placa ESP32 LoRa V3 17
1.5. Distribución de pines placa ESP32 LoRa V3 17
1.6. Topología de la red LoRaWAN 21

2.1. Diagrama de conexión 27
2.2. Esquema general de conexiones para prototipo 28
2.3. Diseño esquemático de la fuente 29
2.4. Diseño PCB de la fuente 29
2.5. Esquema de independencia de energía 30
2.6. Diagrama programa electrónico 31
2.7. Diseño carcasa principal 35
2.8. Medidas carcasa principal 36
2.9. Diseño de tapas para la carcasa 37
2.10. Medidas tapas 37
2.11. Diseño soporte para panel solar 38
2.12. Medidas soporte principal 38
2.13. Diseño soporte principal para carcasa 39
2.14. Medidas conjunto sujeción 39
2.15. Ensamblaje del prototipo para la detección de incendios 40
2.16. Diagrama funcionamiento frontend 41
2.17. Diagrama de funcionamiento de la comunicación Gateway - Página web 80

2.18. Diseño del flujo en Node-Red 81
2.19. Ingreso a la página web 81
2.20. Dashboard 82
2.21. Devices 82
2.22. Alarms 83
2.23. Templates 83
2.24. Mapa 84
2.25. Notificaciones 84
2.26. Registro de nuevos usuarios 85
2.27. Base de datos 85
2.28. Uso gateway 86
2.29. Recepción de datos 86
2.30. Datos recibidos 86
2.31. Datos recibidos ampliados 87
2.32. Comunicación activa entre Gateway y Página web 87

3.1. Prototipo para la detección de incendios 89
3.2. Uso de batería 90
3.3. RSSI 90
3.4. SNR 91
3.5. Distancia vs Latencia con SF=12 91
3.6. Distancia vs Latencia con SF=9 92
3.7. Distancia vs Latencia con SF=5 92

Índice de tablas

1.1. Resumen de los trabajos relacionados con Detección temprana de incendios mediante IoT 24

Resumen

En este proyecto, se ha llevado a cabo el desarrollo y la implementación de un prototipo de sistema basado en Internet de las Cosas (IoT) para la detección temprana de incendios forestales. Este trabajo responde a la urgente necesidad de contar con herramientas eficaces y precisas para prevenir y combatir incendios forestales, que representan una amenaza significativa para los ecosistemas naturales, la biodiversidad y las comunidades humanas.

- Eficiencia de Detección Mejorada: Se ha logrado una notable mejora en la eficiencia de detección de incendios forestales, reduciendo el tiempo de respuesta necesario para tomar medidas preventivas o de contención.
- Integración IoT Completa: Se ha desarrollado e implementado un sistema integral que incorpora sensores IoT, una red de comunicación y una plataforma de monitoreo en tiempo real, permitiendo una supervisión continua de áreas forestales remotas.
- Precisión y Fiabilidad: El prototipo ha demostrado una alta precisión en la detección de eventos relacionados con incendios forestales, minimizando tanto los falsos positivos como los falsos negativos y proporcionando una base sólida para la toma de decisiones informadas.
- Robustez en Entornos Hostiles: El sistema ha sido diseñado para funcionar de manera robusta y confiable en entornos forestales adversos, resistiendo condiciones climáticas extremas y garantizando su operatividad en situaciones difíciles.
- Enfoque en la Sostenibilidad: Se ha prestado especial atención a la sostenibilidad

del sistema, tanto en términos de eficiencia energética como de impacto ambiental, asegurando que su implementación contribuya positivamente a la conservación del medio ambiente.

Palabras clave: IoT; Detección de incendios; sensores; LoRa; LoRaWAN; ESP32; Docker

Abstract

This project entails the development and implementation of a prototype Internet of Things (IoT) system for the early detection of forest fires. The endeavor addresses the pressing need for effective and precise tools to prevent and combat forest fires, which pose a significant threat to natural ecosystems, biodiversity, and human communities.

- Enhanced Detection Efficiency: A notable improvement in forest fire detection efficiency has been achieved, significantly reducing the response time required to implement preventive or containment measures.
- Full IoT Integration: An integrated system has been developed and deployed, incorporating IoT sensors, a communication network, and a real-time monitoring platform, enabling continuous monitoring of remote forested areas.
- Precision and Reliability: The prototype has demonstrated high accuracy in detecting events related to forest fires, minimizing both false positives and false negatives and providing a solid basis for informed decision-making.
- Robustness in Hostile Environments: The system has been designed to operate robustly and reliably in adverse forest environments, withstanding extreme weather conditions and ensuring operational continuity in challenging situations.arantizando su operatividad en situaciones difíciles.
- Focus on Sustainability: Special attention has been paid to the sustainability of the system, in terms of both energy efficiency and environmental impact, ensuring that its implementation positively contributes to environmental conservation.

Keywords: IoT; sensor; Fire detection; LoRa; LoRaWAN; ESP32; Docker

Antecedentes

Los incendios forestales son uno de los mayores problemas de seguridad y medio ambiente del mundo. Estos eventos no solo causan daños irreparables a los ecosistemas, sino que también ponen en peligro la vida humana, la propiedad y los recursos económicos [1].

Por ejemplo, en diciembre de 2023, en la zona de la laguna Llaviuco del Parque Nacional El Cajas, en la zona rural de la ciudad de Cuenca, Ecuador, se desató un devastador incendio forestal que arrasó aproximadamente 200 hectáreas de bosque y pajonal. Este incidente ha provocado un daño irreparable tanto en el ecosistema como en las personas afectadas [2]. Además, en septiembre de 2023, el Cuerpo de Bomberos de Cuenca contabilizó 30 incendios durante un fin de semana. Todos ocurrieron en las parroquias rurales de la capital azuaya [3].

La detección temprana de incendios forestales es crucial para reducir su propagación y los efectos negativos asociados [4]. Por ello, el desarrollo de sistemas de detección sofisticados se ha convertido en un campo de investigación y desarrollo activo para satisfacer esta necesidad crucial [5]. Debido a su capacidad para integrar una variedad de sensores y tecnologías de comunicación para monitorizar condiciones ambientales de manera remota y en tiempo real, las plataformas de Internet de las cosas (IoT) se han convertido en una solución prometedora en este contexto [6].

La necesidad de desarrollar un prototipo para la detección temprana de incendios forestales mediante una plataforma IoT se fundamenta en múltiples razones de importancia crítica. El objetivo de esta iniciativa es preservar el medio ambiente, proteger propiedades y vidas humanas, optimizar los recursos de respuesta a emergencias, reducir el impacto del cambio climático y aprovechar los avances tecnológicos para mejorar la eficiencia y la efectividad de la gestión de

incendios forestales. La importancia de abordar este problema y desarrollar soluciones innovadoras para combatir de manera efectiva los incendios forestales se justifica por la combinación de estos elementos.

Justificación

La importancia de detectar incendios forestales en etapas tempranas es crucial para minimizar su impacto negativo. Recientes incidentes subrayan la urgencia de abordar esta problemática. La aplicación de la tecnología de Internet de las cosas (IoT) emerge como una solución prometedora para vigilar las condiciones ambientales en tiempo real. Este proyecto se enfoca en la conservación del medio ambiente, la protección de vidas y propiedades, la optimización de las respuestas ante emergencias y la mitigación del cambio climático. La necesidad de desarrollar este prototipo se basa en la importancia crítica de combatir incendios forestales de manera eficiente para garantizar la seguridad de las comunidades y promover la gestión sostenible de los recursos naturales.

Objetivos

Objetivo General

- Desarrollar un prototipo para la detección temprana de incendios forestales utilizando una plataforma IoT.

Objetivos específicos:

- Diseñar un sistema de sensores que pueda detectar señales de incendios forestales, integrando tecnologías de IoT para la recolección y transmisión de datos en tiempo real.
- Diseñar una base de datos para interpretar la información recopilada por el sistema de sensores, permitiendo una detección temprana y precisa de posibles incendios forestales.
- Diseñar una interfaz web para la visualización en tiempo real de los datos, que permita a los usuarios recibir alertas y notificaciones sobre posibles incendios forestales en su área.

Introducción

El sistema de detección temprana de incendios forestales se implementará utilizando tecnologías LoRaWAN junto con un ESP32. Cada punto de monitoreo estará equipado con sensores de temperatura, humedad, dióxido de carbono y GPS para mantener un seguimiento de las condiciones ambientales en tiempo real. La información recabada por estos sensores se enviará a través de la red hacia un punto central, los cuales serán transmitidos a las plataformas IoT mediante una conexión a Internet. En la plataforma, los datos se almacenarán en una base de datos para su posterior análisis y visualización.

Estos datos recopilados estarán disponibles en una aplicación web. La aplicación web ofrecerá una interfaz de usuario para visualizar los datos en tiempo real, incluyendo la ubicación y las mediciones de temperatura y humedad de cada punto de monitoreo. Además, se incorporará un mapa interactivo que mostrará la ubicación exacta de cada punto de monitoreo y cualquier alerta de incendio detectada.

Cuando el sistema detecte una temperatura elevada o un cambio brusco en la humedad, que pueda indicar la presencia de un posible incendio forestal, enviará una alerta a los usuarios a través de un mensaje y también mostrará una notificación visual en el mapa de la aplicación web.

El enfoque del prototipo estará dirigido hacia la zonas rurales, bosques o pajonales, excluyendo consideraciones sobre la extinción de incendios para enfocarse únicamente en la detección temprana. Sin embargo, su alcance estará restringido por la cobertura de cada nodo, determinada por la ubicación y la línea de visión desde el punto de acceso a los sensores, lo que podría afectar la cobertura en áreas con topografía compleja o vegetación densa. La precisión en la detección estará directamente influenciada por la calidad y calibración de los sensores de temperatura

y humedad, así como por la disponibilidad y precisión de los datos GPS.

Cabe recalcar que en este trabajo de titulación se construirá únicamente un prototipo con dos nodos, las pruebas se realizarán en ambientes controlados de laboratorio, y en pruebas de campo simulando escenarios reales.

Capítulo 1

Capítulo I: Fundamentación Teórica

La creciente amenaza de incendios forestales en todo el mundo ha generado una necesidad urgente de desarrollar sistemas de detección temprana efectivos y eficientes. En respuesta a este desafío, se ha propuesto el uso de tecnologías emergentes como el Internet de las Cosas (IoT) para mejorar la capacidad de monitoreo y respuesta ante incendios forestales.

Este capítulo se enfoca en establecer una sólida base teórica que sustente el desarrollo de un prototipo IoT para la detección temprana de incendios forestales. Se explorarán los principales conceptos y teorías relacionadas con la detección de incendios, el funcionamiento del IoT y su aplicación en entornos forestales.

1.1. Conceptos Fundamentales

1.1.1. Redes IoT

Definición

El intercambio de bienes y servicios entre las redes de la cadena de suministro se facilita con la arquitectura emergente del Internet de las cosas (IoT), que se basa en Internet global [7]. En 1999, Kevin Ashton trabajó en tecnología de RFID. La Internet de las cosas ha dado a las personas numerosas oportunidades de acceso a datos y servicios en áreas como la educación, la seguridad, la atención médica y el transporte, entre otras, lo que ha tenido un impacto significativo en la calidad de vida de las

personas. Además, brinda la oportunidad de explorar nuevos modelos de negocios y la creación de dispositivos inteligentes interconectados.

Actualmente, los dispositivos IoT incluyen una amplia variedad de dispositivos conectados, tanto para uso de consumo como para uso comercial (B2B). [8]. De 2010 a 2020, el número de dispositivos conectados al Internet de las cosas aumentó en un 1,000 %. Aproximadamente 10 mil millones de estos dispositivos se han desplegado e interconectado en los últimos diez años. Muchos de estos dispositivos de Internet de las cosas (IoT) dependen de microcontroladores de bajo consumo. Cada vez más despliegues dependen de microcontroladores con códigos más complejos y conectados a una red, ya sea de forma directa o indirecta.

La estandarización de una arquitectura para la IoT aún está en proceso de desarrollo. Enfocados a resolver dos problemas fundamentales: *establecer una forma estándar de acceso al medio y a los dispositivos* e *integrar los dispositivos a Internet* [9]. Diversas organizaciones están trabajando en propuestas de estandarización, incluido IEEE, con sus grupos de trabajo 802.15 y el protocolo 802.15.4, que permite la comunicación con bajas tasas de transmisión. Varios protocolos que la IETF ha desarrollado incluyen el protocolo CoAP, que se especializa en la transferencia de datos a través de la Web y permite la comunicación entre dispositivos con recursos limitados al correr sobre UDP. Estos son solo algunos ejemplos porque hay muchas organizaciones que trabajan en la estandarización del medio.

Aplicaciones de los Sistemas IoT

De manera general, las características de las IoT permiten tener diferentes ramas de aplicaciones entre las cuales tenemos [10]:

- **Detección y Compartición de Información de Ubicación:** El sistema IoT tiene la capacidad de recopilar y compartir información de ubicación de terminales IoT y nodos finales, utilizando datos geográficos obtenidos a través de GPS, CellID, RFID, entre otros. Esta información se aprovecha en diversas aplicaciones como el seguimiento móvil de activos, que permite monitorear el estado de mercancías, la gestión de flotas para programar vehículos y conductores según las necesidades empresariales, y el sistema de información de tráfico, que ofrece

detalles sobre el estado de las carreteras y áreas congestionadas mediante el seguimiento de la información de posición de múltiples vehículos.

- **Detección del entorno:** A través de terminales desplegados a gran escala, los sistemas de Internet de las cosas pueden recopilar y procesar una variedad de parámetros ambientales físicos o químicos. Estos datos incluyen temperatura, humedad, ruido, etc. Las aplicaciones típicas abarcan desde la monitorización medioambiental y ecológica en bosques y glaciares, la vigilancia de catástrofes como volcanes y sismos, hasta la supervisión en fábricas. Estos sistemas cuentan con alarmas automáticas basadas en parámetros ambientales recopilados por una gran cantidad de sensores. Además, el IoT se utiliza en la monitorización médica remota para analizar datos indicadores recurrentes recolectados por dispositivos colocados en el cuerpo de los pacientes, proporcionando tendencias de salud y consejos médicos a los usuarios.

- **Control remoto:** Los sistemas IoT tienen la capacidad de controlar los terminales IoT y ejecutar funciones basadas en comandos de aplicación, combinados con la información recopilada de los dispositivos y los requisitos del servicio. Esto se refleja en dos áreas principales: el control de aparatos, donde las personas pueden gestionar remotamente el funcionamiento de electrodomésticos, y la recuperación de desastres, que permite a los usuarios activar de manera remota las instalaciones de tratamiento de emergencias para minimizar pérdidas ocasionadas por eventos catastróficos, utilizando datos previamente monitoreados.

- **Redes ad hoc:** El sistema IoT debe poseer la capacidad de autoorganización rápida en red y ser capaz de interoperar con la capa de red/servicio para ofrecer servicios relacionados. Por ejemplo, en una red de vehículos, la infraestructura vial y los propios vehículos pueden autoorganizarse rápidamente para transferir datos de manera eficiente.

- **Comunicación segura:** En función de las necesidades del servicio, el sistema de Internet de las cosas puede establecer un canal seguro de transmisión de datos entre la plataforma de aplicaciones o servicios y los terminales de Internet de las

cosas. Una aplicación IoT puede incluir una variedad de capacidades e incluso aplicaciones que se adapten a los requisitos del servicio.

- **Smart City:** La ciudad inteligente es un nuevo modelo de desarrollo de una ciudad que utiliza nuevas tecnologías, como IoT, computación en la nube y análisis de big data, para impulsar el intercambio y la coordinación de información dentro de un sistema urbano. IoT es un medio y una herramienta importante para construir ciudades inteligentes, y es una construcción de ciudades inteligentes que sustenta la infraestructura. La construcción de ciudades inteligentes depende de muchas aplicaciones de IoT para diferentes industrias [11]

- **Agricultura:** En el ámbito de la alimentación y la agricultura, hay un gran potencial de aplicación para las tecnologías de Internet de las cosas, especialmente en vista de los desafíos sociales y ambientales que enfrenta este sector. Las tecnologías de Internet de las cosas podrían cambiar el sector de la granja a la mesa, asegurando la seguridad alimentaria y reduciendo el desperdicio y los insumos agrícolas. [12].

- **Transporte:** El sistema de transporte inteligente en China se está desarrollando y manteniendo una alta tasa de crecimiento. La velocidad creciente del mercado de sistemas de control de transporte inteligente urbano de China es alta, incluida la policía electrónica, el control inteligente de señales de tráfico, el monitoreo por video del tráfico, la gestión inteligente de servicios de taxi, la tecnología de información del transporte público urbano y etc. Hasta la finalización del XII Plan Quinquenal, el 60 por ciento de las carreteras nacionales de China tendrán la instalación de infraestructuras de IoT [13].

- **Smart Mobility:** Smart Mobility es la metodología que permite viajar sin problemas, eficiente y flexible en varios modos. Con el cambio de tiempo y el constante aumento de las demandas de la sociedad, la Red Vehicular Ad-hoc (VANET) ha estado mucho en conversaciones. Por tanto, se puede afirmar que se trata de un cambio de paradigma hacia un sistema de transporte más flexible y multimodal. De hecho, es el pilar de Internet de los vehículos (IoV) que tiende

a mejorar la seguridad vial, ya sea previniendo o reduciendo los accidentes de tráfico, brindando nuevas soluciones hacia modos de transporte optimizados [14].

- **Smart Grid:**Una red de suministro de energía inteligente utiliza tecnología de comunicaciones digitales para detectar y responder a los cambios en el uso local. También puede denominarse tecnología digital que permite la comunicación bidireccional, lo que permite a los clientes satisfacer sus necesidades de electricidad después de realizar observaciones con la ayuda de sensores y en reciprocidad [14].

- **Medicina:** A menudo denominado Internet de las cosas médicas (IoMT), es el modo de aplicación sistemática que conecta los servicios de atención médica al sistema de TI a través de varias redes informáticas en línea. Los dispositivos médicos están equipados con sistemas Wi-Fi incorporados que permiten aún más la comunicación entre máquinas basada en el concepto IoMT [14].

Retos y Barreras de la IoT

Hacer realidad la visión de la IoT no es tarea fácil se enfrenta a diversos desafíos que requieren atención. Entre ellos se encuentran la disponibilidad, fiabilidad, movilidad, rendimiento, escalabilidad, interoperabilidad, seguridad, gestión y confianza. Los proveedores de servicios y desarrolladores de aplicaciones pueden implementar sus servicios de manera efectiva al abordar estos problemas. A continuación, se proporciona un análisis de los principales problemas en las fases de desarrollo y despliegue de Internet de las cosas. [15].

- **Disponibilidad:** Se refiere tanto al hardware como al software para brindar servicios a los clientes en cualquier momento y lugar. En cuanto al software, la disponibilidad se relaciona con la capacidad de proporcionar servicios a nivel global y simultáneamente en diferentes ubicaciones. Por otro lado, la disponibilidad del hardware se refiere a la presencia constante de dispositivos compatibles con las funcionalidades y protocolos del IoT.

- **Fiabilidad:** La fiabilidad se define como la capacidad del sistema para funcionar correctamente de acuerdo con su especificación, con el objetivo de aumentar la tasa de éxito en la prestación de servicios IoT. Este aspecto está relacionado con la disponibilidad, ya que garantiza que la información y los servicios estén disponibles de manera continua y consistente a lo largo del tiempo.

- **Movilidad:** Se prevé que la mayoría de los servicios IoT se brinden a usuarios móviles, por lo que conectar de manera continua a los usuarios con los servicios que requieren mientras se desplazan es fundamental en la IoT. La interrupción del servicio para dispositivos móviles puede ocurrir al transferirse de una pasarela a otra. Además, el gran número de dispositivos inteligentes en los sistemas IoT demanda la implementación de mecanismos eficientes para la gestión de la movilidad.

- **Rendimiento:** Evaluar el rendimiento de los servicios IoT representa un desafío significativo, ya que depende del desempeño de numerosos componentes y tecnologías adjuntas. Es crucial supervisar y evaluar los dispositivos IoT para garantizar un rendimiento óptimo a un costo asequible para los clientes. Para evaluar el rendimiento del IoT, se pueden emplear diversos parámetros, como la velocidad de procesamiento, la velocidad de comunicación, el factor de forma del dispositivo y el coste.

- **Gestión:** La conexión de miles de millones de dispositivos inteligentes plantea a los proveedores de servicios enormes problemas para gestionar los aspectos de fallos, configuración, contabilidad, rendimiento y seguridad (FCAPS) de estos dispositivos. Este esfuerzo de gestión hace necesario el desarrollo de nuevos protocolos de gestión ligeros para manejar la potencial pesadilla de gestión que se derivará del despliegue del IoT en los próximos años.

- **Escalabilidad:** Se define como la capacidad de incorporar nuevos dispositivos, servicios y funciones para los clientes sin comprometer la calidad de los servicios existentes. Agregar nuevas operaciones y dar soporte a nuevos dispositivos puede resultar desafiante, especialmente con la presencia de diversas plataformas de hardware y protocolos de comunicación. Por lo tanto,

las aplicaciones IoT deben diseñarse desde el principio para permitir servicios y operaciones que sean fácilmente escalables.

- **Interoperabilidad:** Representa un desafío adicional para la IoT debido a la diversidad de dispositivos pertenecientes a diferentes plataformas. Tanto los desarrolladores de aplicaciones como los fabricantes de dispositivos IoT deben considerar la interoperabilidad para asegurar la prestación de servicios a todos los clientes, sin importar las especificaciones de la plataforma de hardware utilizada.

- **Seguridad y privacidad:** Representa un desafío para las implementaciones de IoT debido a la falta de estándares y arquitecturas comunes en este ámbito. En redes heterogéneas como la IoT, garantizar la seguridad y la privacidad de los usuarios no resulta sencillo. La funcionalidad principal de la IoT implica el intercambio de información entre miles de millones de objetos conectados a Internet. Un problema abierto en la seguridad del IoT es la distribución de claves entre los dispositivos, un aspecto que aún no se ha abordado adecuadamente en los estándares. Es crucial asegurar los intercambios de datos para prevenir la pérdida o la vulneración de la privacidad.

1.1.2. Raspberry Pi Pico

La fundación Raspberry Pi ha cambiado el mundo al proporcionar placas de computadora potentes y de bajo costo. La Raspberry Pi es, con diferencia, la más vendida y la más popular de las muchas placas de ordenador pequeñas disponibles. Quizás aún más importante sea la Raspberry Pi diseñada para la educación. Los educadores pueden utilizar Raspberry Pi para enseñar proyectos de informática, electrónica, automatización de hardware y Internet de las cosas (IoT) utilizando lenguajes de programación Python, Java o C++. La Raspberry Pi Pico se aleja del dominio de las placas de computadora pequeñas Raspberry Pi porque no es otra placa de computadora pequeña. Por lo tanto, no tiene la capacidad de ejecutar un sistema operativo y no hay puertos de video, ni puertos de host USB, ni siquiera un conector de alimentación. Más bien, Raspberry Pi Pico es el primer microcontrolador

que utiliza un pequeño chip basado en Raspberry Pi (RP2040). [16]. La Raspberry Pico es una pequeña placa de circuito impreso verde del tamaño de una barra de chicle. A lo largo de cada lado largo están los pines GPIO con un conector micro-USB en uno de los extremos más cortos. En el otro extremo hay un conjunto de pines de depuración que puede utilizar para diagnósticos avanzados. La Figura 1.1 muestra el Pico desde arriba orientado con el puerto USB hacia la derecha. Los encabezados GPIO en los bordes superior e inferior. Los tres pines de la izquierda son los pines de depuración. El único otro componente de la placa que debemos conocer es el interruptor BOOTSEL ubicado en la parte superior derecha de la figura. Este interruptor se usa para colocar el Pico en modo de inicio donde ejecuta la plataforma MicroPython o, si se mantiene presionado mientras el cable USB está conectado a su computadora, se conectará como una unidad extraíble que le permitirá cargar nuevos archivos o cambiar la base. archivos de plataforma. Veremos cómo hacer esto más adelante en esta sección [16].

Figura 1.1: Vista superior de la placa Raspberry Pico

La Figura 1.2 muestra la parte inferior del Pico. Observamos que los pines GPIO están etiquetados, lo que facilita la localización de un pin específico. Los lugares etiquetados con "TP"son puntos de prueba que puede usar para probar el voltaje en caso de que necesite realizar algún diagnóstico avanzado de la placa. Una vez más, los pines de la izquierda son para la interfaz Serial Wire Debug (SWD).

Figura 1.2: Vista inferior de la placa Raspberry Pico

La distribución de pines se muestra en la siguiente figura 1.3.

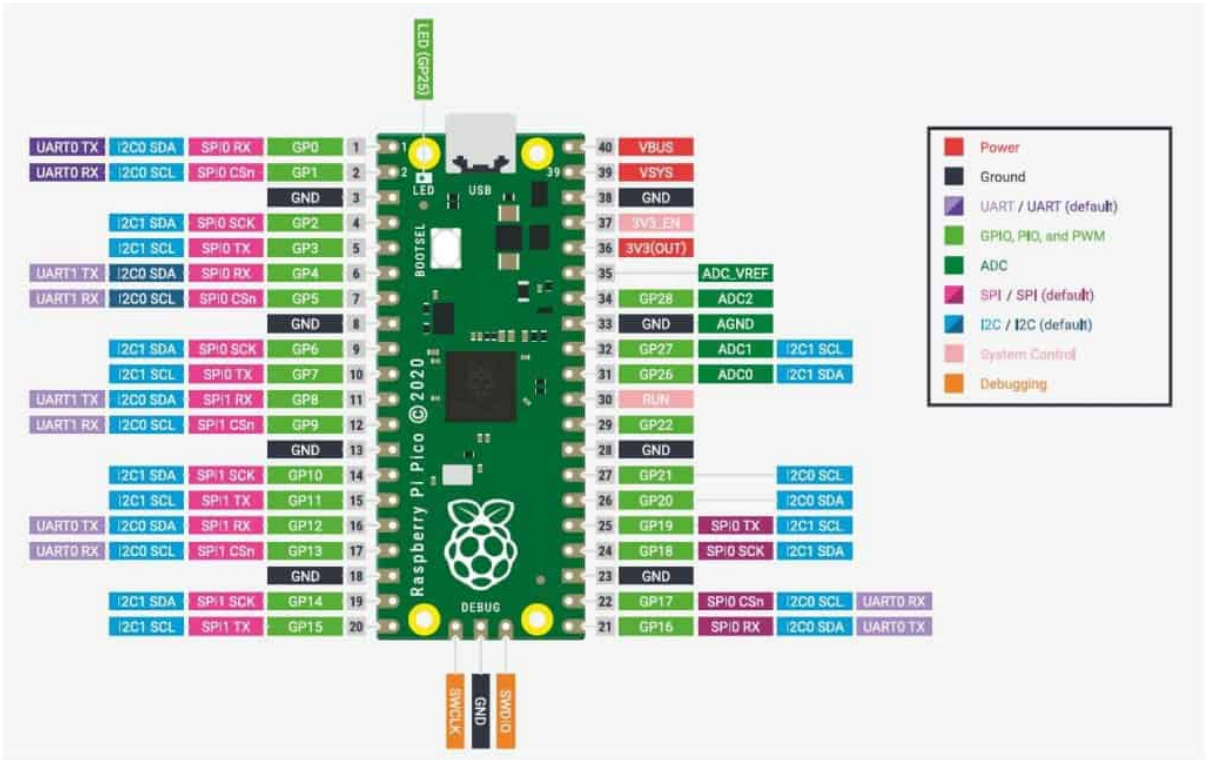

Figura 1.3: Distribución de pines placa Raspberry Pi Pico

1.1.3. ESP32

ESP32 es un potente microcontrolador SoC (System on Chip) con Wi-Fi 802.11 b/g/n integrado, Bluetooth de modo dual versión 4.2 y variedad de periféricos. Es

un sucesor avanzado del chip 8266 principalmente en la implementación de dos núcleos sincronizados en diferentes versiones hasta 240 MHz. En comparación con su predecesor, además de estas características, también amplía el número de pines GPIO de 17 a 36, el número de canales PWM por 16 y está equipado con 4 MB de memoria flash 1.4.

ESP32 incluye dos núcleos. Los núcleos de la CPU se pueden controlar individualmente. Hay 520 KB de SRAM en chip para datos e instrucciones disponibles. Algunos módulos SoC, como ESP32-Wrover, cuentan con 4 MB de flash SPI externo y 8 MB adicionales de SPI PSRAM. Tenemos la posibilidad de utilizar SPI, I2S, I2C, CAN, UART, Ethernet MAC e IR en varias cantidades, dependiendo del tipo de placa. El equipamiento estándar también incluye un sensor de efecto Hall, un sensor de temperatura y un sensor táctil; otros sensores integrados se implementan en Azure IoT y el kit de desarrollador. SoC también proporciona aceleración de hardware criptográfico 1.4.

Las placas ESP32 se producen en diseños de prototipos que se pueden utilizar en aplicaciones domésticas inteligentes, automatización, dispositivos portátiles, aplicaciones de audio, aplicaciones de IoT basadas en la nube y más. Es posible elegir un kit de desarrollo específico o diseñar un sistema integrado personalizado construido sobre el microcontrolador ESP32 [17]. La placa ESP32 se muestar en la siguiente figura 1.4.

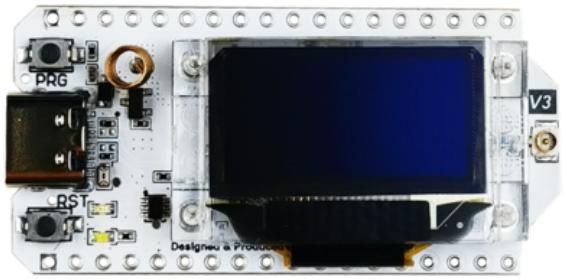

Figura 1.4: Vista superior de la placa ESP32 LoRa V3

Los pines de la placa ESP32 LoRa V3 se observan en la figura 1.5.

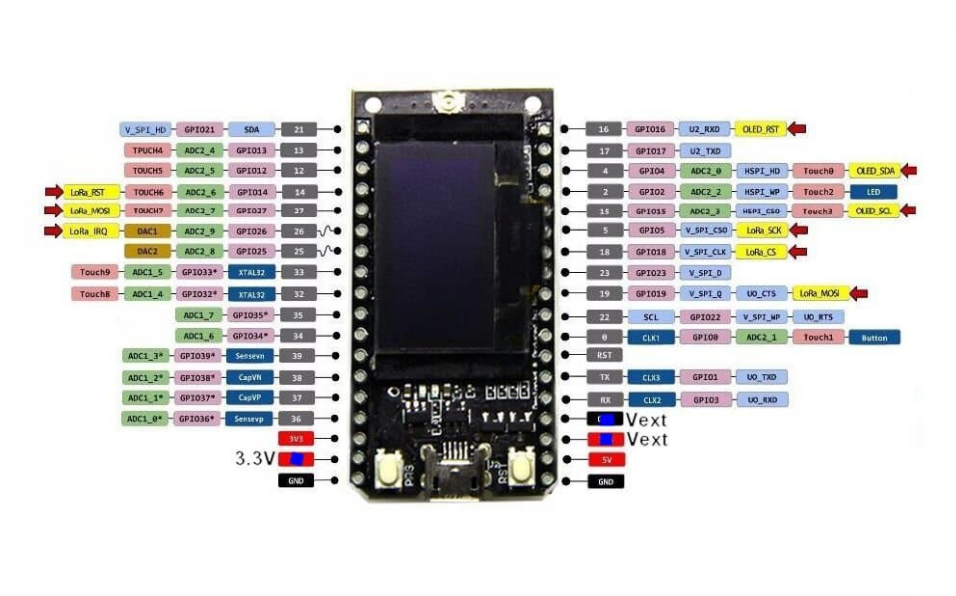

Figura 1.5: Distribución de pines placa ESP32 LoRa V3

1.1.4. Sistema de detección temprana

Los incendios forestales son fuegos incontrolados que se extienden libremente en áreas boscosas. La detección es el primer paso crucial en el manejo y control de estos incendios. La importancia de la detección temprana de incendios forestales (DTIF) radica en que permite su extinción a un costo menor, evitando que grandes áreas de bosque sean consumidas. Es esencial utilizar sensores que monitoreen variables químicas como las emisiones de humo, CO, CO2 y CH4, así como variables físicas como la radiación infrarroja y el calor. La DTIF está diseñada para emitir una alarma en cuanto se detecta el inicio de un incendio [18].

1.1.5. LoRa

La capa física responsable de proporcionar enlaces de comunicación de largo alcance es LoRa. Con la adición de una capa MAC, LoRaWAN (Red de Área Amplia de Largo Alcance) se ha estandarizado y ampliado, estableciendo el protocolo de comunicación y la arquitectura de la red. La Alianza LoRa ofrece tanto código abierto como especificaciones estandarizadas para LoRaWAN.

Como su nombre lo sugiere, LoRa es un protocolo diseñado para largas distancias, capaz de transmitir datos a distancias considerables. Una única puerta de enlace puede cubrir un área de hasta cien kilómetros cuadrados. El alcance extenso de la tecnología LoRa se debe a su presupuesto de enlace y a la modulación de espectro ensanchado chirp que utiliza.

LoRa emplea la técnica de modulación de espectro ensanchado chirp, una técnica robusta y de largo alcance utilizada en comunicaciones militares y espaciales durante décadas. Ahora, se aplica comercialmente en las comunicaciones LoRa, proporcionando inmunidad a los trayectos múltiples y al desvanecimiento. El espectro ensanchado chirp tiene bajos requerimientos de potencia de transmisión. Un chirrido es una señal cuya frecuencia varía con el tiempo, pudiendo ser ascendente o descendente [19].

La red Lora consiste en cuatro puntos básicos:

- Nodos

- Gateway
- Servidor de red
- Servidor de la aplicación

Nodos

Los nodos LoRa comprenden sensores o aplicaciones donde se lleva a cabo la detección y el control, como sensores, dispositivos de seguimiento, etc. Este es el corazón de un sistema integrado. Estos incluyen medidores de agua, alarmas de humo, monitoreo de gas y aplicaciones similares.

Gateway

El gateway comprende el elemento neto de la red LoRa. Cada nodo final está conectado a cada puerta de enlace (gateway). El nodo envía datos a todas las puertas de enlace, y cada puerta de enlace recibe una señal y la envía a un servidor de red basado en la nube a través de Ethernet, satélite, Wi-Fi o celular. La puerta de enlace puede ser una micro puerta de enlace o una pico puerta de enlace. Las micropuertas de enlace se utilizan en la red pública para brindar cobertura a nivel urbano o nacional, mientras que la puerta de enlace Pico se usa en áreas densas de difícil acceso para mejorar la calidad del servicio y la capacidad de la red. La micro puerta de enlace ofrece una alta cobertura. En la estación base LoRa se utilizan antenas omnidireccionales y multisectoriales.

Servidor de red

El servidor de red tiene toda la inteligencia de la red. Los datos recibidos de diferentes puertas de enlace se filtran, se realizan comprobaciones de seguridad, velocidades de datos adaptables, etc. y se envía un acuse de recibo a las puertas de enlace . El servidor de red es quien identifica si los datos recibidos están destinados a algún servidor de aplicaciones y, por lo tanto, se envían al servidor de aplicaciones previsto a través de algún backhaul.

1.1.6. LoraWan

LoRaWAN es un protocolo de red de capa MAC basado en la nube, desarrollado y mantenido por LoRa Alliance, que se encarga de definir las capas superiores de las redes de área amplia de largo alcance utilizando la capa física de LoRa. Esta tecnología LPWAN está diseñada para satisfacer las principales necesidades del Internet de las Cosas (IoT), incluyendo la comunicación bidireccional, la seguridad de extremo a extremo, la movilidad y la localización. Se implementa en redes regionales, nacionales o globales para conectar dispositivos inalámbricos alimentados por batería a Internet. La capa física de LoRa permite enlaces de comunicación a larga distancia, mientras que el protocolo LoRaWAN se enfoca en enrutar la capa de red entre las puertas de enlace LoRaWAN y los dispositivos LoRa. Además, LoRaWAN gestiona la velocidad de datos, las frecuencias de comunicación y la potencia de transmisión de todos los nodos de la red, que son asíncronos y transmiten datos según sea necesario.[20].

La arquitectura de red LoRaWAN tiene una topología en estrella. Un servidor de red central está conectado a múltiples puertas de enlace. Las puertas de enlace LoRaWAN solo pueden enviar paquetes de datos sin procesar desde los nodos finales al servidor de red encapsulándolos en paquetes UDP/IP [21].

La arquitectura de red LoRaWAN resultante se muestra en la Figura 1.6.

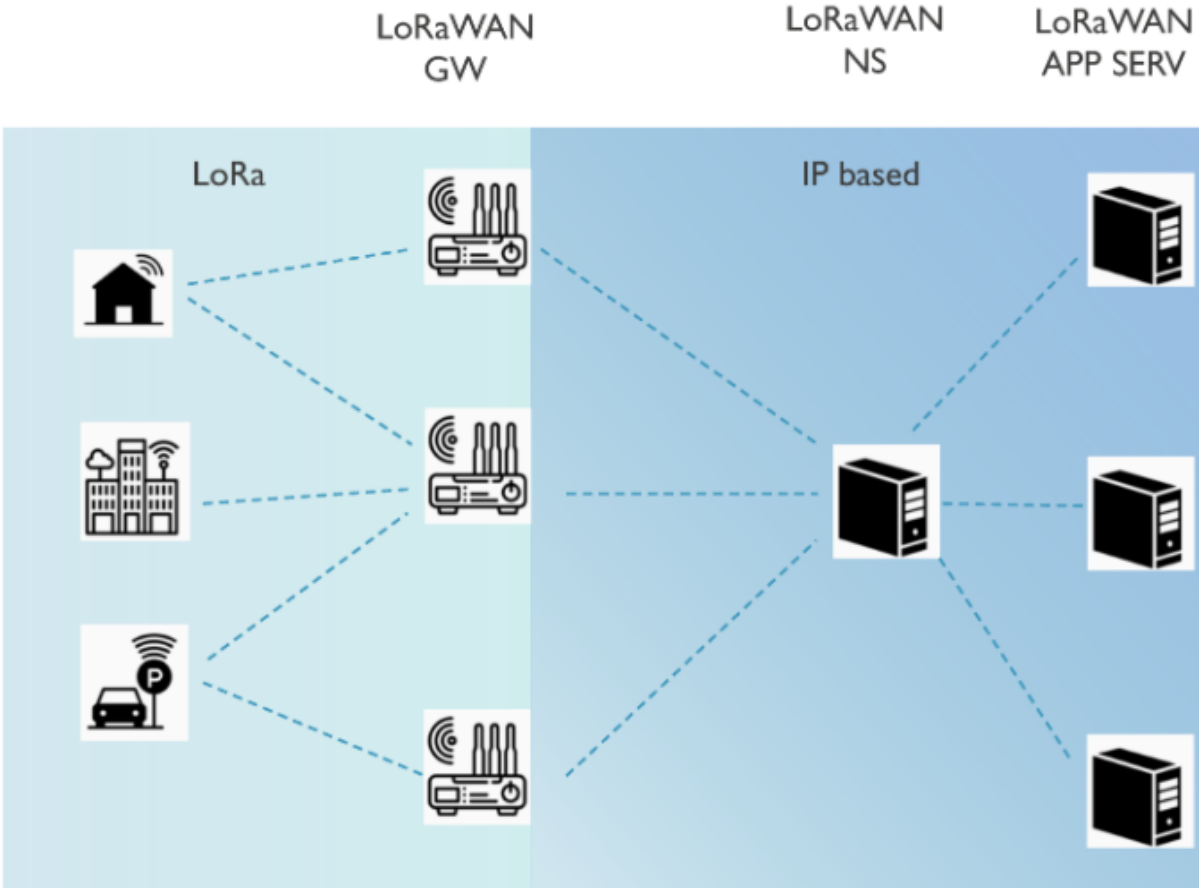

Figura 1.6: Topología de la red LoRaWAN

Para evitar el plagio, puedes parafrasear el texto, manteniendo la información esencial pero expresándola con tus propias palabras. Aquí tienes una versión parafraseada del párrafo:

Los dispositivos finales en una red LoRaWAN se clasifican en clases A a C. Los dispositivos de Clase A tienen un conjunto de opciones básicas que deben implementar para unirse a la red LoRaWAN. Estos dispositivos permiten la comunicación bidireccional: cada transmisión de enlace ascendente es seguida por dos breves ventanas de recepción de enlace descendente, durante las cuales el dispositivo escucha posibles transmisiones de enlace descendente. Por lo tanto, la comunicación descendente es activada por el dispositivo final, lo que significa que cada mensaje descendente debe esperar una transmisión ascendente. Las primeras y segundas ventanas de recepción del enlace descendente comienzan 1 y 2 segundos, respectivamente, después de la transmisión ascendente. Si la transmisión descendente ocurre durante la primera ventana, se usa el mismo canal que para la transmisión

ascendente, y la tasa de propagación (SF) se determina por el parámetro RX1DROffset. Si la transmisión descendente se realiza en la segunda ventana, se utilizan un SF y un canal fijos, normalmente el canal de 125 kHz centrado en 869,525 MHz con SF12, un ciclo de trabajo del 10 % y una alta potencia de transmisión de 24 dBm. Es responsabilidad del servidor de red programar el tráfico descendente en el momento exacto y gestionar la sincronización. Los dispositivos de Clase A son los más eficientes en términos de consumo de energía, ya que permanecen inactivos la mayor parte del tiempo [21].

LoRaWAN incorpora un mecanismo de velocidad de datos adaptable (ADR) para gestionar dinámicamente los parámetros de enlace de un nodo final con el fin de aumentar la tasa de entrega de paquetes. El mecanismo ADR gestiona la velocidad de datos y la potencia de transmisión de los dispositivos finales. Si el dispositivo desea permitir que el servidor de red administre sus parámetros de transmisión, configurará el bit ADR de enlace ascendente en sus paquetes de comunicación de enlace ascendente. De lo contrario, el dispositivo final tiene la posibilidad de gestionar él mismo sus parámetros de transmisión haciendo uso del mecanismo ADR que reside en el lado del dispositivo final. Por tanto, ambas partes del mecanismo ADR se ejecutan de forma asincrónica en el servidor de red y en el nodo final [21].

1.1.7. Angular

Angular es un framework que emplea HTML y TypeScript para desarrollar aplicaciones de una sola página en el cliente. Este framework utiliza TypeScript como su lenguaje principal. Las bibliotecas de TypeScript que se importan en las aplicaciones proporcionan tanto funcionalidades básicas como adicionales. La estructura de una aplicación Angular se sustenta en conceptos clave. Los NgModules, que ofrecen un contexto de compilación para los componentes, son los elementos fundamentales. Estos NgModules agrupan código relacionado en conjuntos funcionales, y un conjunto de NgModules define una aplicación Angular. Siempre hay al menos un módulo raíz que inicia la aplicación, aunque normalmente hay varios módulos adicionales que aportan diferentes funcionalidades. [22].

1.1.8. Bases de datos

Para el almacenamiento estructurado de datos, las bases de datos son el método preferido [23].

La información se guarda en una base de datos y se conecta a una unidad lógica junto con los metadatos necesarios para su procesamiento. Las bases de datos son herramientas muy útiles para gestionar grandes volúmenes de datos y realizar consultas. Además, en muchas situaciones, se puede definir un esquema de permisos que especifica qué personas o programas pueden acceder a los datos, con el objetivo de presentar el contenido de manera adecuada y comprensible. Los sistemas de bases de datos varían conceptualmente, por lo que cada uno tiene sus propios beneficios y desventajas [24].

NoSQL

El término NoSQL se utilizó por primera vez en 1998 para una base de datos que no tenía una interfaz SQL. Adquirió una importancia creciente durante la década de 2000, especialmente con la rápida expansión de Internet. La creciente popularidad de los servicios web globales provocó un aumento en el uso de bases de datos a escala web, ya que existía la necesidad de sistemas de gestión de datos que pudieran manejar las enormes cantidades de datos (a veces en el rango de petabytes y más) generados por los servicios web [25].

Estos nuevos sistemas de bases de datos, por definición, no son relacionales, por lo que no pueden utilizar la funcionalidad SQL completa. Además, a diferencia de las bases de datos relacionales, comparten coherencia y seguridad debido a la escalabilidad y el rendimiento. [26].

1.2. Node-Red

Node-RED es un entorno de desarrollo basado en JavaScript de código abierto basado en Node.js que los ingenieros de IBM crearon para que sea más apropiado para la creación de sistemas de Internet de las cosas. En pocas palabras, es un ambiente de programación virtual basado en procesos [27]. que conecta hardware

y software para generar "flujos de datos"desde el sensor a la nube. Es apto para escribir procesos de datos, lo que facilita los datos. Procesamiento Se puede utilizar fácilmente para compilar la lógica de procesamiento de datos y transferir los datos procesados a sistemas de nivel superior (servidor SQL, gestión de sistemas empresariales, recopilador central de datos y servicios en la nube) en minutos o mostrarlos inmediatamente. En lugar de programar una página web que muestre diferentes datos de sensores, Node-RED proporciona una interfaz de panel que le permite crear un espectacular interfaz sin la necesidad de conocimientos especiales de HTLM o CSS [27].

1.3. Estudios Relacionados

Tabla 1.1: Resumen de los trabajos relacionados con Detección temprana de incendios mediante IoT

Artículo	**Problema de IoT**					**Restricciones en el diseño**					**Propósito**			
Autor	Eficiencia energética	Gestión ambiental	Conectividad a larga distancia	Seguridad de zonas	Vigilancia del entorno	Seguridad de los datos	Interoperabilidad	Consumo energético	Costo de implementación y mantenimiento	Conectividad	**Automatización y optimización de procesos**	Mejora de la calidad del ambiente	Optimización de recursos y sostenibilidad	Mejora de la toma de decisiones
Mehta, 2021 [28]		✠		✠	✠		✠		✠	✠		✠		✠
Nagolu, 2023 [29]		✠		✠					✠		✠	✠		✠
Kumari, 2022 [30]		✠		✠	✠		✠		✠	✠	✠	✠		
Kumar, 2022[31]	✠				✠		✠		✠			✠	✠	✠
Venkatesan, 2023 [32]			✠		✠	✠		✠	✠		✠		✠	
Ishitha, 2021 [33]		✠		✠	✠	✠			✠	✠	✠	✠		✠
Dasari, 2020 [34]		✠	✠	✠	✠				✠	✠	✠	✠		✠
Ananthi, 2022 [35]		✠	✠	✠			✠			✠		✠	✠	✠
Khennou, 2021 [36]	✠	✠		✠			✠			✠		✠	✠	
Este trabajo	✠	✠	✠	✠	✠		✠		✠			✠		✠

Metha [28]emplea una conexión WiFi para la transmisión de datos, lo que resulta en una comunicación ineficiente y costosa, además de limitada en cuanto a distancia, con un alcance máximo de 100 metros. En segundo lugar, Nagolu, al igual que Metha, utiliza WiFi para la transmisión de datos, pero en este caso, se utiliza para el control de un sector en una fábrica, lo que implica que, si bien se emplea IoT, no se utiliza para el control ambiental.

Por otro lado, Kumari [30] utiliza WiFi para conectarse a su base de datos en la detección temprana de incendios forestales. Sin embargo, esta conexión resulta costosa y limitada en alcance, además de consumir una cantidad significativa de energía. A pesar de estas limitaciones, al igual que los otros proyectos mencionados, Kumari busca automatizar procesos, optimizar recursos y mejorar la toma de decisiones.

Adicionalmente, Venkatesan [32] utiliza tecnología GSM para lograr una conectividad a larga distancia en el control de un lugar de trabajo, ofreciendo además seguridad de datos. Aunque este enfoque presenta un alto costo de implementación, el proyecto busca automatizar procesos y optimizar recursos.

Ishitha [33], por su parte, desarrolla un proyecto destinado a la protección del ambiente, utilizando tecnología WiFi y enfrentando los mismos desafíos de costo y alcance mencionados anteriormente. Sin embargo, se destaca por su énfasis en la seguridad para restringir el acceso solo al personal autorizado, con un costo de implementación relativamente bajo. Este proyecto tiene como objetivo automatizar procesos, mejorar el ambiente y facilitar la toma de decisiones.

En el caso de Dasari [34], su proyecto se centra en la detección temprana de incendios forestales, utilizando WiFi y enfrentando los mismos desafíos mencionados anteriormente. Además, la inclusión de una carcasa costosa dificulta aún más su implementación.

Anathi [35], utiliza Deep Learning para la detección de incendios forestales, añadiendo un valor adicional a su proyecto. Sin embargo, al emplear WiFi para la transmisión de datos, se enfrenta a los mismos problemas de alcance y costos mencionados anteriormente.

Finalmente, Khennou [36] utiliza LoRaWAN, lo que proporciona una comunicación a larga distancia a bajo costo y con una eficiencia energética notable.

Además, aprovecha el Deep Learning para mejorar la toma de decisiones y optimizar recursos.

Capítulo 2

Capítulo II: Diseño

El diseño de un prototipo efectivo para la detección temprana de incendios forestales implica la integración de múltiples elementos, desde la selección de sensores adecuados hasta el diseño de la carcasa y la interfaz de usuario integrada con la base de datos. Este capítulo se centra en el proceso de diseño del sistema IoT, abordando aspectos clave como la selección de componentes, la integración entre base de datos-página web y la usabilidad del dispositivo.

El diagrama básico de conexión entre las diferentes partes del prototipo se muestra a continuación en la figura 2.1:

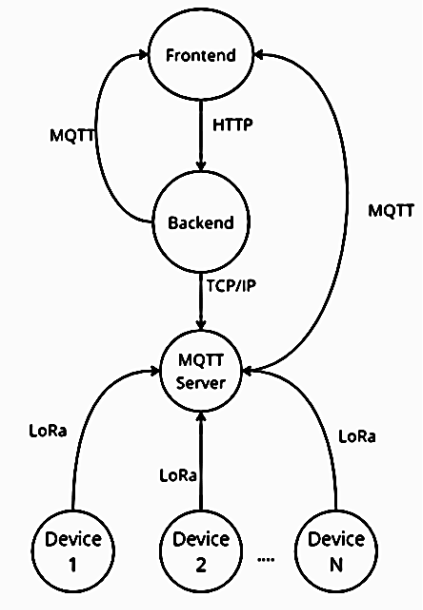

Figura 2.1: Diagrama de conexión

2.1. Diseño electrónico del prototipo

Para el diseño físico de la parte electrónica, se consideró que los sensores utilizan diferentes tipos de señales para la transmisión de la información. Por ejemplo, el sensor de temperatura y humedad (SHTC3) usa transmisión I2C, un protocolo de comunicación serial utilizado principalmente para comunicar microcontroladores con periféricos. De igual manera, el sensor de calidad del aire (MQ135) puede transmitir datos tanto de manera analógica como digital; sin embargo, para este proyecto se optó por el puerto analógico. Por último, el módulo GPS (TinyGPS++) emplea el protocolo de comunicación serial UART, un protocolo asíncrono basado en dos líneas principales: TX y RX.

Considerando los detalles proporcionados sobre los datos transmitidos, a continuación se presentan los pines GPIO utilizados:

- **SHTC3** - Puerto SDA: GPIO 41 y Puerto SCL: GPIO 42
- **MQ135** - Puerto Analógico: GPIO 7
- **GPS** - Puerto TX: GPIO 45 y RX: GPIO 46

El esquema general de conexiones para la placa se puede observar en la figura 2.2.

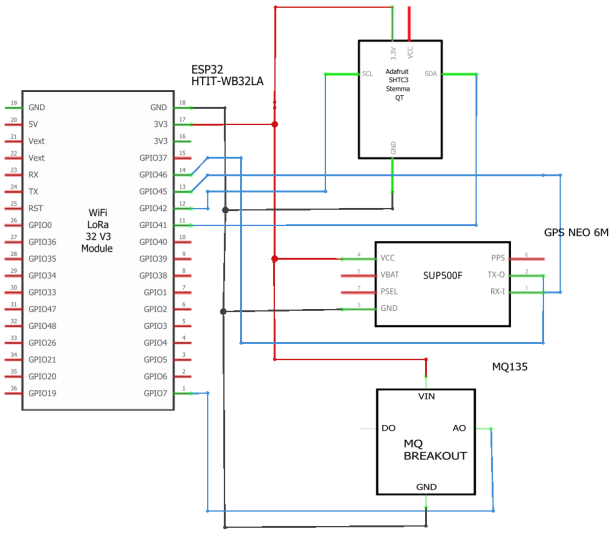

Figura 2.2: Esquema general de conexiones para prototipo

Generado el diseño esquemático de conexiones del ESP32, se procedió a crear un PCB para controlar la fuente de voltaje destinada a los sensores y el GPS. Esto se debe a que el ESP32 no cuenta con suficientes salidas de voltaje de 3.3V para alimentar todos los componentes. Por lo tanto, se decidió dividir una fuente para que pudiera suministrar energía a varios sensores simultáneamente, en este caso al SHTC3, MQ135 y GPS NEO6M. En la figura 2.3, se puede observar el diseño esquemático de la fuente, incluyendo la ubicación de los pines macho que se colocaron.

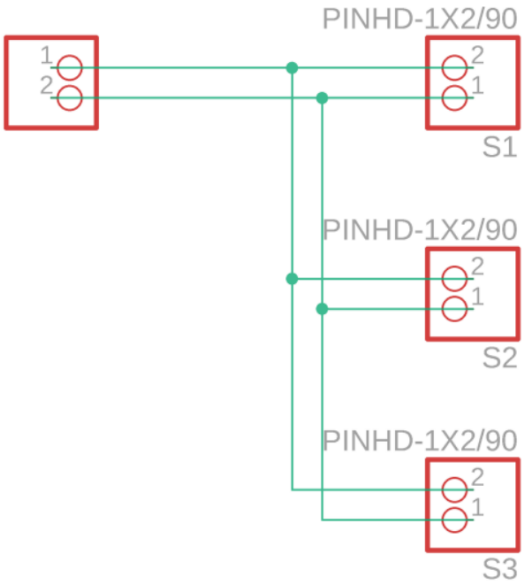

Figura 2.3: Diseño esquemático de la fuente

En la figura 2.4, se presenta el diseño final de la placa PCB, que considera las normativas adecuadas con un ancho de 10 mil, suficiente para soportar la corriente que se manejará en la placa.

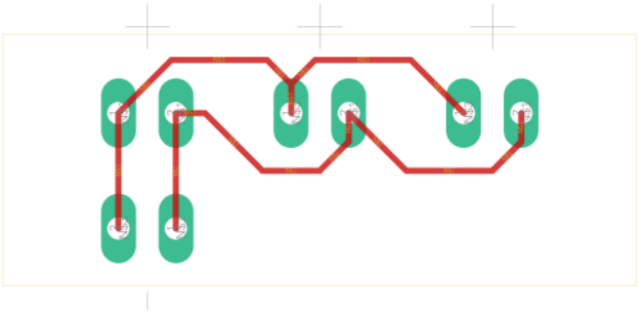

Figura 2.4: Diseño PCB de la fuente

Continuando con la parte física, una vez diseñadas las conexiones e

independizado la fuente de voltaje para los diferentes sensores, procedemos a explicar la sección de diseño para la independencia de energía, es decir, que sea autosustentable, dependiente de su batería y de una fuente externa. Para ello, se utilizó el módulo TP4056, un cargador para baterías de litio LiPo y Li-ion de una celda, con protección integrada y una corriente de carga de 1A. Su voltaje de entrada es de 4.5V a 5.5V y sus fases de carga se indican mediante LEDs [37]. En conjunto, se empleó un panel solar monocristalino con una capacidad de salida de 5V y 250mA, y una potencia de 1.25W, ofreciendo un rendimiento confiable con buena iluminación [38]. El uso combinado de estos componentes se debe a que corresponden a valores adecuados para el funcionamiento del sistema.

La función de este sistema es que, mientras haya luz solar, cargue la batería y suministre un valor constante de voltaje en la salida. Cuando no haya luminosidad, la batería se utilizará de manera directa como salida del módulo, generando un sistema de carga y descarga de la batería, pero con una alimentación constante. La figura 2.5, muestra el diseño del sistema de alimentación del prototipo para la detección de incendios.

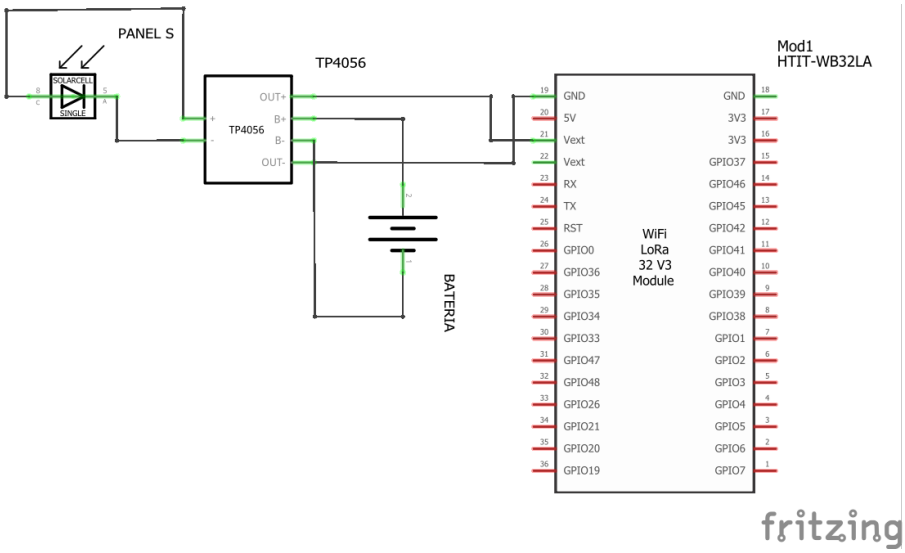

Figura 2.5: Esquema de independencia de energía

El diagrama del programa realizado se observa en la siguiente figura 2.6.

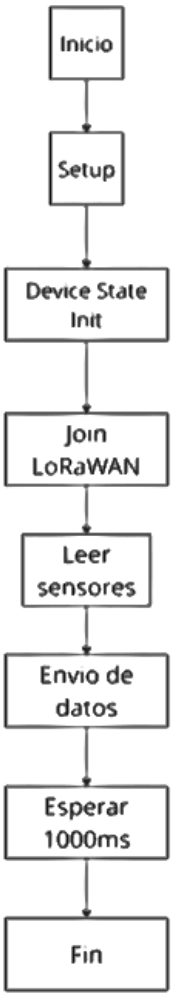

Figura 2.6: Diagrama programa electrónico

El siguiente código está diseñado para ejecutarse en un microcontrolador como un ESP32 - WiFi LoRa 32(V3) utilizando el entorno de desarrollo de Arduino. Este código realiza la lectura de datos de sensores de temperatura y humedad (SHTC3), calidad del aire (MQ135) y GPS (TinyGPS++), así como también configura el modulo LORA que posee el microcontrolador para la transmisión de los datos. Vamos a desglosarlo y explicar cada sección.

Primero, se incluyen las librerías necesarias para los sensores, el GPS y el módulo LoRa: Arduino.h [39] funciones básicas de Arduino, SHTC3.h [40] para el sensor de temperatura y humedad, MQ135.h [41] para el sensor de gas, TinyGPS++.h [42] para el GPS, LoRaWanAPP.h [43] para la comunicación LoRa, y Wire.h [44] para la comunicación I2C. Luego, se define el pin para el sensor MQ135 (PINMQ135). Finalmente, se crean instancias de los objetos necesarios para interactuar con los

sensores y el GPS.

```
#include <Arduino.h>
#include "SHTC3.h"
#include <MQ135.h>
#include <TinyGPS++.h>
#include "LoRaWan_APP.h"
#include "Wire.h"
#define PIN_MQ135 18
SHTC3 s(Wire);
MQ135 mq135_sensor(PIN_MQ135);
TinyGPSPlus gps;
```

Luego se definen los identificadores y claves OTAA y ABP para las conexiones LoRaWAN. Las configuraciones de LoRaWAN incluyen la región, la clase de dispositivo, el ciclo de transmisión, la activación de OTAA/ABP, la habilitación de ADR, el tipo de mensaje (confirmado o no confirmado) y el puerto de la aplicación. Además, confirmedNbTrials indica la cantidad de intentos que se deben realizar para transmitir el mensaje en caso de que no se reciba una confirmación.

```
uint8_t devEui[] = { 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x06, 0x53, 0xC8 };
uint8_t appEui[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t appKey[] = { 0x74, 0xD6, 0x6E, 0x63};
uint16_t userChannelsMask[6] = { 0x00FF, 0x0000, 0x0000, 0x0000};
LoRaMacRegion_t loraWanRegion = ACTIVE_REGION;
DeviceClass_t loraWanClass = CLASS_A;
uint32_t appTxDutyCycle = 15000;
bool overTheAirActivation = true;
bool loraWanAdr = true;
bool isTxConfirmed = true;
uint8_t appPort = 2;
uint8_t confirmedNbTrials = 4;
```

Para luego inicializar la comunicación I2C, se toman las muestras de los sensores. Se leen los valores de temperatura, humedad y gas (con sus correcciones) y se almacenan en las variables correspondientes. También se leen los datos del GPS (latitud y longitud). Los valores obtenidos se empaquetan en el arreglo appData para ser enviados por LoRaWAN.

```
static void prepareTxFrame(uint8_t port) {
  float temp = s.readTempC();
  float hum = s.readHumidity();
  float rzero = mq135_sensor.getRZero();
  float correctedRZero = mq135_sensor.getCorrectedRZero(temp, hum);
  float resistance = mq135_sensor.getResistance();
  float ppm = mq135_sensor.getPPM();
  float correctedPPM = mq135_sensor.getCorrectedPPM(temp, hum);
  while (Serial2.available() > 0)
  unsigned char *puc;
  appDataSize = 0;
  puc = (unsigned char *)(&temp);
  for (int i = 0; i < 4; i++) appData[appDataSize++] = puc[i];
  puc = (unsigned char *)(&hum);
  for (int i = 0; i < 4; i++) appData[appDataSize++] = puc[i];
  puc = (unsigned char *)(&correctedPPM);
  for (int i = 0; i < 4; i++) appData[appDataSize++] = puc[i];
  puc = (unsigned char *)(&lat);
  for (int i = 0; i < 4; i++) appData[appDataSize++] = puc[i];
  puc = (unsigned char *)(&lng);
  for (int i = 0; i < 4; i++) appData[appDataSize++] = puc[i];
  Wire.end();
}
```

En la siguiente porción de código, se inicia la comunicación serial a 115200 baudios para enviar datos al ordenador y recibir posibles comandos. También se inicia Serial2 a 9600 baudios para comunicarse con el módulo GPS, utilizando los

pines 46 (RX) y 45 (TX) del microcontrolador. Finalmente, se inicializa la MCU con los parámetros especificados.

```
void setup() {
  Serial.begin(115200);
  Serial2.begin(9600, SERIAL_8N1, 46, 45);
  Mcu.begin(HELTEC_BOARD, SLOW_CLK_TPYE);
}
```

Finalmente, en el bucle loop(), se maneja el estado del dispositivo utilizando una máquina de estados. En DEVICESTATEINIT, se inicializa LoRaWAN y se configura el DR (Data Rate) predeterminado. En DEVICESTATEJOIN, se realiza el proceso de unión a la red LoRaWAN. En DEVICESTATESEND, se prepara el frame de datos y se envía a través de LoRaWAN. En DEVICESTATECYCLE, se programa la transmisión del siguiente paquete, ajustando el ciclo de transmisión. En cualquier otro estado, el dispositivo se reinicia al estado de inicialización.

```
  switch (deviceState) {
    case DEVICE_STATE_INIT:
      LoRaWAN.init(loraWanClass, loraWanRegion);
      LoRaWAN.setDefaultDR(3);
      break;
    case DEVICE_STATE_JOIN:
      LoRaWAN.join();
      break;
    case DEVICE_STATE_SEND:
      prepareTxFrame(appPort);
      LoRaWAN.send();
      break;
    default:
      deviceState = DEVICE_STATE_INIT;
      break;}}
```

Basándonos en el trabajo realizado por Almeida [45], se configuró el LoraWan Gateway E890-915LG12, el alcance de el gateway es de 3Km.

2.2. Diseño de la carcasa física del prototipo

Para el apartado de diseño de la carcasa física, se trabajó por secciones utilizando Inventor para el diseño completo y el ensamblado, el cual fue impreso en 3D. La carcasa principal contendrá tanto la parte electrónica como los sensores para las mediciones. La caja completa, incluyendo las tapas, posee unas dimensiones de 79.55x74.20x87. Este diseño está generado con soportes que permiten la sujeción segura de la caja, juntamente con un ajuste entre la tapa superior y la caja principal. En la figura 2.7, se muestra la carcasa principal del diseño, que posee un compartimiento separado para la batería y la parte electrónica. Los agujeros están diseñados para tornillos de 3 mm que permiten un ajuste adecuado. Se implementó también un espacio para un interruptor de encendido y apagado, así como un espacio para el puerto SMA correspondiente a la antena del microcontrolador para el uso de LoRaWAN. En la parte inferior se diseñaron rejillas que permiten el paso del aire, facilitando una medición adecuada del entorno. En ese mismo espacio se colocarán los sensores, y se diseñaron agujeros para la sujeción de los sensores, también de 3 mm.

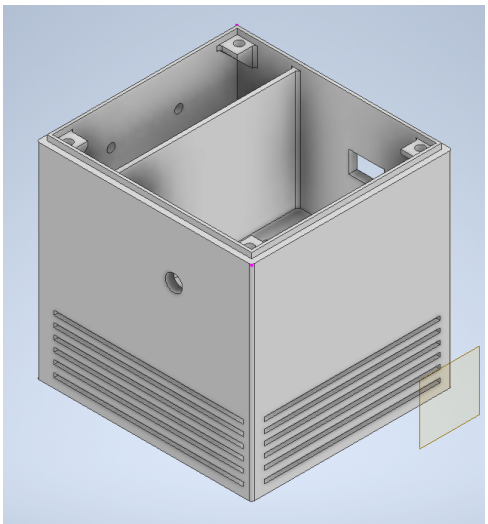

Figura 2.7: Diseño carcasa principal

Las dimensiones de la carcasa principal se muestran a continuación en la figura 2.8.

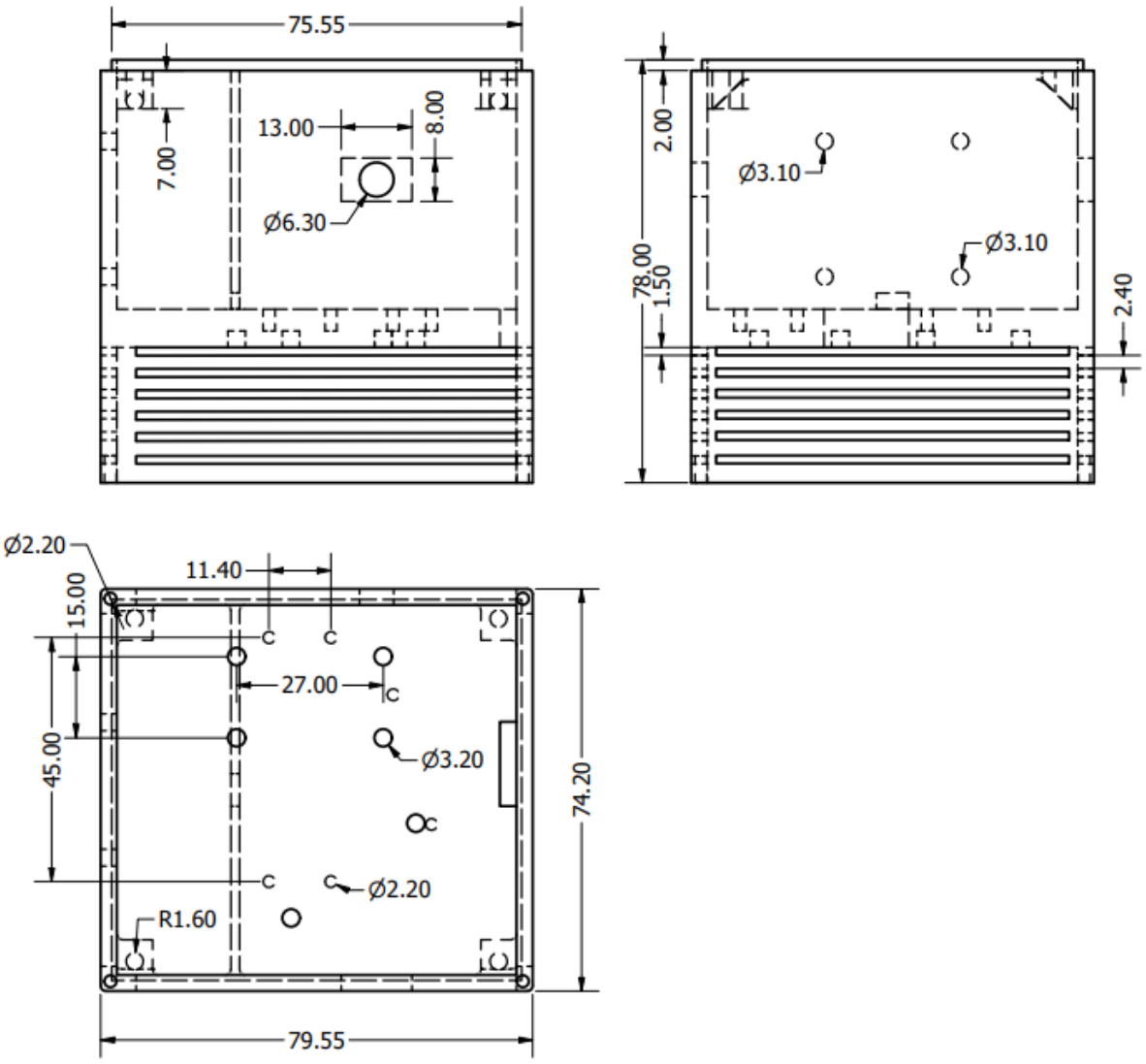

Figura 2.8: Medidas carcasa principal

En la figura 2.9, se muestra el diseño de las tapas inferior y superior. La figura 2.9a muestra una tapa simple sin sujeción, con agujeros de 2 mm para el soporte de la tapa de los sensores, con las mismas dimensiones de la caja principal: 79.55x74.20x3mm. En cambio, la figura 2.9b muestra el diseño de la tapa superior con ajuste, que permite, además del soporte de los tornillos, un soporte adicional de la misma caja. Esta tapa posee agujeros de 3 mm para los tornillos y un soporte central para la pieza de sujeción del panel solar. Las dimensiones son 79.55x74.20x6mm.

(a) Tapa inferior de carcasa (b) Tapa superior de carcasa

Figura 2.9: Diseño de tapas para la carcasa

Las dimensiones de las tapas superior e inferior se muestran a continuación en la figura 2.10.

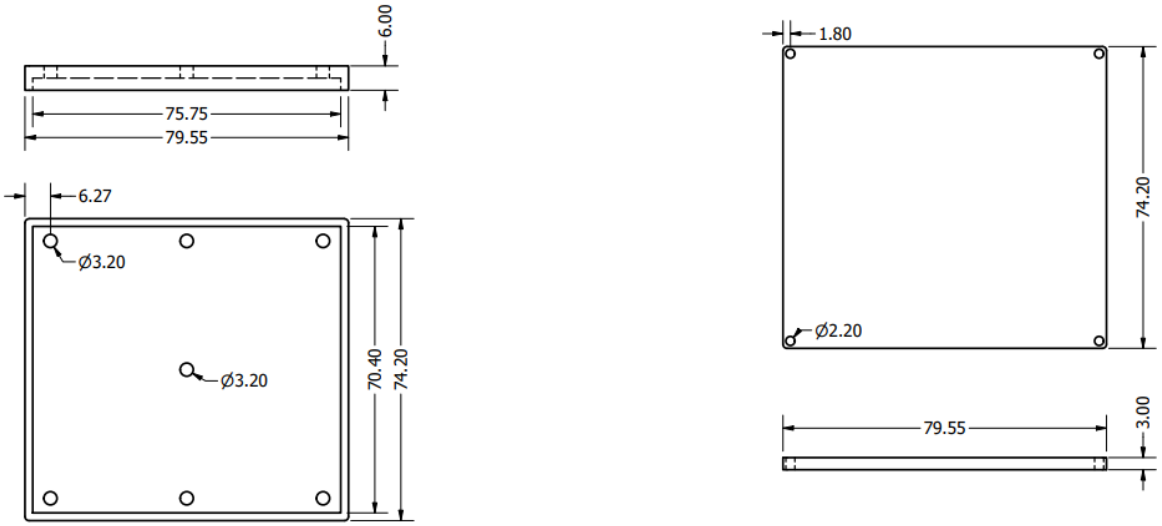

Figura 2.10: Medidas tapas

Seguidamente, en la figura 2.11, se puede observar el soporte del panel solar. Este también posee agujeros de 3 mm para los tornillos que conectan la tapa superior con esta pieza, así como un agujero en la parte superior de 3 mm para el cableado del panel solar. Las dimensiones de esta pieza son 14 x 74.2 x 80 mm.

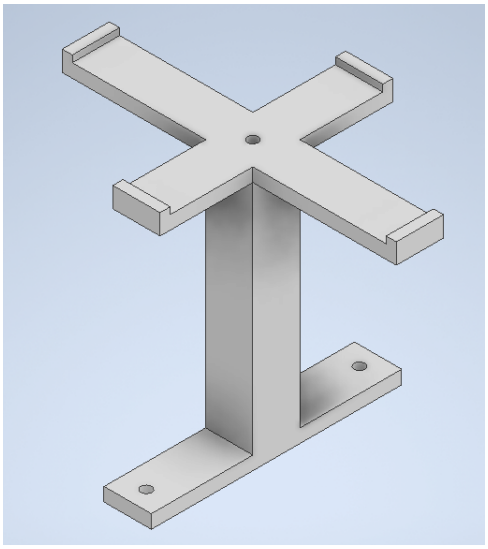

Figura 2.11: Diseño soporte para panel solar

Las dimensiones del soporte se muestra a continuación en la figura 2.12.

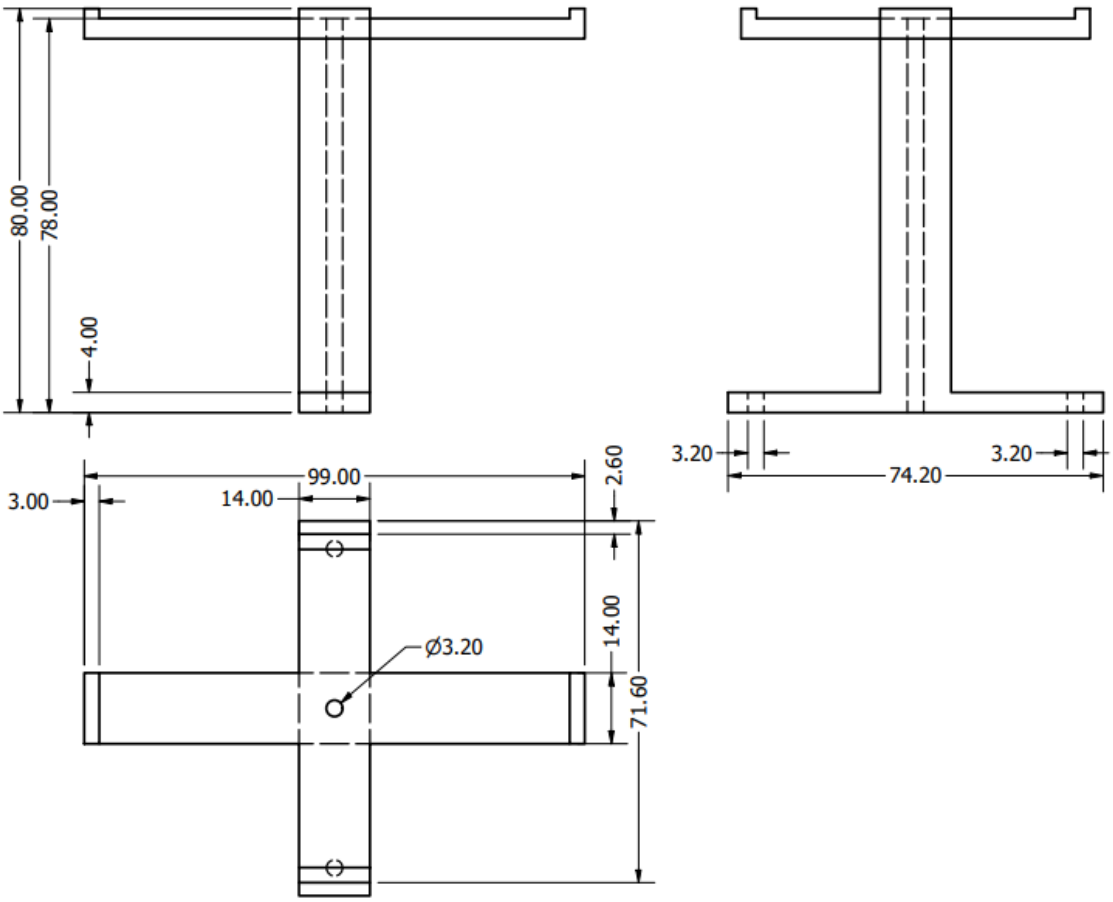

Figura 2.12: Medidas soporte principal

La figura 2.13 muestra la última pieza, que corresponde a un conjunto de sujeción para el soporte de la caja, considerando, por ejemplo, un tubo. Esta pieza posee un soporte para la parte central de la caja con agujeros de 3 mm, juntamente con un soporte para la unión de ambas piezas, también con agujeros para tornillos de 3 mm. Las dimensiones de esta pieza son 44.92x42.07x30mm.

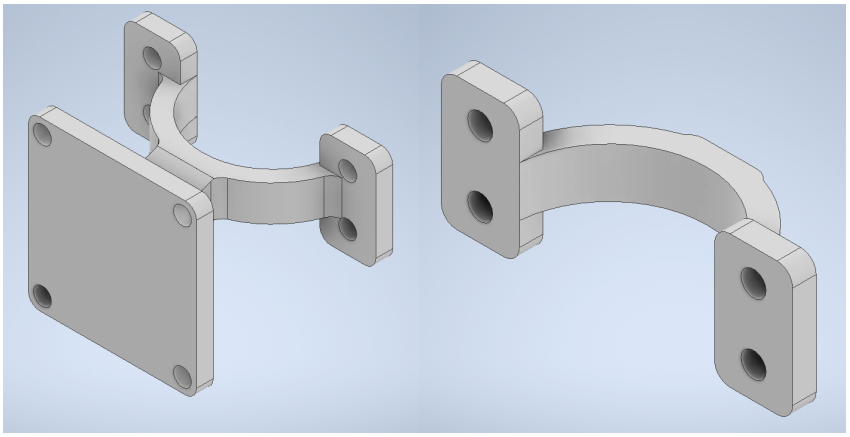

Figura 2.13: Diseño soporte principal para carcasa

Las dimensiones del conjunto de sujeción para el soporte de la caja se muestran a continuación en la figura 2.14.

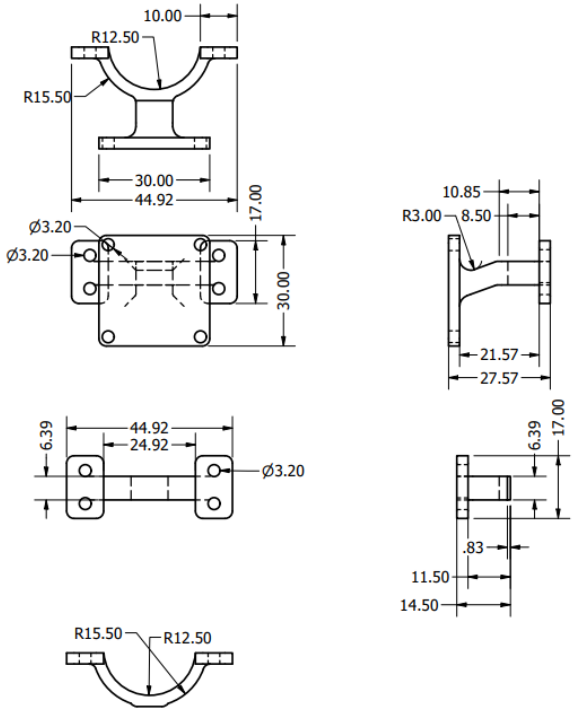

Figura 2.14: Medidas conjunto sujeción

Cabe aclarar que son piezas separadas, no un conjunto único, lo que permite que sean desmontables según la necesidad del usuario. Una vez visto el diseño individual de cada pieza, se puede obtener el diseño final como un ensamblaje general. En la figura 2.15, se muestra, a nivel de simulación, cómo quedaría la pieza final.

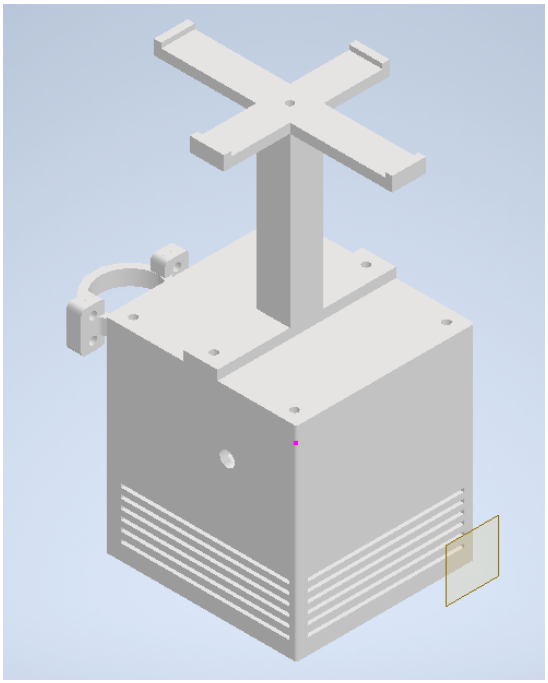

Figura 2.15: Ensamblaje del prototipo para la detección de incendios

2.3. Diseño de la página web

Para el diseño de la página web se uso el framework Angular, en este caso la parte de diseño de la página web se realizó en HTML el cual soporta Angular, por otro lado, la parte de funcionalidades la página web se realizó con TypeScript.

El diagrama que explica el funcionamiento del frontend se observa en la siguiente figura 2.16.

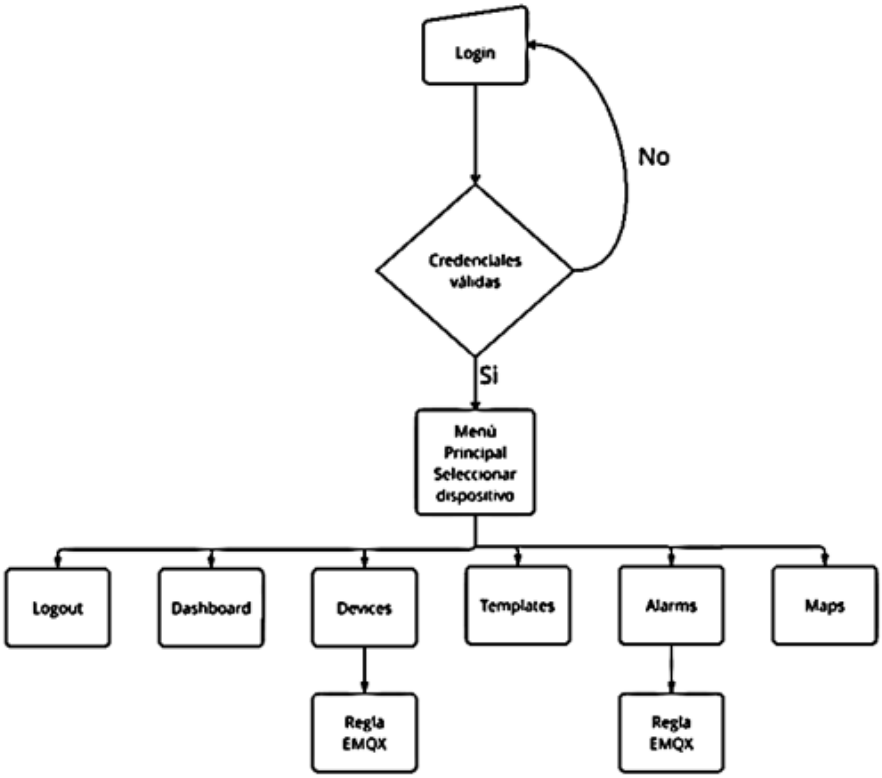

Figura 2.16: Diagrama funcionamiento frontend

En principio creamos los componentes el primero de ellos fue el de los devices los sirven para controlar todos los dispositivos en nuestra red de sensores.

El siguiente código define un componente en Angular llamado DevicesComponent. En primer lugar, se realizan varias importaciones esenciales: Component de @angular/core para definir el componente Angular, IDevice y ITemplates de un archivo de modelos que definen la estructura de los datos, y dos servicios, PeticionesHttpService y UiServiceService, que manejan peticiones HTTP y operaciones de UI, respectivamente.

```
import { Component } from '@angular/core';
import { IDevice, ITemplates } from 'src/app/services/clases.models';
import { PeticionesHttpService } from 'src/app/peticiones-http.service';
import { UiServiceService } from 'src/app/services/ui-service.service';
```

El @Component configura el componente DevicesComponent con un selector app-devices, que se utiliza para insertar este componente en una plantilla HTML.

Además, especifica la URL de la plantilla HTML y un array de estilos, que en este caso está vacío, pero puede contener estilos específicos para el componente.

```
@Component({
  selector: 'app-devices',
  templateUrl: './devices.component.html',
  styles: []
})
```

Dentro de la clase DevicesComponent, se definen varias propiedades: templates es un array que almacenará las plantillas, device es un objeto que representa un dispositivo con valores iniciales, templateSelected es un índice del template seleccionado, inicializado a -1 para indicar que no hay ninguna plantilla seleccionada, y devices es un array que contiene dispositivos iniciales con algunos valores predefinidos.

```
export class DevicesComponent {
  templates: ITemplates[] = [];
  device: IDevice = {name: '', dId: '', saverRule: {userId: '', dId: ''}};
  templateSelected = -1;
  devices: IDevice[] = [
    {name: "Home", dId: "5551", password: "z87654321"},
    {name: "Office", dId: "5552", password: "z87654321"},
    {name: "Facilities", dId: "5553", password: "z87654321"}
  ];
```

Además, dentro de la clase DevicesComponent, se definen varias propiedades: Templates es un array que almacenará las plantillas, device es un objeto que representa un dispositivo con valores iniciales, templateSelected es un índice del template seleccionado, inicializado a -1 para indicar que no hay ninguna plantilla seleccionada, y devices es un array que contiene dispositivos iniciales con algunos valores predefinidos.

```
export class DevicesComponent {
  templates: ITemplates[] = [];
  device: IDevice = {name: '', dId: '', saverRule: {userId: '', dId: ''}};
  templateSelected = -1;
  devices: IDevice[] = [
    {name: "Home", dId: "5551", password: "z87654321"},
    {name: "Office", dId: "5552", password: "z87654321"},
    {name: "Facilities", dId: "5553", password: "z87654321""}
  ];
```

El constructor de la clase inyecta dos servicios, PeticionesHttpService y UiServiceService, que se utilizarán para realizar operaciones HTTP y manejar la interfaz de usuario, respectivamente.

```
constructor(private _peticionesHttp: PeticionesHttpService,
private _uiServices: UiServiceService) {}
```

El método ngOnInit se ejecuta después de que el componente se inicializa y se encarga de obtener las plantillas y dispositivos desde el servidor, utilizando los métodos readTemplates y getDevices del servicio PeticionesHttpService.

```
ngOnInit(): void {
  this._peticionesHttp.readTemplates().then(templates1 => {
    this.templates = templates1;
  });
  this.devices = [];
  this._peticionesHttp.getDevices().then(devices => {
    console.warn(devices);
    this.devices = devices;
  });
}
```

El componente devices incluye varios métodos para gestionar los dispositivos. El método deleteDevice elimina un dispositivo de la lista de dispositivos en una posición específica. Llama al método deleteDevice del servicio HTTP (peticionesHttp) y, si la respuesta es positiva, elimina el dispositivo de la lista local. Si la eliminación falla, muestra un mensaje de error utilizando el servicio de UI (uiServices).

```
deleteDevice(pos: number) {
  console.log('Selected Device is:', pos);
  this._peticionesHttp.deleteDevice(this.devices[pos])
    .then(resp => {
      if(resp) this.devices.splice(pos, 1);
    });
}
```

El método limpiarDatoDevice restablece los datos del objeto device a valores vacíos y establece templateSelected en -1, indicando que no hay ninguna plantilla seleccionada.

```
limpiarDatoDevice() {
  this.device.name = '';
  this.device.dId = '';
  this.templateSelected = -1;
}
```

El método addDevice añade un nuevo dispositivo a la lista si los datos proporcionados son válidos. Primero, verifica que device.name, templateSelected y device.dId no estén vacíos. Luego, asigna valores al dispositivo desde la plantilla seleccionada y llama al método addDevice del servicio HTTP. Si la respuesta es positiva, actualiza la lista de dispositivos y limpia los datos del dispositivo actual.

```
addDevice() {
  if ((this.device.name.length > 0)
  && (this.templateSelected >= 0)
  && this.device.dId.length > 0) {
    this._peticionesHttp.addDevice(this.device,
    this.templates[this.templateSelected])
      .then(async respok => {
        if (respok) {
          this.device.templateName =
          this.templates[this.templateSelected].name;
          await this.getDevices();
          this.limpiarDatoDevice();
        }});}}
```

El método updateSaverRule actualiza el estado de la regla de guardado (saverRule) de un dispositivo específico. Si el dispositivo no tiene una saverRule, muestra un mensaje informativo. De lo contrario, llama al método updateSaverRuleStatus del servicio HTTP y, si la respuesta es positiva, invierte el estado de saverRule del dispositivo.

```
updateSaverRule(id: number) {
  const nameD = this.devices[id].name;
  then(resp => {
    if (resp) {
      console.log('Se modifico el dispositivo' + nameD);
      this.devices[id].saverRule!.status =

      !this.devices[id].saverRule!.status;
    } else {
      this.devices[id].saverRule);
    }
  });
}
```

El método getDevices obtiene la lista de dispositivos desde el servidor llamando al método getDevices del servicio HTTP. Almacena la lista obtenida en la propiedad devices y registra la lista en la consola.

```
async getDevices() {
  await this._peticionesHttp.getDevices().then(devices1 => {
    this.devices = devices1;
    console.log('getdevices():', this.devices);
  });
}
```

El método status devuelve el estado de la regla de guardado (saverRule) de un dispositivo específico. Si el dispositivo no tiene una saverRule, devuelve false. De lo contrario, devuelve el estado (status) de la saverRule.

```
status(id: number) {
  if (this.devices[id].saverRule === undefined) {
    return false;
  } else {
    return this.devices[id].saverRule!.status;
  }
}
```

El siguiente código HTML representa una interfaz de usuario para gestionar dispositivos en la aplicación Angular. A continuación, se describe este código:

Este bloque de código representa un formulario dentro de una tarjeta que permite agregar un nuevo dispositivo. Los campos del formulario incluyen el nombre del dispositivo, su ID y una selección de plantillas. Los datos están enlazados bidireccionalmente con las propiedades del componente Angular (device.name, device.dId y templateSelected). El botón .Add"llama al método addDevice cuando se hace clic.

```
<div class="main-content">
<div class="row">
<div class="col-md-12">
<div class="card">
<div class="card-header">
<h5 class="title">Add New Device</h5>
</div>
<div class="card-body">
<form>
<div class="row">
<div class="col-md-5 pr-1">
<div class="form-group">
<label>Device Name</label>
<input type="text" class="form-control">
</div>
</div>
<div class="col-md-3 px-1">
<div class="form-group">
<label>Id device</label>
<input type="text" class="form-control"[(ngModel)]="device.dId">
</div>
</div>
<div class="col-md-4 pl-1">
<div class="form-group">
<label for="email">Template</label>
<select class="form-select" [(ngModel)]="templateSelected">
<option  value="-1">Choose...</option
<option *ngFor="let template of templates; let i=index" value="{{i}}">
</select></div></div></div></form>
<div class="row pull-right">
<div class="col-12">
<button type="button" (click)="addDevice();">
Add
</button></div></div></div></div></div></div>
```

Este bloque de código muestra una lista de dispositivos en una tabla dentro de una tarjeta. La tabla contiene columnas para el nombre del dispositivo, ID, contraseña, plantilla y acciones. Las acciones incluyen un botón para eliminar el dispositivo y un interruptor para actualizar el estado de la regla de guardado (saverRule). Cada fila de la tabla se genera dinámicamente a partir del array devices utilizando la directiva *ngFor.

El botón "Delete" llama al método deleteDevice cuando se hace clic. El interruptor llama al método updateSaverRule y utiliza el método status para determinar si debe estar marcado o no.

```
<div class="row">
<div class="col-12">
<div class="card">
<div class="card-header">
<h5 class="card-category">All Devices List</h5>
<h4 class="card-title"> Devices Stats</h4>
</div>
<div class="card-body">
<div class="table-responsive">
<table class="table">
<tbody>
<tr *ngFor="let device of devices; let i=index">
<div class="row">
<div class="col" >
<button type="button"(click)="deleteDevice(i)" >
<i class="now-ui-icons ui-1_simple-remove"></i>
</div></div></div>
</td></tr></tbody></table>
</div></div></div></div></div></div>
```

A continuación, se muestra el código de un componente Angular llamado IotbuttonComponent, que se encarga de manejar un botón IoT. A continuación, se explica cada parte del código:

El @Component define el componente Angular. Especifica el selector (app-iotbutton), la plantilla HTML (./iotbutton.component.html) y un array vacío para los estilos CSS.

```
@Component({
  selector: 'app-iotbutton',
  templateUrl: './iotbutton.component.html',
  styles: []
})
```

El componente IotbuttonComponent tiene tres propiedades:
@Input() config: any. Esta propiedad se utiliza para recibir datos de un componente padre. La variable config puede ser de cualquier tipo (any). value = true. Inicializa la propiedad value a true. Esta propiedad podría representar el estado actual del botón (encendido/apagado). sending = false. Inicializa la propiedad sending a false. Esta propiedad podría indicar si un mensaje está siendo enviado actualmente.

```
export class IotbuttonComponent {
  @Input() config: any;
  value = true;
  sending = false;
  }
```

El constructor inyecta una dependencia del servicio EventBusService en el componente, que se almacena en la propiedad eventBus.

```
constructor(private _eventBus: EventBusService) {}
```

El método getIconColor devuelve un color para el icono del botón. Llama a getIconColor del servicio eventBus y le pasa dos parámetros: una condición booleana (this.sending || !this.value) y la configuración (this.config).

```
getIconColor(): string {
  return this._eventBus.getIconColor(this.sending || !this.value, this.config);
}
```

El método sendValue construye un objeto toSend que contiene un tema (topic) y un mensaje (msg). El tema se construye concatenando userId, dId del dispositivo seleccionado, la variable, y una cadena fija (/acdata). El mensaje contiene un valor (value) tomado de config.message. Luego, el método emite un evento (mqtt-sender) a través del servicio eventBus, pasando el objeto toSend.

```
sendValue() {
  const toSend = {
    topic: this.config.userId + "/" +
    this.config.selectedDevice.dId + "/"
    + this.config.variable + "/acdata",
    msg: {
      value: this.config.message
    }
  }
  this._eventBus.emit('mqtt-sender', toSend);
}
```

El código HTML para el componente IotbuttonComponent, el cual se encarga de renderizar un botón IoT con ciertas propiedades y estilos. A continuación, se explica cada parte del código:

El encabezado de la tarjeta (div con slot="header") tiene una clase text-center para centrar el texto. El título de la tarjeta (h4 con clase card-title) se genera dinámicamente utilizando las propiedades name del dispositivo seleccionado y variableFullName desde el objeto config.

```
<div slot="header" class="text-center">
    <h4 class="card-title">{
    {{config.variableFullName}}</h4>
</div>
```

El cuerpo de la tarjeta (div con class= card-body) también tiene la clase text-center para centrar su contenido y un estilo en línea que alinea el contenido al centro.

```
<div class="card-body" class="text-center"
style="align-content: center;">
```

El icono utiliza clases dinámicas para definir su estilo y color:
getIconColor() llama al método getIconColor del componente para obtener el color del icono. El estilo en línea establece el tamaño de fuente a 30px y la alineación vertical al medio.

```
<i class="fa {{config.icon}} {{getIconColor()}}"
style="font-size: 30px;
vertical-align: middle;">  </i>
```

El botón (div con la clase btn btn-outline-secondary) se renderiza con el texto definido en config.text. Al hacer clic en el botón, se llama al método sendValue del componente. La clase del botón (type=config.class) y su texto (config.text) se establecen dinámicamente desde config.

```
<div class="btn btn-outline-secondary"
(click)="sendValue()"
type="config.class">{{config.text}}</div>
```

El siguiente código describe un componente Angular llamado IotindicatorComponent, el cual se utiliza para mostrar un indicador IoT. A continuación, se explica cada parte del código:
El @Component define el componente Angular. Especifica el selector

(app-iotindicator), la plantilla HTML (./iotindicator.component.html) y un array vacío para los estilos CSS.

```
@Component({
  selector: 'app-iotindicator',
  templateUrl: './iotindicator.component.html',
  styles: []
})
```

El componente IotindicatorComponent tiene tres propiedades:
@Input() config: any. Esta propiedad se utiliza para recibir datos de un componente padre. config puede ser de cualquier tipo (any). topicini = . Inicializa la propiedad topicini como una cadena vacía. Esta propiedad almacenará el MQTT para suscribirse. value = false: Inicializa la propiedad value a false. Esta propiedad representa el estado actual del indicador.

```
export class IotindicatorComponent {
  @Input() config: any;
  topic_ini = "";
  value = false;
```

El constructor inyecta una dependencia del servicio EventBusService en el componente, que se almacena en la propiedad eventBusService.

```
constructor(private _eventBusService: EventBusService) {}
```

Los datos recibidos son procesados por el método processReceiveData. Intenta asignar el valor de data1.value al valor de la propiedad y corregir cualquier error que pueda ocurrir mientras realiza el proceso.

```
processReceiveData(data_1: any) {
  try {
    console.log('Data_1 recibidos', data_1);
    this.value = data_1.value;
  } catch (error) {
    console.error('Error: ', error);
  }
}
```

El método ngOnInit se ejecuta una vez que se inicializa el componente. Construye el MQTT (topicini) utilizando las propiedades de config y se suscribe a este tema usando el servicio eventBusService. Cuando se reciben datos en este tema, se llama al método processReceiveData.

```
ngOnInit(): void {
  this.topic_ini = this.config.userId
  + "/" + this.config.selectedDevice.dId + "/"
  + this.config.variable + "/sdata";
  this._eventBusService.on(this.topic_ini).subscribe(data => {
    this.processReceiveData(data);
  });
  console.log('NGN=>topic:', this.topic_ini);
}
```

El método ngOnChanges se ejecuta cuando cambian las propiedades enlazadas al componente. Si la propiedad config cambia, se espera 300 milisegundos, se resetea value a false, se desuscribe del tema antiguo, se construye un nuevo tema y se suscribe a él.

```
ngOnChanges(changes: SimpleChanges): void {
  if (changes["config"]) {
    setTimeout(() => {
      this.value = false;
      this._eventBusService.off(this.topic_ini);
      this.topic_ini = this.config.userId + "/"
      + this.config.selectedDevice.dId + "/"
      + this.config.variable + "/sdata";
      this._eventBusService.on(this.topic_ini).subscribe(data => {
        this.processReceiveData(data);
      });
    }, 300);
  }
}
```

El método ngOnDestroy se ejecuta cuando el componente se destruye. Se desuscribe del MQTT para limpiar los recursos.

```
ngOnDestroy(): void {
  this._eventBusService.off(this.topic_ini);
}
```

El siguiente código HTML renderiza una tarjeta con un encabezado y un cuerpo centrado que contiene un icono. Los valores del encabezado y las clases del icono se determinan dinámicamente a partir de la propiedad config del componente IotindicatorComponent. El código utilizado se muestra a continuación.

```
<div class="card" style="width: 100%;">
    <div slot="header" class="text-center">
        <h4 class="card-title">{{config.selectedDevice.name}}</h4>
    </div>
<div class="card-body" class="text-center">
<i class="fa {{config.icon}} {{getIconColor()}}"
style="font-size: 30px;"></i>
    </div>
</div>
```

El siguiente código es un componente de Angular llamado NavbarComponent. Este componente representa una barra de navegación que permite al usuario seleccionar dispositivos, ver títulos de páginas y controlar la visibilidad de una barra lateral. A continuación, se explica cada parte del código de manera concisa:
Este decorador define el selector del componente (app-navbar), su plantilla HTML (navbar.component.html) y sus estilos (navbar.component.css). La clase NavbarComponent implementa la interfaz OnInit.

```
@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.css']
})
```

El constructor inyecta los servicios necesarios (PeticionesHttpService, Location, ElementRef, y Router) y establece valores iniciales.

```
constructor(
  private _peticionesHttp: PeticionesHttpService,
  location: Location,
  private element: ElementRef,
  private router: Router
) {
  this.location = location;
  this.sidebarVisible = false;
}
```

NgOninit se encarga de suscribir a los dispositivos observables y al dispositivo seleccionado. Filtrar las rutas para obtener los títulos de la navegación. Configurar el botón de alternar y se suscribe a los eventos de la ruta para cerrar la barra lateral cuando cambia la ruta.

```
async ngOnInit() {
  await this._peticionesHttp.devicesAsObsevable().subscribe(disp => {
    this.devices = disp;
    this._peticionesHttp.getdeviceSelected().subscribe(dispSeleccionado => {
      this.deviceSelected = dispSeleccionado;
    });
  });}
```

La siguiente plantilla HTML es una barra de navegación (navbar) para un componente de Angular, el NavbarComponent. Debajo se explica cómo se estructura y cómo funciona:
El navbar principal utiliza clases de Bootstrap para estilizar y posicionar la barra de navegación.

```
<nav class="navbar navbar-expand-lg
navbar-transparent navbar-absolute
bg-primary fixed-top">
```

El navbar Wrapper contiene el botón para alternar la visibilidad de la barra lateral (sidebarToggle) y un enlace que muestra el título actual de la página (getTitle()).

```
<div class="navbar-wrapper">
    <div class="navbar-toggle">
    <button type="button"class=navbar-toggler (click)="sidebarToggle()">
            <span class="navbar-toggler-bar bar1"></span>
            <span class="navbar-toggler-bar bar2"></span>
            <span class="navbar-toggler-bar bar3"></span>
        </button>
    </div>
    <a class="navbar-brand" href="#pablo">{{getTitle()}}</a>
</div>
```

El NumberchartComponent es un componente de Angular que representa un gráfico de líneas. Debajo se muestra el código completo y una explicación detallada de cómo funciona.

A través de varios métodos del ciclo de vida de Angular (ngOnInit, ngAfterViewInit, ngOnChanges, y ngOnDestroy), el componente se inicializa, se actualiza y se destruye de manera eficiente, asegurando que los datos del gráfico se actualicen adecuadamente cuando cambian las configuraciones o se reciben nuevos datos. El método actualizarLabels es crucial para actualizar las etiquetas del gráfico. Recorre los datos del gráfico, formatea las fechas en un formato específico (día-mes-hh:mm:ss), y actualiza el array lineChartLabels. Esto asegura que las etiquetas del gráfico reflejen correctamente los datos temporales recibidos, proporcionando una representación visual clara y precisa.

```
actualizarLabels(){
  this.lineChartLabels=[];
  this.lineChartData[0].data.map(item=>{
    const date=new Date(item.x);
    const mes=date.toLocaleString('default',{month:'short'});
    const dia=date.getDate();
    const hh=date.getHours();
    const mm=date.getMinutes();
    const ss=date.getSeconds();
    this.lineChartLabels.push(`${dia}-${mes}-${hh}:${mm}:${ss}`);
  });
}
```

El método processReceiveData se encarga de procesar los datos entrantes. Al recibir nuevos datos, actualiza la variable this.value con el valor recibido y registra la hora actual en this.time. Este método es esencial para mantener el estado del componente sincronizado con los datos en tiempo real, permitiendo que el gráfico refleje los cambios a medida que ocurren.

```
processReceiveData(data:any){
  this.time=Date.now();
  this.value=data.value;
}
```

En ngAfterViewInit, después de que la vista del componente se ha inicializado completamente, se suscribe a un topic específico para recibir datos y actualiza el gráfico en consecuencia. También configura un timeout para actualizar el nombre del gráfico y el color después de un segundo. Esto permite que el gráfico se ajuste dinámicamente a los cambios en la configuración del componente.

```
async ngAfterViewInit(){
  await this.iniciarValores();
  this.topic_ini=this.config.userId+"/"+
  this.config.selectedDevice.dId+"/"+
  this.config.variable+"/sdata";
  this._eventBus.on(this.topic_ini).subscribe(data=>{
    this.processReceiveData(data);
  });
  setTimeout(()=>{
    this.lineChartData[0].name=
    `${this.config.variableFullName} ${this.config.unit}`;
    this.updateColorClass();
    window.dispatchEvent(new Event('resize'));
  },1000);
}
```

El método ngOnChanges se llama cuando se detectan cambios en las propiedades vinculadas de entrada del componente. Si hay cambios en config, el componente se desconecta del topic anterior, se suscribe al nuevo, y actualiza el gráfico después de un segundo. Esto asegura que el gráfico siempre esté conectado al topic correcto y refleje la configuración actual del componente.

```
ngOnChanges(changes: SimpleChanges): void {
  if (changes['config']) {
    this._eventBus.off(this.topic_ini);
    this.topic_ini = this.config.userId +
    "/" + this.config.selectedDevice.dId + "/"
    this.config.variable + "/sdata";
    this._eventBus.on(this.topic_ini).subscribe(data => {
      this.processReceiveData(data);
    });
    setTimeout(() => {
      this.lineChartData[0].name =
      `${this.config.variableFullName} ${this.config.unit}`;
      this.updateColorClass();
      this.getNow();
      window.dispatchEvent(new Event('resize'));
    }, 1000);
  }
}
```

Por último, el método getChartData obtiene los datos del gráfico, ya sea insertando datos de prueba si config.demo es verdadero, o haciendo una solicitud HTTP para obtener los datos reales. Los datos se procesan y se actualizan las etiquetas del gráfico, asegurando que el gráfico muestre información precisa y actualizada. Este método es fundamental para la funcionalidad del gráfico, permitiendo que se actualice con datos reales o de prueba según sea necesario.

```
getChartData(){
  this.isMounted=false;
  this.config.demo=true
    console.log('data:',this.lineChartData[0].data);
    this.actualizarLabels();
    console.log('labels:',this.lineChartLabels);
    this.isMounted=true;
    this.config.demo=false;
    this.value=25.7;
    return;

  }
  const http_headers={
    headers:{token:'token de acceso'},
    params:{
      dId:this.config.selectedDevice.dId,
      variable:this.config.variable,
      chartTimeAgo:this.config.chartTimeAgo,
    },
  }
```

Este código define un servicio en Angular, MngWidgetsService, está diseñada para gestionar widgets en la aplicación. El decorador @Injectable en la parte superior de la clase permite que Angular proporcione una instancia única de este servicio a lo largo de toda la aplicación, garantizando que todos los componentes que lo inyecten compartan la misma instancia.

La clase contiene varias propiedades clave: templates, widgets, y widgetType. templates es un array vacío que se utilizará para almacenar plantillas de widgets, mientras que widgets es un array que contiene los widgets configurados en forma de objetos IConfigW. widgetType es un array de cadenas que especifica los diferentes tipos de widgets disponibles, como 'numberChart', 'indicator', 'map', 'switch', y 'button'.

```
templates = [];
widgets: IConfigW[] = [];
widgetType = ['numberChart', 'indicator', 'map', 'switch', 'button'];
```

El método readTypeWidgets permite obtener el tipo de widget correspondiente a un índice dado. Si el índice es válido (es decir, menor que la longitud del array widgetType), el método devuelve el tipo de widget correspondiente; de lo contrario, devuelve una cadena vacía. Este método es útil para traducir índices numéricos a tipos de widgets legibles.

```
readTypeWidgets(type: number) {
  if (type < this.widgetType.length) {
    return this.widgetType[type];
  }
  return '';
}
```

El método readWidgets devuelve la lista de widgets almacenada en la propiedad widgets. Esto permite a otros componentes o servicios obtener la lista completa de widgets para su visualización o manipulación. El método addNewWidget permite añadir un nuevo widget a la lista. Para evitar mutaciones no deseadas del objeto original, el método añade una copia profunda (deep copy) del nuevo widget al array widgets.

```
readWidgets(): IConfigW[] {
  return this.widgets;
}
```

El método delWidget permite eliminar un widget de la lista basado en su posición. Si la posición es válida, el método elimina el widget en la posición especificada del array widgets. Esto es útil para permitir la eliminación dinámica de widgets en la interfaz de usuario. El método makeId genera un identificador único de la longitud especificada. Utiliza caracteres alfanuméricos y el punto (.) para crear una

cadena aleatoria, lo cual es útil para asignar identificadores únicos a nuevos widgets.

```
delWidget(pos: number) {
  if (pos < this.widgets.length) {
    this.widgets.splice(pos, 1);
  }
}
```

El siguiente código define una clase de servicio Angular, PeticionesHttpService, que maneja varias operaciones relacionadas con la autenticación de usuarios, la gestión de dispositivos, plantillas, y reglas de alarmas. Este servicio se inyecta a nivel de raíz, lo que significa que es una instancia única en toda la aplicación Angular. El @Injectable indica que este servicio será proporcionado en la raíz de la aplicación, asegurando una única instancia. La clase tiene varias propiedades, incluyendo token, expiresIn, devices, y templates. También tiene dos BehaviorSubject para manejar el estado observable de los dispositivos seleccionados y públicos.

```
@Injectable({
  providedIn: 'root'
})
export class PeticionesHttpService {
  token: string = '';
  expiresIn: number = 0;
  devices: IDevice[] = [];
  templates: ITemplates[] = [];
  private deviceSelected = new BehaviorSubject<number>(-1);
  private devicePublic = new BehaviorSubject<IDevice[]>([]);

  constructor(private http: HttpClient,
  private _uiService: UiServiceService) { }
```

Los siguientes métodos permiten obtener y establecer el dispositivo seleccionado y obtener los dispositivos públicos como observables. Esto es útil

para que otros componentes puedan suscribirse a estos cambios y reaccionar en consecuencia.

```
getdeviceSelected() {
  return this.deviceSelected.asObservable();
}

setDeviceSelected(newDato: number) {
  this.deviceSelected.next(newDato);
}

devicesAsObsevable() {
  return this.devicePublic.asObservable();
}
```

A continuación, los siguientes métodos manejan el almacenamiento y la validación del token de autenticación. guardarToken almacena el token y su expiración en localStorage. cargarToken carga estos valores desde localStorage si no están ya en memoria. isAuthenticated verifica si el token es válido y no ha expirado.

```
guardarToken(token: string, expiresIn: number) {
  this.token = token;
  this.expiresIn = expiresIn;
  localStorage.setItem('token', token);
}
cargarToken() {
  if (!this.token) {
    this.token = localStorage.getItem('token') || '';
  }
}
isAuthenticated(): boolean {
  this.cargarToken();
  if (!this.token || this.token === undefined
  || this.token === 'undefined') return false;
  if (this.token.length < 10) return false;
  const timeOk = this.expiresIn >= (Date.now() / 1000);
  if (timeOk) {
    return true;
  } else {
    this.token = '';
    localStorage.clear();
    this._uiService.alertaError('Token invalido');
    return false;
  }
}
```

Los siguientes métodos manejan las operaciones de login y registro de usuarios. login envía una solicitud POST a la API con las credenciales del usuario y, si es exitosa, guarda el token. register envía una solicitud POST para crear un nuevo usuario y, si es exitosa, también guarda el token.

```
  login(email: string, password: string) {
    const data = { email, password };
    return new Promise(resolve => {
      this.http.post(`${URL}/user/login`, data).subscribe(resp => {
        if (resp['ok']) {
          this.guardarToken(resp['token'], resp['expiresIn']);
          resolve(true);
        }
      });
    });
  }
  register(usuario: User) {
    return new Promise(resolve => {
      try {
        this.http.post(`${URL}/user/create`, usuario).subscribe(resp => {
          if (resp['ok']) {
            this.guardarToken(resp['token'], resp['expiresIn']);
            resolve(true);
          }
    });
  }
```

El sigiente código proporciona un servicio Angular llamado ConexionMqttService que maneja la conexión a un broker MQTT. Este servicio está decorado con @Injectable con la opción providedIn: 'root', lo que lo convierte en un singleton disponible en toda la aplicación. La clase contiene varias propiedades y métodos para configurar y manejar la comunicación MQTT.

```
@Injectable({
  providedIn: 'root'
})
```

Las propiedades principales incluyen nameUser, que obtiene el nombre del

usuario a través de un servicio de peticiones HTTP (PeticionesHttpService); client, que es el cliente MQTT; y varias configuraciones de conexión almacenadas en el objeto connection, como el host, puerto, protocolo, y credenciales de usuario.

```
nameUser: string = this._peticionesHttp.user.name;
client: MqttService | undefined;
isConnection = false;
endpoint = '/mqtt';
connectUrl = '';
connection: IMqttServiceOptions = {
  host: MQTT_SERVICE_OPTIONS.hostname,
  port: MQTT_SERVICE_OPTIONS.port,
  path: MQTT_SERVICE_OPTIONS.path,
  clean: true,
  connectTimeout: 5000,
  reconnectPeriod: 5000,
  clientId: "web_" + this.nameUser,
  username: "userp64",
  password: "12345678",
  protocol: MQTT_SERVICE_OPTIONS.protocol
}
```

El método actualizaConnection actualiza la configuración de conexión del cliente MQTT, incluyendo la generación de un nuevo clientId y la construcción de la URL de conexión. Este método es llamado dentro de startMqttClient, que configura y establece la conexión MQTT. Este último método también se asegura de que el usuario esté autenticado antes de proceder con la conexión y configura los tópicos a los que se suscribirá el cliente.

```
actualizaConnection() {
  this.connection.host = MQTT_SERVICE_OPTIONS.hostname;
  this.connectUrl = MQTT_SERVICE_OPTIONS.protocol;
}
```

Dentro de startMqttClient, se manejan varios eventos importantes. El evento onConnect se dispara cuando la conexión es exitosa, estableciendo isConnection a true. El evento onError se maneja para capturar y registrar errores de conexión. Además, el método suscribe al cliente a dos tópicos (deviceSubscribeTopic y notifSubscribeTopic) y define la lógica para manejar los mensajes recibidos en estos tópicos. También maneja la lógica para reconexiones y publica mensajes a través del bus de eventos.

```
startMqttClient() {
 console.warn('this.connection:', this.connection);
 if (!this._peticionesHttp.isAuthenticated()) {
   this.client?.disconnect();
 }
 this.nameUser = this._peticionesHttp.user.name;
 this.actualizaConnection();
 const _id = this._peticionesHttp.user._id;
 const deviceSubscribeTopic = _id + "/+/+/sdata";
 const notifSubscribeTopic = _id + "/+/+/notif";
   this.client = this._mqttService;
   console.warn('this.connection:', this.connection);
   this.client.connect(this.connection);
 this.client?.onConnect.subscribe((xx1) => {
   this.isConnection = true;
   console.log('Conexion exitosa, clientId:', this.client?.clientId);
 });
 this._eventBus.on('mqtt-sender').subscribe(toSend=>{
   console.log('************_eventBus.on***************');
   console.log('toSend');
   this.client?.unsafePublish(toSend.topic, JSON.stringify(toSend.msg));
 });}}
```

2.4. Diseño de la base de datos

Para el diseño de la base de datos se usó MongoDB una base de datos no relacional de libre uso. El primer paso fue configurar el servidor en el index.ts que se puede observar en el siguiente código. En este código se puede observar que s esta utilizando el localhost en el puerto predeterminado. Además de que en este código se establece los errores en el caso de que la base de datos no se conecte debidamente. Por último, se establece las rutas para el uso de los usuarios, dispositivos y templates.

```
const server=new Server();
//conectar db
const options:ConnectOptions={
    autoIndex:true,
}
mongoose.connect('mongodb://127.0.0.1:27017/IoT_p64', options)
    .then((msnOK)=>{
        console.log(colors.bgYellow('BASE DE DATOS ONLINE'));
    }
//Levantar el servidor express
server.start(()=>{
    console.log(`SERVIDOR-P64 corriendo en puerto ${server.port}`)
})
//body parser
server.app.use(bodyParser.urlencoded({extended:true}));
server.app.use(bodyParser.json());
// ruta de usuario
server.app.use('/user',userRoutes);
//ruta de dispositivos
server.app.use('/api',devicesRoutes);
// ruta de templates
server.app.use('/widgets',templatesRoutes);
```

A continuación, se presenta la configuración de la base de datos para los usuarios. En primer lugar, para garantizar una mayor seguridad, se utiliza un token

que se genera automáticamente mediante el siguiente código, en el cual planteamos una semilla y una caducidad de 30 dias para cada token:

```
export default class Token{
    private static seed:string='es-la-semilla-del-App-p64';
    private static caducidad='30d';
    constructor(){}
    static getJwtToken(payload:any):string{
        return jwt.sign({
            usuario:payload
        },this.seed,{expiresIn:this.caducidad});
    static comprobarToken(userToken:string){
       return new Promise((resolve,reject)=>{
        jwt.verify(userToken,this.seed,(err,decoded)=>{
            if(err){
                reject('ERROR EN TOKEN');
            }else{
                resolve(decoded);
            }})})}}
```

Después de generar el token, configuramos la creación de usuarios, un sistema de inicio de sesión, y las diversas consultas que se realiza en la base de datos. La primera de ellas va a ser crear un usuario para el cual se extraen los datos del body de la solicitud (req.body), se crea un nuevo objeto user con los datos proporcionados y la contraseña encriptada. Luego, se guarda en la base de datos. Si la operación es exitosa, se genera un token JWT para el usuario creado y se envía una respuesta JSON con los detalles del usuario y el token. Si hay un error, se captura y se envía un mensaje de error.

```
userRoutes.post('/create', (req: Request, resp: Response) => {
    const user = {
        name: req.body.name,
        email: req.body.email,
        password: bcrypt.hashSync(req.body.password, 10),
        avatar: req.body.avatar
    };
    Usuario.create(user).then(userDB => {
        const tokenUser = Token.getJwtToken({
            _id: userDB._id,
            name: userDB.name,
            email: userDB.email,
            avatar: userDB.avatar
        });
        resp.json({
            ok: true,
            mensaje: 'TODO OK EN CREATE',
            token: tokenUser,
            expiresIn: Date.now() + 30 * 24 * 3600 * 1000,
            user: userDB
        });
    }).catch(err => {
        console.log('ERROR en catch:', err);
    });
```

Como siguiente paso realizamos el código para el inicio de sesión en el cual se busca un usuario en la base de datos con el correo electrónico proporcionado. Si no se encuentra, se devuelve un mensaje de error. Si se encuentra, se compara la contraseña proporcionada con la almacenada. Si coinciden, se genera un token JWT para el usuario y se envía una respuesta JSON con los detalles del usuario y el token. Si las contraseñas no coinciden, se devuelve un mensaje de error. Si hay un error, se captura y se envía un mensaje de error.

```
userRoutes.post('/login', (req: Request, resp: Response) => {
    const body = req.body;
    Usuario.findOne({ email: body.email }).then(userDB => {
        console.log(colors.bgCyan('En login'), body);
        if (!userDB) {
            return resp.json({
                ok: false,
                mensaje: 'Usuario/Contraseña no validos'
            });
        }
        if (userDB.compararPassword(body.password)) {
            const tokenUser = Token.getJwtToken({
                _id: userDB._id,
                name: userDB.name,
                email: userDB.email,
                avatar: userDB.avatar
            });
        }
    }).catch(err => {
        console.log('ERROR EN LOGIN', err);
        if (err) throw err;
    });
});
```

Para actualizar los datos de los usuarios, el código a continuación se extrae del cuerpo de la solicitud (req.body). Se encripta si se le da una nueva contraseña. Luego, el usuario se actualiza en la base de datos utilizando su ID. Se crea un nuevo token JWT para el usuario actualizado y se envía una respuesta JSON con el nuevo token si la operación es exitosa. Un mensaje de error se devuelve si el usuario no se encuentra. Se captura y envía un mensaje de error si hay un error.

```
userRoutes.get('/', [verificarToken], (req: Request, resp: Response) => {
    const usuario = req.body.usuario;
    resp.json({
        pk: true,
        usuario,
        expiresIn: req.body.expiresIn
    });
});
```

Luego de configurar los usuarios configuraremos de igual manera los dispositivos a conectarse en la base de datos. Este código define varias rutas para manejar dispositivos en una aplicación Node.js utilizando Express y TypeScript. Aquí se explica cada parte del código:

```
const devicesRoutes = Router();
```

Luego se define una ruta para crear un dispositivo, en la cual se definen los parámetros que van a tener los dispositivos en donde se extraen los datos del body (req.body), se crea un nuevo objeto IDevice y se guarda en la base de datos. Si la operación es exitosa, se genera un token JWT para el dispositivo creado y se envía una respuesta JSON con los detalles del dispositivo y el token. Si hay un error, se captura y se envía un mensaje de error:

```
devicesRoutes.post('/device', verificarToken => {
    const newDevice: IDevice = {
        userId: user._id,
        dId: body.dId,
        name: body.name,
        password: makeId(10),
        selected: body.selected,
        templateId: body.templateId,
        templateName: body.templateName,
        createdTime: Date.now()
    };
});
```

A continuación se observa el código para leer los dispositivos en donde primero, deselecciona todos los dispositivos del usuario (selected: false). Luego, selecciona el dispositivo especificado (selected: true). Si la operación es exitosa, se envía una respuesta JSON indicando que el dispositivo fue seleccionado correctamente. Si hay un error, se captura y se envía un mensaje de error.

```
devicesRoutes.get('/device', (req: Request, resp: Response) => {
    Device.find({ userId: req.body.usuario._id }).then(deviceDB => {
        resp.status(200).json({
            ok: true,
            deviceDB
        });
    }).catch(err => {
        console.log(colors.bgRed('ERROR EN GET DEVICE'), err);
});
```

Luego el siguiente código actualiza el estado de selección de los dispositivos de un usuario. Primero, deselecciona todos los dispositivos del usuario (selected: false). Luego, selecciona el dispositivo especificado (selected: true). Si la operación es exitosa, se envía una respuesta JSON indicando que el dispositivo fue seleccionado correctamente. Si hay un error, se captura y se envía un mensaje de error.

```
devicesRoutes.put('/device', async (req: Request, resp: Response) => {
    try {
        await Device.updateMany({ userId: req.body.usuario._id });
        await Device.updateOne({ userId: req.body.usuario._id});
        resp.json({
            ok: true,
            mensaje: 'Dispositivo seleccionado correctamente',
        });
    } catch (err) {
        resp.status(500).json({
            ok: false,
            error: err
        });
    }
});
```

El siguiente código es utilizado para borrar un dispositivo, en el cual se busca y elimina el dispositivo identificado por userId y dId del usuario. Si la operación es exitosa, se envía una respuesta JSON con el resultado. Si hay un error, se captura y se envía un mensaje de error.

```
devicesRoutes.delete('/device', (req: Request, resp: Response) => {
    Device.deleteOne({ userId: req.body.usuario._id, }).then(result => {
        resp.status(200).json({
            ok: true,
            result: result
        });
    }).catch(err => {
        console.log(colors.bgRed('ERROR EN DELETE DEVICE'), err);
        resp.json({
            ok: false,
            mensaje: 'Error en DELETE de device',
            error: err.errors
        });
    });
});
```

Por último se integró el código para ingresar los templates para los datos, el código explicado se encuentra continuación. En la primera parte el código se usa para listar todas los templates de un usuario en especifico, para ello se buscan las plantillas en la base de datos que pertenezcan al usuario identificado por req.body.usuario.id. Si se encuentran plantillas, se devuelven en una respuesta JSON. Si no se encuentran plantillas o hay un error, se envía un mensaje de error.

```
templatesRoutes.get('/template', (req: Request, resp: Response) => {
    Template.find({ userId: req.body.usuario._id }).then(templateDB => {
        if (templateDB) {
            resp.status(200).json({
                templateDB
            });
    }).catch(err => {
            mensaje: 'Error en get template',
});
```

Luego, el código para crear templates en el cual primero se extraen los datos del cuerpo de la solicitud (req.body.template), se crea un nuevo objeto newTemplate con los datos proporcionados, incluyendo userId, name, descripcion, createdTime y widgets. Luego, se guarda en la base de datos. Si la operación es exitosa, se devuelve una respuesta JSON con los detalles de la plantilla creada. Si hay un error, se captura y se envía un mensaje de error.

```
templatesRoutes.post('/template', async (req: Request, resp: Response) => {
    const template1 = req.body.template;
    const newTemplate = {
        userId: req.body.usuario._id,
        name: template1.name,
        descripcion: template1.descripcion,
        createdTime: Date.now(),
        widgets: template1.widgets || []
    }).catch(err => {
        console.log(colors.bgRed('ERROR EN CREAR TEMPLATE'));
        console.log(colors.red(err));
        resp.json({
            ok: false,
            mensaje: 'Error en create template',
            error: err
        });
    });
});
```

En el siguiente código se usa para actualizar un template para lo cual se verifica el token del usuario y se envía una respuesta JSON indicando que la operación fue exitosa.

```
templatesRoutes.put('/template', (req: Request, resp: Response) => {
    resp.json({
        ok: true,
        mensaje: 'PUT FUNCIONA TODO OK in templates'
    });
});
```

Por último se realizó el código para eliminar un template par lo cual se extrae idTemplate del cuerpo de la solicitud (req.body.idTemplate). Si idTemplate es inválido (menos de 20 caracteres), se devuelve un mensaje de error. Luego, se intenta eliminar la plantilla de la base de datos usando el id y userId. Si la operación es exitosa y se elimina alguna plantilla, se devuelve una respuesta JSON con el resultado. Si no se encuentra la plantilla o hay un error, se envía un mensaje de error.

```
templatesRoutes.delete('/template', (req: Request, resp: Response) => {
    const temp1 = req.body.idTemplate || '';
    if (temp1.length < 20) {
        return resp.status(200).json({
            ok: false,
            idTemplate: req.body.idTemplate || '',
            mensaje: 'idTemplate no valido'
        });
    }
    Template.deleteOne({ _id: temp1, userId:
    }).catch(err => {
        resp.json({
            ok: false,
            mensaje: 'Error en DELETE template',
            error: err
        });
    });
});
```

2.5. Diseño de Comunicación LoRaWAN y Página Web

En este apartado se utilizaron varias herramientas, entre las cuales se encuentra ChirpStack como servidor del Gateway de LoRaWAN. Este cuenta con su propio broker MQTT a través de Mosquitto y usa el puerto 1884. Los datos son transmitidos a través de este broker e inyectados en Node-RED directamente mediante un nodo MQTT-IN, que únicamente solicita el tópico de referencia de los eventos generados por la aplicación de ChirpStack y su dirección, que corresponde al puerto IPv4 de la computadora utilizada como central en el puerto antes mencionado.
En Node-RED, los datos se procesan de tal manera que puedan ser leídos por la página web diseñada. En este caso, los datos tienen el formato {"save": 1, "value": any}, donde *any* es el valor correspondiente a subir al MQTT de la página web. Estos datos se referencian mediante un nodo MQTT-OUT hacia el puerto del MQTT de la página, que está dirigido a la IPv4 de la computadora y usa el puerto 1883. Estos datos son enviados a EMQX, que es el broker MQTT utilizado para la página. El tópico de referencia se asignará de acuerdo al dispositivo referenciado en la página.
La figura 2.17 muestra el diagrama de flujo del funcionamiento del enlace de comunicación entre el Gateway y la página web.

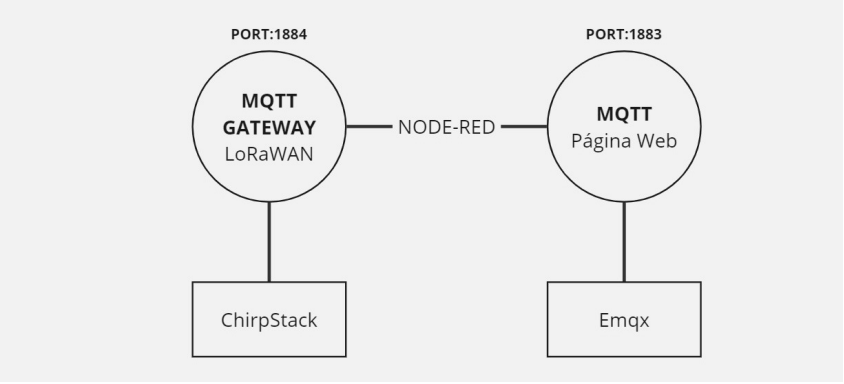

Figura 2.17: Diagrama de funcionamiento de la comunicación Gateway - Página web

En la figura 2.18, se puede observar el diseño dado para Node-RED. Los eventos captados por ChirpStack son transmitidos a los diferentes tópicos correspondientes en la página web, relacionados con las variables analizadas.

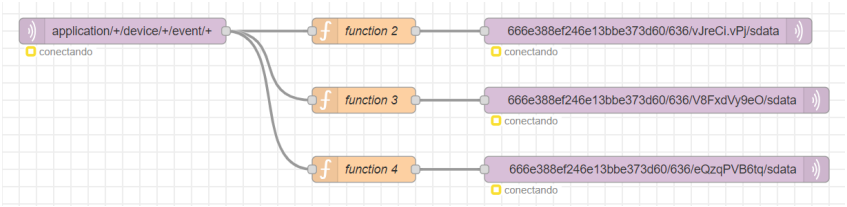

Figura 2.18: Diseño del flujo en Node-Red

2.6. Análisis de resultados

Los resultados realizados a las diferentes partes del prototipo se muestran en las siguientes figuras. En la figura 2.19 se muestra el ingreso del usuario en la página web, el cual debe hacerse con el usuario y contraseña proporcionado.

Figura 2.19: Ingreso a la página web

En la figura 2.20 se muestra el dashboard en el cual se mostrarán los datos y figuras que el usuario escoja.

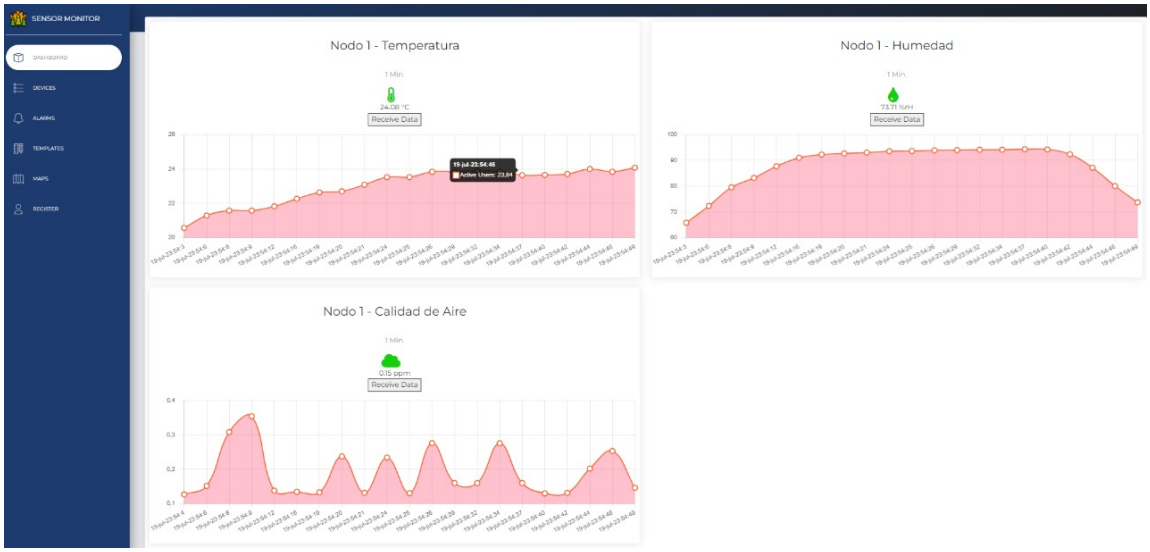

Figura 2.20: Dashboard

En la siguiente figura 2.21 se observa la página para agregar y quitar los dispositivos de la página web.

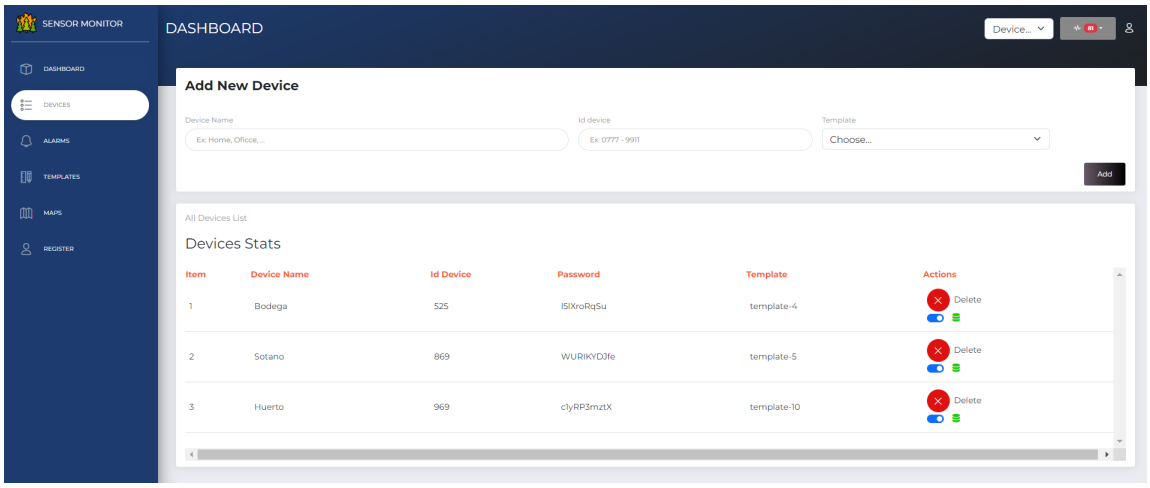

Figura 2.21: Devices

En la figura 2.22, se muestra la creación de alarmas para cada uno de los dispositivos añadidos. Es decir, en la página web es donde se configuran las especificaciones de cada sensor a utilizar.

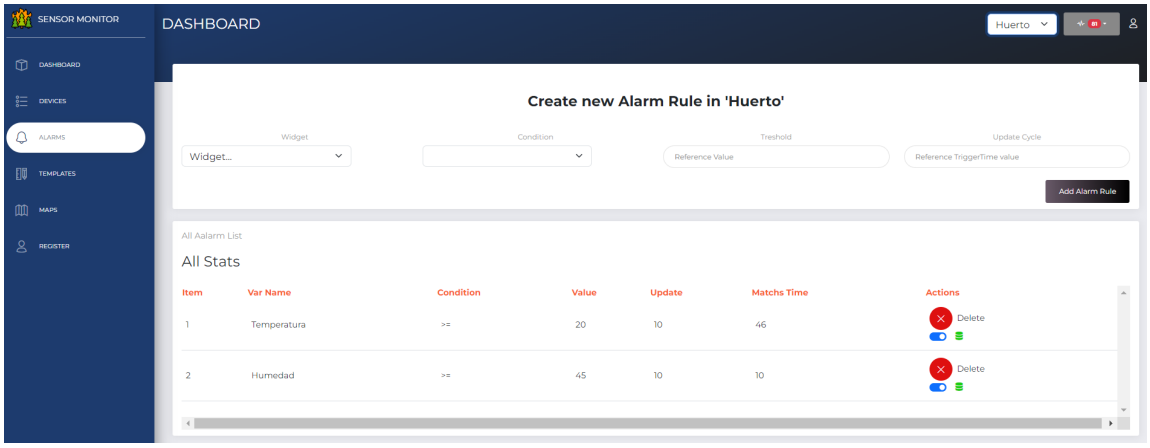

Figura 2.22: Alarms

En la figura 2.23, se muestra la creación de los templates. Es decir, se especifican los elementos que se desean incluir en el dashboard, como indicadores, gráficos, botones, seleccionadores, etc. Estos elementos se guardan como un "Templatez se muestra la cantidad de widgets que contiene.

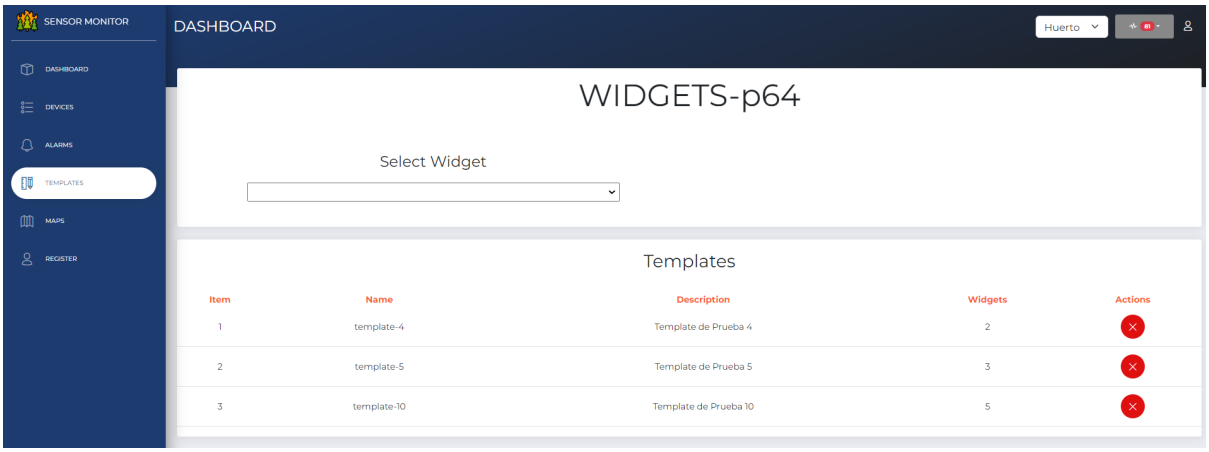

Figura 2.23: Templates

Una siguiente figura 2.24, muestra el mapa de la aplicación que fue realizado con la api de google maps.

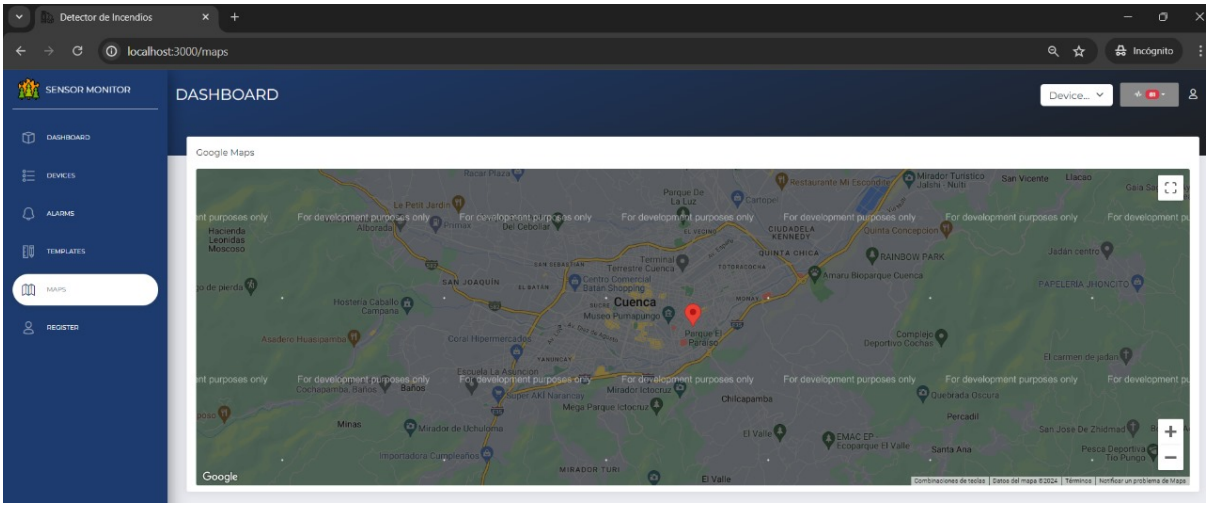

Figura 2.24: Mapa

En la siguiente figura 2.25 se muestran las notificaciones en caso de que una alarma sea activada.

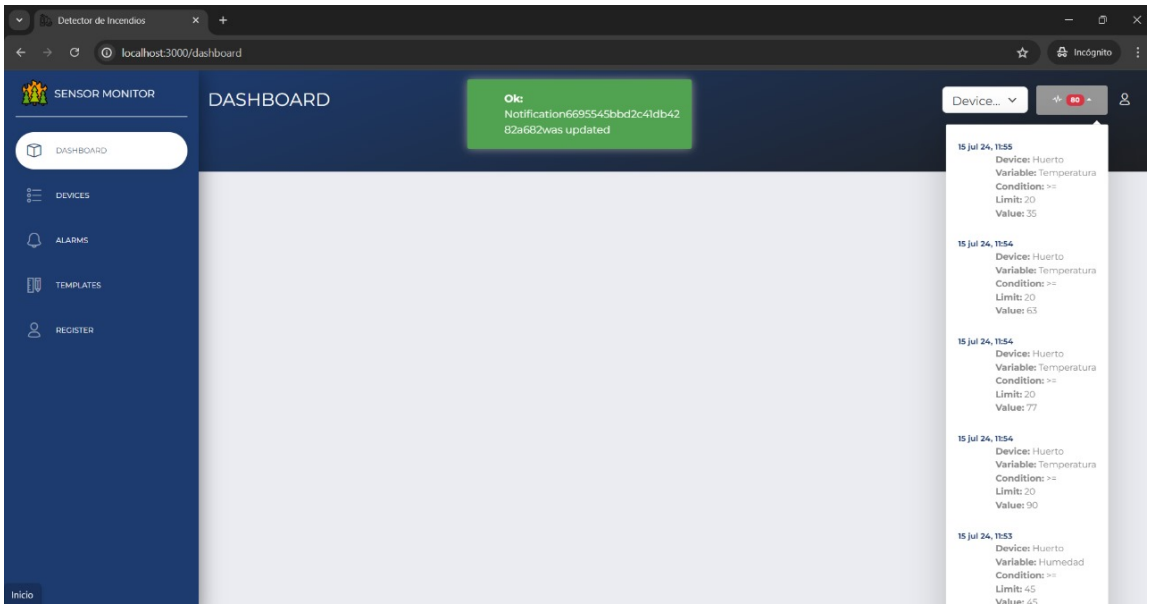

Figura 2.25: Notificaciones

Y para finalizar, se generó un apartado visual para el registro de nuevos usuarios que deseen agregar nuevos dispositivos. Cabe recalcar que cada usuario es independiente de los demás, es decir, sus identificadores están guardados de manera separada, por lo que nunca habrá algún cruce de información entre ellos. Esto se puede observar en la figura 2.26.

Figura 2.26: Registro de nuevos usuarios

Para comprobar el backend se muestra en la siguiente figura todos los registros generados en la base de datos de MongoDB.

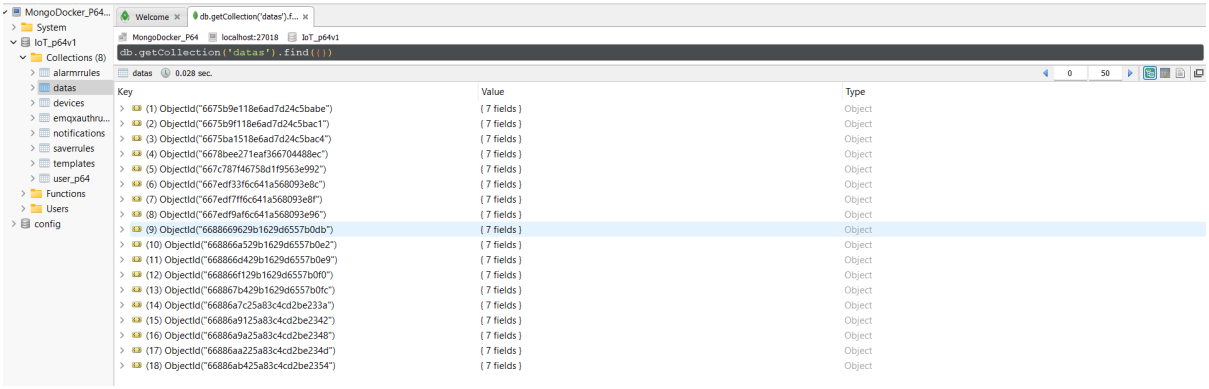

Figura 2.27: Base de datos

Por último para comprobar la parte electrónica se comprueba mediante el uso del gateway LoRa verificando su correcto funcionamiento. En las siguientes figuras 2.28 y 2.29 se muestra la correcta conexión del gateway y el envío de datos.

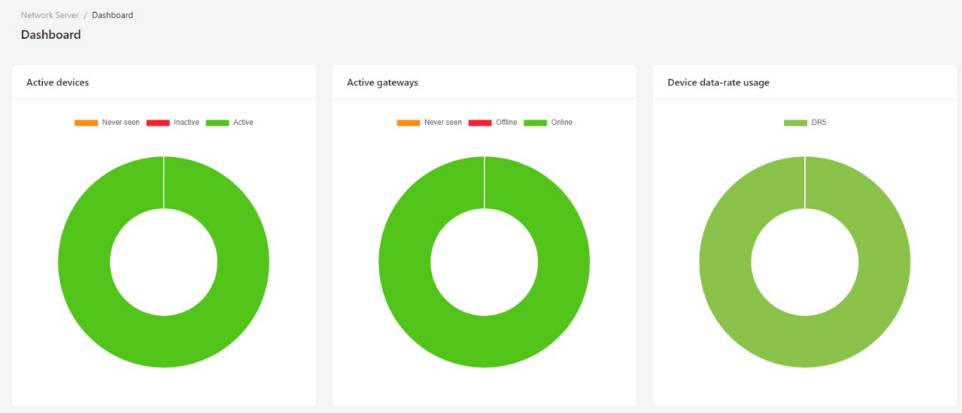

Figura 2.28: Uso gateway

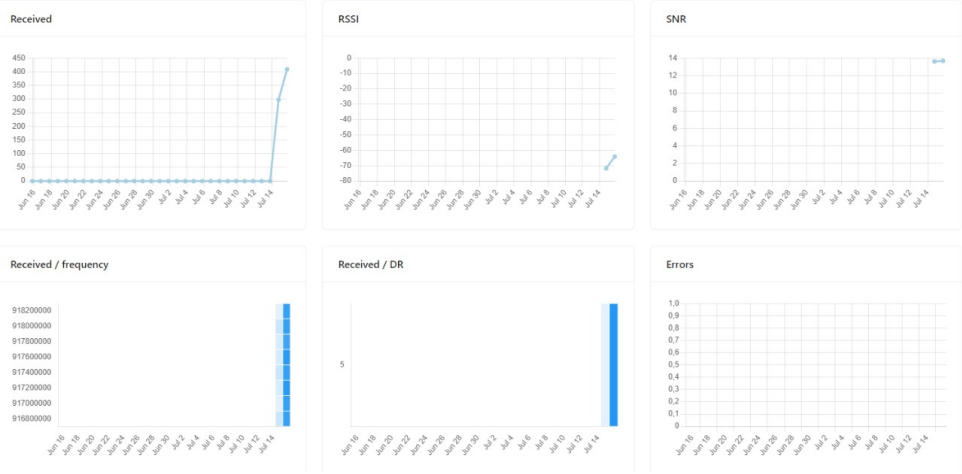

Figura 2.29: Recepción de datos

Los datos en el servidor LoRa se muestran a continuación en las siguientes figuras 2.30 y 2.31, cabe destacar que los datos deben ser desencriptados para su uso.

Figura 2.30: Datos recibidos

Figura 2.31: Datos recibidos ampliados

De la misma forma, en la figura 2.32, se puede observar el funcionamiento de la comunicación entre los brokers MQTT del gateway y la página web, con un envío constante de datos.

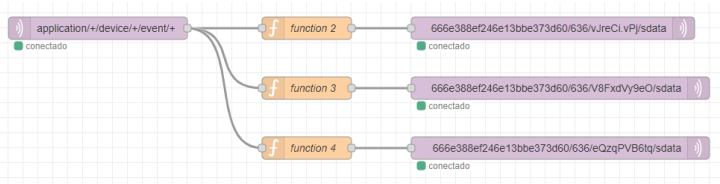

Figura 2.32: Comunicación activa entre Gateway y Página web

Capítulo 3

Capítulo III: Validación Experimental

La validación experimental es una etapa crucial en el desarrollo de cualquier sistema, y el prototipo IoT para la detección temprana de incendios forestales no es una excepción. En este capítulo, se detallará el proceso de validación experimental diseñado para evaluar el rendimiento, la precisión y la confiabilidad del sistema en condiciones reales de operación.

3.1. Construcción del prototipo

La construcción del prototipo se realizó mediante impresión 3D con PVC, un material retardante que previene la incineración rápida del interior. Asimismo, el prototipo se fabricó en varias partes ensamblables para facilitar su manipulación y montaje. A continuación, se muestra el prototipo armado en la figura 3.1. Cabe resaltar que, internamente, en la parte superior se encuentra toda la parte electrónica aislada, y en la parte inferior, a través de las ventilas, se ubican los sensores utilizados. Además, se han instalado todos los componentes adicionales, como el panel solar, la batería y las antenas de comunicación.

Figura 3.1: Prototipo para la detección de incendios

3.2. Experimentación

Para la experimentación se plantearon diferentes pruebas en la zona rural de Racar del cantón Cuenca, la primera de ellas fue el gasto de energía en un día. Para ello se realizaron los siguientes cálculos de gasto de energía.En la ecuación 4.1 sumamos todos los gastos de energía por parte de los sensores y de la misma tarjeta electrónica.

$$\textbf{Total} = I_{ESP32} + I_{MQ135} + I_{SHTC3} + I_{NEO6M} \tag{3.1}$$

En la siguiente ecuación 3.2 se obtuvo el consumo total por hora de nuestro prototipo basados en los datasheets de los sensores utilizados todos ellos en miliAmpereos/hora.

$$\textbf{Total} = 80 + 60 + 0{,}002 + 45 = 185mA \tag{3.2}$$

Por último calculamos en la siguiente ecuación 3.3 el total de energía usado en 24 horas

$$\textbf{Total} = 185x24 = 4440mA \tag{3.3}$$

Basados en el anterior cálculo realizamos mediciones cada hora para verificar

como la batería se desgasta a lo largo del día, en la siguiente figura 3.2 se muestra el contraste de lo calculado de lo medido.

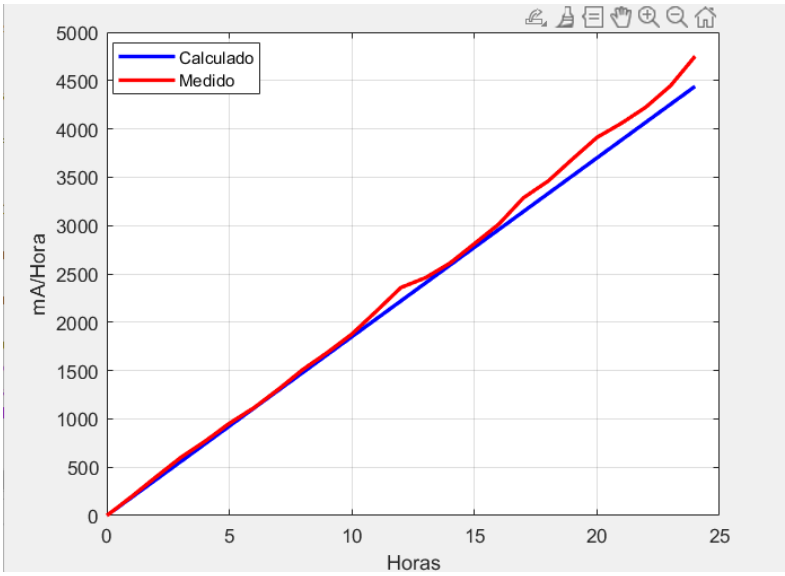

Figura 3.2: Uso de batería

En la siguiente prueba se analizará la latencia según la distancia, para ello se medirá la latencia del mensaje y se realizará un contraste versus la distancia.

En la figura 3.3 se observa el RSSI que se refiere a la potencia de la señal la cual disminuye a medida que se aleja el nodo del gateway, el RSSI disminuye hasta -100 a 1000 metros cuando se desconecta por completo el nodo del gateway.

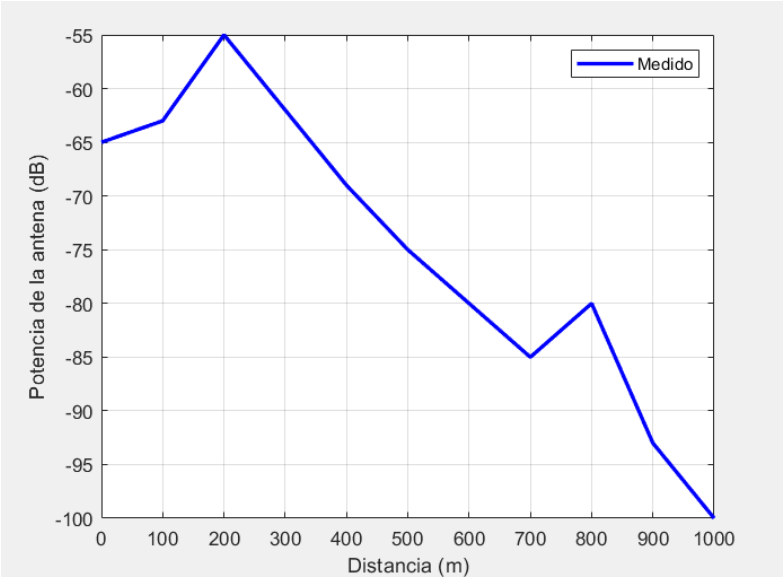

Figura 3.3: RSSI

En la siguiente figura 3.4 se muestra el SNR a lo largo de la distancia que en principio es 14dB y va disminuyendo hasta 7dB a medida que nos alejamos del gateway.

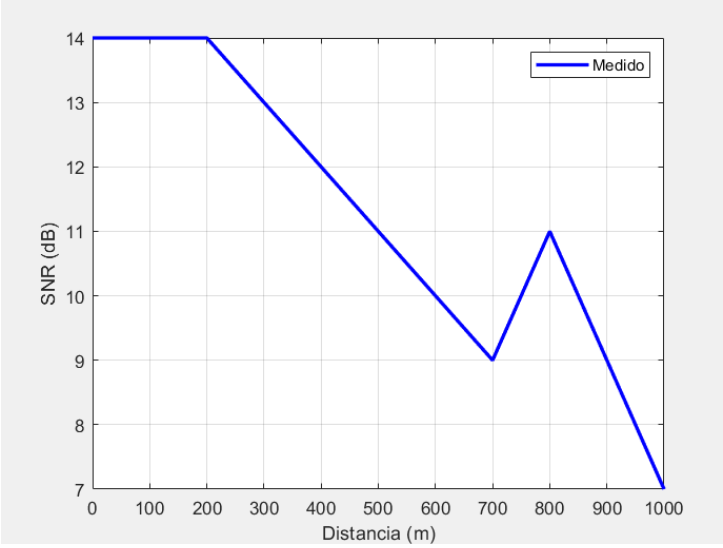

Figura 3.4: SNR

Por último, realizaremos el análisis de la latencia según el spread factor de LoRa, con un ancho de banda del canal de 125KHz, para ello se realizará las diferentes configuraciones diferentes, se escogió un spread factor de 5, 9 y 12 para las siguientes pruebas. El primer resultado con un spread factor de 12 con un $R_s = \frac{BW}{2^{SF}} = 3{,}9kbps$ se puede observar en la siguiente figura 3.5.

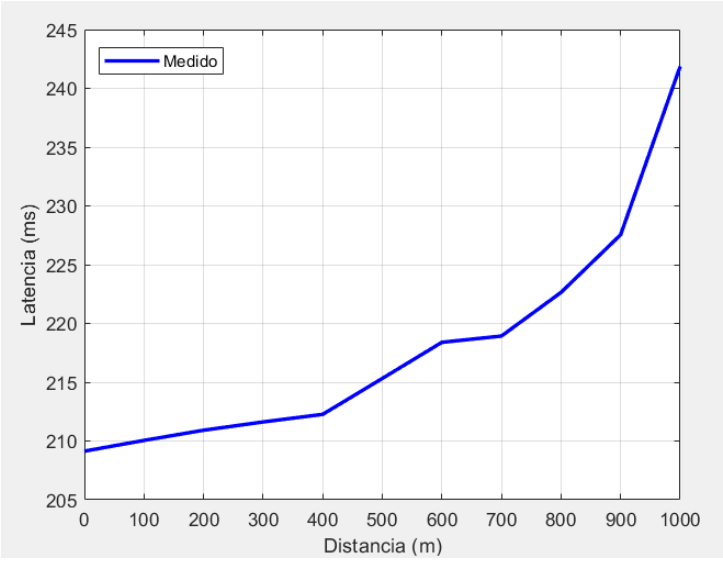

Figura 3.5: Distancia vs Latencia con SF=12

El segundo resultado con un spread factor de 9 con un $R_s = \frac{BW}{2^{SF}} = 0{,}244Kbps$ se puede observar en la siguiente figura 3.6 .

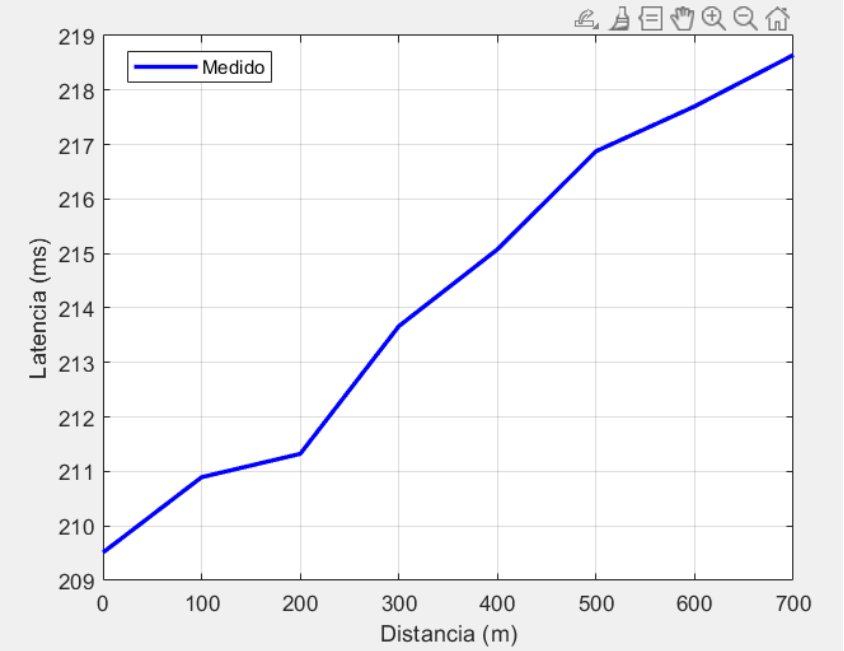

Figura 3.6: Distancia vs Latencia con SF=9

El último resultado con un spread factor de 5 con un $R_s = \frac{BW}{2^{SF}} = 0{,}03Kbps$ se puede observar en la siguiente figura 3.7.

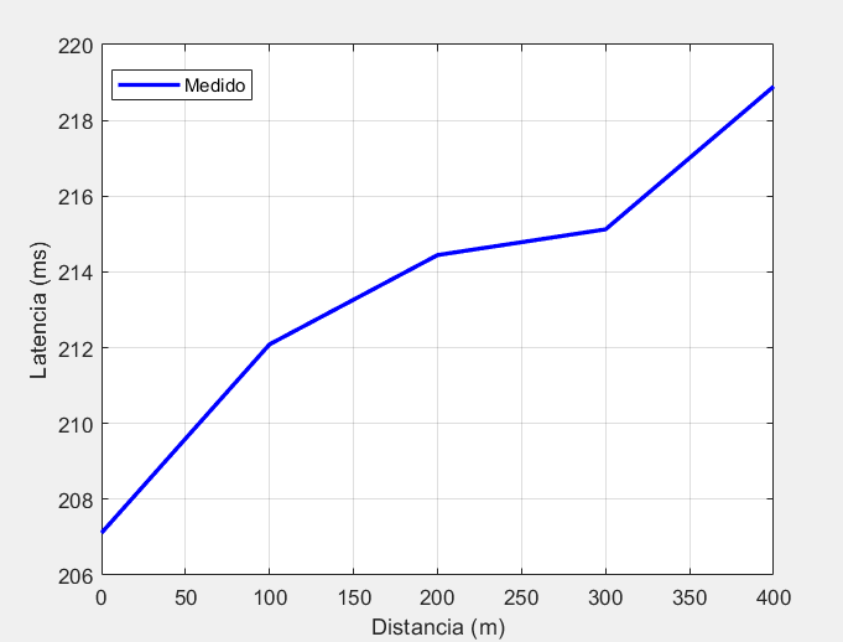

Figura 3.7: Distancia vs Latencia con SF=5

Para el cálculo de la latencia se realizó el siguiente código en Matlab.

```
% Datos del evento
time = '2024-07-16T18:22:39.831211735+00:00';
ns_time = '2024-07-16T18:22:39.616252334+00:00';
% Convertir strings a objetos datetime
time_dt = datetime(time, 'InputFormat', 'TimeZone', 'UTC');
ns_time_dt = datetime(ns_time, 'InputFormat', 'TimeZone', 'UTC');
% Calcular la latencia
latency = time_dt - ns_time_dt;
% Convertir la latencia a milisegundos
latency_ms = milliseconds(latency);
fprintf('Latencia: %.3f ms\n', latency_ms);
```

3.3. Análisis de datos experimentales

En el primer experimento se pudo evidenciar que el consumo de energía del sistema compuesto por el ESP32, el sensor MQ135, el sensor SHTC3 y el GPS NEO-6M es mayor al calculado inicialmente. Esto se debe a que el consumo real de cada sensor y del ESP32 varía con respecto a las especificaciones del datasheet debido a las variaciones en su construcción y operación.

Inicialmente, se calculó el consumo energético basado en las especificaciones de los datasheets de cada componente:

ESP32: 80 mA

MQ135: 60 mA

SHTC3: 0.002 mA

GPS NEO-6M: 45 mA

Esto daba un consumo total aproximado de 185 mA. Sin embargo, las mediciones reales mostraron que el consumo era superior, posiblemente debido a variaciones en el entorno de operación, picos de corriente no considerados y la ineficiencia de ciertos componentes. Con las mediciones realizadas en el experimento, se pudo obtener un consumo más preciso para cada componente. Los resultados

corregidos fueron:

ESP32: 85 mA por picos de corriente y operación real.

MQ135: 65 mA considerando el tiempo de calentamiento.

SHTC3: 0.5 mA variaciones en modo activo.

GPS NEO-6M: 50 mA (por picos de adquisición de señal)

El nuevo consumo de batería es de 200 mAHora

Para mejorar aún más la autonomía y asegurar la operación continua del sistema, se añadió un panel solar junto con un módulo de carga. El panel solar permite la recarga de la batería durante las horas de luz solar, prolongando así la vida útil del sistema.

La figura 3.3 muestra que la señal recibida se mantiene relativamente estable y fuerte entre -70 dBm y -60 dBm desde los 100 y 200 m cuando la distancia era pequeña con respecto al nodo, pero experimenta una disminución significativa a partir de los 300m cuando nos alejamos del nodo, alcanzando alrededor de -100 dBm cuando alcanzamos la distancia máxima de 1Km. Esta caída puede deberse a un aumento de interferencias, cambios en las condiciones ambientales o la presencia de obstáculos. Para mitigar estos efectos, se recomienda monitorear continuamente el RSSI, optimizar la ubicación de los nodos y gateways, y considerar la implementación de redundancia en la red LoRaWAN.

La gráfica de SNR (Signal-to-Noise Ratio) muestra la variación de la relación señal-ruido a lo largo de la distancia . Inicialmente, entre los 100 y 200m, el SNR se mantiene constante y alto cuando el nodo estaba a una corta distancia del gateway, con alrededor de 13-14 dB, indicando una señal significativamente más fuerte que el ruido de fondo. Sin embargo, después de los 300m cuando el nodo comenzó a alejarse del gateway, el SNR disminuye notablemente, alcanzando alrededor de 8 dB cuando alcanzamos la distancia máxima de 1Km, lo que sugiere una mayor influencia del ruido o una señal debilitada.

En la figura 3.5 muestra que al utilizar un Spread Factor (SF) de 12 en una red LoRaWAN, la latencia aumenta conforme se incrementa la distancia entre el transmisor y el receptor. A distancias menores entre 0 a 200 metros, la latencia se mantiene relativamente baja y estable, alrededor de 210 ms. A medida que la

distancia aumenta de 200 a 400 metros, la latencia se incrementa ligeramente hasta aproximadamente 215 ms. A partir de 400 metros, el aumento de la latencia se vuelve más pronunciado, especialmente a partir de 800 metros, alcanzando hasta 245 ms a 1000 metros. Este comportamiento refleja la mayor dificultad de mantener la calidad de la señal y el tiempo de transmisión en mayores distancias.

En la figura 3.6 muestra la relación entre la distancia (m) y la latencia (ms) utilizando un Spread Factor (SF) de 9 en una red LoRaWAN. La latencia comienza alrededor de 209 ms a 0 metros y aumenta de manera constante a medida que la distancia incrementa, alcanzando aproximadamente 218 ms a 700 metros. Este aumento es más gradual y lineal en comparación con el Spread Factor de 12, sugiriendo que con un SF de 9, la latencia es menos sensible a incrementos en la distancia, aunque el alcance máximo posible podría ser menor en comparación con un SF más alto.

En la figura 3.7 muestra la relación entre la distancia (m) y la latencia (ms) utilizando un Spread Factor (SF) de 5 en una red LoRaWAN. La latencia comienza alrededor de 206 ms a 0 metros y aumenta de manera constante a medida que la distancia incrementa, alcanzando aproximadamente 220 ms a 400 metros. Este incremento es más notable en distancias cortas y medias en comparación con SF más altos, lo que sugiere que con un SF de 5, aunque la latencia inicial es baja, se incrementa rápidamente con la distancia. Esto refleja que un SF más bajo, como el 5, es menos eficiente para mantener una baja latencia en distancias mayores, a pesar de su ventaja inicial en proximidad cercana.

Capítulo 4

Capítulo IV: Discusión

4.1. Interpretación de los resultados

En el primer experimento, se observaron discrepancias entre el consumo teórico calculado y el consumo real medido. Las especificaciones de los datasheets proporcionaron los siguientes valores de corriente para los componentes del sistema:

ESP32: 80 mA

MQ135: 60 mA

SHTC3: 0.002 mA

GPS NEO-6M: 45 mA

El consumo total inicial estimado fue de aproximadamente 185 mA.

Las mediciones reales indicaron un mayor consumo de energía debido a varios factores, como picos de corriente y condiciones operativas específicas. Las corrientes medidas fueron:

ESP32: 85 mA

MQ135: 65 mA

SHTC3: 0.5 mA

GPS NEO-6M: 50 mA

El nuevo consumo total real es de 200.5 mA.

Inicialmente, se utilizó una batería de 4500 mAHora. Con el consumo real de

200.5 mA, la autonomía de la batería se calculó como:

$$Autonomia = \frac{4500mA/Hora}{200{,}5mAHora} = 22{,}4horas \tag{4.1}$$

Para mejorar la autonomía, se decidió cambiar a una batería de 8800 mAh:

$$Autonomia = \frac{8800mA/Hora}{200{,}5mAHora} = 43{,}89horas \tag{4.2}$$

Para prolongar aún más la autonomía y garantizar el funcionamiento continuo del sistema, se añadió un panel solar con un módulo de carga. Suponiendo que el panel solar aporte 250 mAHora por hora durante las horas de luz solar (asumimos 6 horas de luz solar efectiva por día):

$$AportePanel = 250mAx6horas = 1500mAHora \tag{4.3}$$

Este aporte reduce la descarga neta diaria de la batería, extendiendo aún más la autonomía del sistema.

En cuanto a la segunda prueba observamos en la figura 3.3, que el RSSI y el SNR son dos métricas clave que reflejan la calidad de la señal recibida por un nodo desde el gateway. Al analizar cómo estos valores cambian con la distancia entre el nodo y el gateway, podemos entender mejor la cobertura y la eficiencia de la red.

A medida que aumenta la distancia entre el nodo y el gateway, el RSSI disminuye. Esto se debe a la atenuación de la señal a medida que viaja a través del medio. Al igual que el RSSI, el SNR tiende a disminuir a medida que la distancia aumenta. Esto se debe a que la señal se debilita y se vuelve más comparable al nivel de ruido de fondo.

Por último, en la pruebas finales se evidenció que en LoRaWAN, el Spread Factor (SF) es un parámetro crucial que influye en varios aspectos clave de la comunicación, incluyendo el alcance y la tasa de datos. Un Spread Factor bajo, como SF7, proporciona un alcance más corto, lo que lo hace adecuado para entornos urbanos o interiores donde las distancias de comunicación son relativamente pequeñas. Sin embargo, este menor alcance se compensa con una mayor tasa de datos, permitiendo

que más información se transmita en menos tiempo, lo cual es beneficioso para aplicaciones que requieren transmisiones frecuentes y rápidas. Además, un SF más bajo implica un menor consumo de energía, lo que es ideal para dispositivos alimentados por batería que necesitan maximizar su vida útil operativa.

Por otro lado, un Spread Factor alto, como SF12, ofrece un alcance significativamente mayor, lo que es ventajoso en entornos rurales o áreas con obstáculos donde se requiere cubrir distancias más largas. No obstante, este mayor alcance viene acompañado de una tasa de datos más baja, resultando en transmisiones de información más lentas. Adicionalmente, el uso de un SF alto incrementa el consumo de energía, ya que los dispositivos necesitan más tiempo para transmitir los datos, lo cual puede ser una consideración importante para aplicaciones donde la eficiencia energética es vital.

4.2. Comparación con estudios previos

En este trabajo, se ha implementado un sistema para la detección temprana de incendios forestales basado en LoRaWAN que ofrece mejoras significativas en términos de seguridad, eficiencia en el uso de la batería y alcance de la señal en comparación con métodos que emplean WiFi, como los propuestos por Metha, Kumari, Venkantesan, Ishitha y Dasari [5], [30], [32]-[34]. Además, en contraste con el trabajo de Khennou, que también utiliza LoRaWAN, nuestro sistema se destaca por la seguridad mejorada de la página web de monitoreo, aunque reconocemos la superioridad de las técnicas de deep learning empleadas por Khennou [36] en la toma de decisiones.

La seguridad se mejora considerablemente con LoRaWAN, ya que utiliza encriptación de extremo a extremo, lo que asegura que los datos transmitidos estén protegidos contra interceptaciones y accesos no autorizados. Esto contrasta con WiFi, donde la configuración y el mantenimiento de las medidas de seguridad pueden ser más complejos y menos consistentes.

LoRaWAN está diseñado para aplicaciones de baja potencia, permitiendo que los dispositivos funcionen durante meses o incluso años con una sola carga de batería,

debido a su bajo consumo energético durante la transmisión de datos. En cambio, WiFi consume significativamente más energía debido a la necesidad de mantener una conexión constante y a la mayor potencia requerida para las transmisiones.

LoRaWAN ofrece un alcance de señal mucho mayor que WiFi, con capacidades de transmisión de varios kilómetros, lo que es ideal para aplicaciones en áreas extensas o rurales. Por otro lado, WiFi tiene un alcance limitado, típicamente hasta unos pocos cientos de metros, lo que puede ser insuficiente para ciertas aplicaciones de monitoreo remoto.

En comparación con el trabajo de Khennou [36], que también utiliza LoRaWAN, nuestro enfoque se distingue en los siguientes aspectos:

Seguridad de la Página Web: Nuestro sistema se destaca por incorporar medidas adicionales de seguridad en la página web de monitoreo. Esto incluye autenticación de usuarios y encriptación de datos en tránsito.

Deep Learning y Toma de Decisiones: Reconocemos que el sistema de Khennou, al utilizar técnicas de deep learning, es superior en la toma de decisiones automatizada. Estas técnicas permiten análisis más complejos y precisos de los datos recogidos, lo que puede mejorar la detección de eventos y la respuesta automatizada.

Capítulo 5

Capítulo V: Conclusiones y Recomendaciones

5.1. Conclusiones

El desarrollo del prototipo para la detección temprana de incendios forestales mediante una plataforma IoT ha demostrado ser una solución prometedora en la lucha contra este devastador fenómeno. A través de la integración de diversos sensores como calor, humedad, CO2 con tecnologías de comunicación, en el presente trabajo se ha logrado establecer un sistema de detección válido hasta 1 Km a la redonda para monitorizar las condiciones ambientales en áreas forestales y detectar posibles incendios en etapas iniciales.

Las pruebas y evaluaciones realizadas en el capítulo 3 han demostrado que el prototipo es efectivo en la detección de incendios forestales, lo que permite a las autoridades pertinentes responder rápidamente y oportunamente. Las mediciones de la latencia que se pueden observar en las figuras 3.5, 3.6 y 3.7 ayudará a mejorar el sistema de detección para lograra la configuración que mejor se destaque según el medio. Además, se destaca la escalabilidad y la versatilidad del sistema por el uso de LoRa y LoRaWAN, que puede adaptarse a diferentes entornos forestales y requerimientos específicos de monitoreo.

La integración de LoRa, LoRaWAN con un frontend y un backend ha mostrado su potencial para mejorar la precisión y eficiencia en la detección temprana de

incendios. El diseño del sistema de sensores, junto con una base de datos bien estructurada y una interfaz web intuitiva, permite una monitorización continua y accesible para los usuarios, facilitando la toma de decisiones informadas.

A pesar de los avances logrados, el prototipo enfrenta varios desafíos y limitaciones que deben ser abordados en estudios futuros. La cobertura de red en áreas rurales alcanzó un máximo de 1Km como se observa en la figura 3.5, la durabilidad de la batería de máximo 44 horas según lo calculado con la ecuación 4.2. Todos estos aspectos en conjunto con la precisión de los sensores como el GPS son aspectos críticos que pueden mejorarse para incrementar la confiabilidad y robustez del sistema.

La implementación de un sistema de detección temprana de incendios forestales basado en IoT tiene el potencial de transformar la manera en que se manejan y previenen los incendios forestales. Con mejoras continuas en la tecnología de sensores, la optimización de algoritmos de análisis de datos y la integración con sistemas de gestión de emergencias, este prototipo puede convertirse en una herramienta vital para la protección de los ecosistemas forestales y la mitigación de riesgos.

5.2. Recomendaciones

Para maximizar el impacto del prototipo, se recomienda la continuación del estudio comenzando con proyectos piloto en áreas específicas de nuestro país. Además, es fundamental asegurar la financiación adecuada para la instalación, mantenimiento y actualización del sistema, así como para la capacitación del personal, comunidades y demás actores sociales. Por último, se sugiere implementar sistemas de detección en la nube, como AWS, permitirá una gestión eficiente y escalable, garantizando el monitoreo y análisis de datos en tiempo real, lo que facilitará la toma de decisiones informadas y la mejora continua del sistema.

5.3. Limitaciones del estudio

La detección temprana de incendios forestales mediante una plataforma IoT enfrenta desafíos significativos debido a la cobertura de red en áreas remotas. Muchas zonas forestales carecen de una cobertura de red adecuada, lo que dificulta la transmisión de datos en tiempo real desde los sensores hasta el sistema central. Además, los dispositivos IoT deben ser energéticamente eficientes o contar con fuentes de energía sostenibles, como paneles solares, para operar de manera continua en entornos donde el acceso a la electricidad es limitado.

Los sensores utilizados en el sistema deben ser lo suficientemente sensibles y precisos para detectar señales tempranas de incendios sin generar falsas alarmas. La durabilidad de los sensores es otro factor crítico, ya que deben ser capaces de soportar condiciones ambientales adversas como humedad, altas temperaturas y exposición prolongada al sol.

La recolección continua de datos de múltiples sensores puede generar grandes volúmenes de información, lo que requiere soluciones robustas de almacenamiento y procesamiento. Además, la latencia en la transmisión y procesamiento de datos puede retrasar las alertas tempranas, comprometiendo la efectividad del sistema en la prevención de incendios.

5.4. Sugerencias para futuras investigaciones

Los estudios futuros pueden centrarse en el desarrollo de sensores más avanzados y específicos para la detección temprana de incendios forestales. Sensores multifuncionales que puedan detectar no solo temperatura y humo, sino también otros indicadores como niveles de CO2, humedad del suelo y patrones de viento, pueden mejorar la precisión y reducir las falsas alarmas. En el caso del GPS se puede mejorar su eficacia implementando GPS diferencial con el cual se evitarían errores de ubicación.

Otros aspecto a considerar es la eficiencia energética de los dispositivos IoT es crucial para su funcionamiento en áreas remotas. Investigaciones futuras pueden explorar el desarrollo de sensores y dispositivos con consumo ultra bajo de energía,

además de incorporar fuentes de energía renovable como paneles solares de alta eficiencia y sistemas de recolección de energía ambiental.

Además en el futuro, se podría estudiar la integración del prototipo con sistemas de gestión de emergencias existentes puede mejorar la respuesta coordinada ante incendios forestales. Esto incluye la interoperabilidad con sistemas de alerta temprana, sistemas de información geográfica y plataformas de comunicación de emergencia.

En futuras investigaciones pueden evaluar el impacto ambiental y social de la implementación de sistemas IoT para la detección de incendios. Esto incluye el estudio de la aceptación comunitaria, la efectividad en la reducción de daños ambientales y económicos, y la contribución a la sostenibilidad y conservación de los bosques.

Adicionalmente, se puede implementar la automatización de ciertos procesos, como el registro masivo de nuevos usuarios a través de hojas electrónicas, optimizando así la eficiencia operativa y reduciendo el margen de error humano. Por último, se sugiere investigar cómo aumentar las distancias de cobertura, ya que son muy importantes para este tipo de aplicaciones, asegurando así una mayor eficacia y alcance del sistema implementado.

Glosario

DB Database.

GPS Global Positioning System.

IoBDT Internet of Biodegradable Things.

IoBT Internet of Battlefield Things.

IoIT Internet of Ingestible Things.

IoMT Internet of Medical Things.

IoT Internet de las Cosas (por sus siglas en inglés: Internet of Things).

IoTM Internet of Military Things.

LoRa Long Range.

Referencias

[1] M. Castillo, P. Pedernera y E. Pena, «Incendios forestales y medio ambiente: una síntesis global,» *Revista Ambiente y Desarrollo*, vol. 19, n.º 3, págs. 44-53, 2003.

[2] *Controlado un 70 % del incendio forestal en el Parque Nacional El Cajas, Azuay – Secretaría Nacional de Gestión de Riesgos*. dirección: https://www.gestionderiesgos.gob.ec/controlado-un-70-del-incendio-forestal-en-el-parque-nacional-el-cajas-azuay/# (visitado 29-03-2024).

[3] *30 incendios forestales afectaron a Cuenca*, es. dirección: https://www.primicias.ec/noticias/sucesos/incendios-forestales-cuenca-septiembre/ (visitado 26-04-2024).

[4] S. Park, K. Han y K. Lee, «A study on fire detection technology through spectrum analysis of smoke particles,» oct. de 2020, págs. 1563-1565. DOI: 10.1109/ICTC49870.2020.9289272.

[5] K. Mehta, S. Sharma y D. Mishra, «Internet-of-Things Enabled Forest Fire Detection System,» págs. 20-23, 2021. DOI: 10.1109/I-SMAC52330.2021.9640900.

[6] C. Nagolu, C. Cheekula, D. Sai Kiran Thota, K. Padmanaban y D. Bhattacharyya, «Real-Time Forest Fire Detection Using IoT and Smart Sensors,» págs. 1441-1447, 2023. DOI: 10.1109/ICICT57646.2023.10134063.

[7] J. Salazar y S. Silvestre, *Internet de las cosas*. České vysoké učení technické v Praze, 2016.

[8] E. Baccelli, *Internet de las cosas (IoT) - Retos sociales y campos de investigación científica en relación con la IoT*. České vysoké učení technické v Praze, 2022.

[9] D. Rodríguez, «Arquitectura y Gestión de la IoT,» *Revista Telem@tica*, vol. 12, n.º 03, págs. 49-60, 2013.

[10] Y. Li, W. Chen, Y. Ding, Y. Qie y C. Zhang, «A Vision of Intelligent IoT — Trends, Characteristics and Functional Architecture,» *IEEE internet of things journal*, págs. 184-189, 2022.

[11] A. Zanella, N. Bui, A. Castellani, L. Vangelista y M. Zorzi, «Internet of Things for Smart Cities,» *IEEE Internet of Things Journal*, vol. 1, n.º 1, págs. 22-32, 2014. DOI: 10.1109/JIOT.2014.2306328.

[12] C. Brewster, I. Roussaki, N. Kalatzis, K. Doolin y K. Ellis, «IoT in Agriculture: Designing a Europe-Wide Large-Scale Pilot,» *IEEE Communications Magazine*, vol. 55, n.º 9, págs. 26-33, 2017. DOI: 10.1109/MCOM.2017.1600528.

[13] S. Chen, H. Xu, D. Liu, B. Hu y H. Wang, «A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective,» *IEEE Internet of Things Journal*, vol. 1, n.º 4, págs. 349-359, 2014. DOI: 10.1109/JIOT.2014.2337336.

[14] A. Khanna y S. Kaur, «Internet of Things (IoT), Applications and Challenges: A Comprehensive Review,» *Wireless Personal Communications*, vol. 114, n.º 2, págs. 1687-1762, sep. de 2020, ISSN: 1572-834X. DOI: 10.1007/s11277-020-07446-4. dirección: https://doi.org/10.1007/s11277-020-07446-4.

[15] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari y M. Ayyash, «Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,» *IEEE Communications Surveys and Tutorials*, vol. 17, n.º 4, págs. 2347-2376, 2015.

[16] C. Bell, «Introducing the Raspberry Pi Pico,» en *Beginning MicroPython with the Raspberry Pi Pico: Build Electronics and IoT Projects*, Springer, 2022, págs. 1-42.

[17] M. Babiuch, P. Foltýnek y P. Smutný, «Using the ESP32 Microcontroller for Data Processing,» en *2019 20th International Carpathian Control Conference (ICCC)*, 2019, págs. 1-6. DOI: 10.1109/CarpathianCC.2019.8765944.

[18] N. Hernández-Hostaller, «Evaluación de tecnologías de sensores para la detección temprana de incendios forestales,» *Revista Tecnología en Marcha*, vol. 29, n.º 4, págs. 123-138, 2016.

[19] S. Devalal y A. Karthikeyan, «LoRa Technology - An Overview,» en *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2018, págs. 284-290. DOI: 10.1109/ICECA.2018.8474715.

[20] M. A. Almuhaya, W. A. Jabbar, N. Sulaiman y S. Abdulmalek, «A survey on Lorawan technology: Recent trends, opportunities, simulation tools and future directions,» *Electronics*, vol. 11, n.º 1, pág. 164, 2022.

[21] J. Haxhibeqiri, E. De Poorter, I. Moerman y J. Hoebeke, «A survey of LoRaWAN for IoT: From technology to application,» *Sensors*, vol. 18, n.º 11, pág. 3995, 2018.

[22] A. Moiseev e Y. Fain, *Angular Development with TypeScript*. Simon y Schuster, 2018.

[23] R. Camps Paré, L. A. Casillas Santillán, D. Costal Costa, M. Gibert Ginestà, C. Martín Escofet y O. Pérez Mora, «Bases de datos: Software libre,» 2007.

[24] *¿Qué es una base de datos?* es-ES, mar. de 2024. dirección: https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/bases-de-datos/ (visitado 06-05-2024).

[25] A. Meier y M. Kaufmann, «NoSQL Databases,» en *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019, págs. 201-218, ISBN: 978-3-658-24549-8. DOI: 10.1007/978-3-658-24549-8_7. dirección: https://doi.org/10.1007/978-3-658-24549-8_7.

[26] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes y J. Abramov, «Security Issues in NoSQL Databases,» en *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, 2011, págs. 541-547. DOI: 10.1109/TrustCom.2011.70.

[27] S. Winkelmann, M. Büttner, D. Deivasihamani, A. von Hoffmann y F. Flohr, «Using Node-RED as a Low-Code Approach to Model Interaction Logic of Machine-Learning-Supported eHMIs for the Virtual Driving Simulator Carla,» en *Adjunct Proceedings of the 15th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 2023, págs. 323-326.

[28] K. Mehta, S. Sharma y D. Mishra, «Internet-of-Things Enabled Forest Fire Detection System,» págs. 20-23, 2021. DOI: 10.1109/I-SMAC52330.2021.9640900.

[29] C. Nagolu, C. Cheekula, D. Sai Kiran Thota, K. Padmanaban y D. Bhattacharyya, «Real-Time Forest Fire Detection Using IoT and Smart Sensors,» págs. 1441-1447, 2023. DOI: 10.1109/ICICT57646.2023.10134063.

[30] V. Kumari y F. A. Ali, «Early Detection of Forest Fire Using Internet of Things,» págs. 24-28, 2022. DOI: 10.1109/ICIDeA53933.2022.9970130.

[31] K. Kumar, S. Sharma, P. Pandey y H. R. Goyal, «IoT Enabled Crop Detection System using Soil Analysis,» págs. 323-327, 2022. DOI: 10.1109/ICCES54183.2022.9835959.

[32] C. Venkateswara Rao, C. Vandana, M. K. V. S. Reddy, K. S. Raju, R. V. P. Sirisha y H. Killamsetti, «Advanced Forest Fire Alert System with Real-time GPS Location Tracking,» págs. 95-99, 2023. DOI: 10.1109/ICACRS58579.2023.10404409.

[33] S. Ishitha, S. Nagaraju, H. A. Mohan, M. Harshitha, G. R. Gowda y J. N., «IoT based Anti-poaching and Fire Alarm System for Forest,» 2021. DOI: 10.1109/MysuruCon52639.2021.9641539.

[34] P. Dasari, G. K. J. Reddy y A. Gudipalli, «Forest fire detection using wireless sensor networks,» *International journal on smart sensing and intelligent systems*, vol. 13, n.º 1, págs. 1-8, 2020.

[35] J. Ananthi, N. Sengottaiyan, S. Anbukaruppusamy, K. Upreti y A. K. Dubey, «Forest fire prediction using IoT and deep learning,» *International Journal of Advanced Technology and Engineering Exploration*, vol. 9, n.º 87, págs. 246-256, 2022.

[36] F. Khennou, J. Ghaoui y M. A. Akhloufi, «Forest fire spread prediction using deep learning,» en *Geospatial Informatics XI*, K. Palaniappan, G. Seetharaman y J. D. Harguess, eds., International Society for Optics y Photonics, vol. 11733, SPIE, 2021, pág. 117330I. DOI: 10.1117/12.2585997. dirección: https://doi.org/10.1117/12.2585997.

[37] G. Electrostore, Módulo TP4056 5V Micro USB 1A cargador para baterías litio con protección. dirección: https://grupoelectrostore.com/shop/modulos-y-shields/cargadores-para-baterias/modulo-tp4056-5v-micro-usb-1a-cargador-para-baterias-litio-con-proteccion/.

[38] M. Libre, Panel solar mini 5V 250mA 1.25W 110x69mm monocristalino. dirección: https://articulo.mercadolibre.com.co/MCO-817828846-panel-solar-mini-5v-250ma-125w-110x69mm-monocristalino-_JM.

[39] Anonymous. dirección: https://github.com/arduino/ArduinoCore-avr/blob/master/cores/arduino/Arduino.h.

[40] Sparkfun. dirección: https://github.com/sparkfun/SparkFun_SHTC3_Arduino_Library.

[41] G.Krocker. dirección: https://github.com/GeorgK/MQ135/blob/master/MQ135.h.

[42] M. Hart. dirección: https://github.com/neosarchizo/TinyGPS.

[43] Beelan. dirección: `https://github.com/GrumpyOldPizza/ArduinoCore-stm32l0/blob/master/libraries/LoRaWAN/src/LoRaWAN.h`.

[44] N. Zambetti. dirección: `https://github.com/SodaqMoja/Wire/blob/master/src/Wire.h`.

[45] D. E. Almeida Correa, «Diseño e implementación de un sistema de localización y rastreo GPS de lanchas pesqueras utilizando comunicación inalámbrica,» B.S. thesis, Universidad Politécnica Salesiana, 2022.