UNIVERSIDAD NACIONAL DEL COMAHUE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE ELECTROTÉCNIA

AUTOMATIZACIÓN INTELIGENTE Y SOSTENIBLE PARA EL ABASTECIMIENTO DEL AGUA POTABLE EN ALUMINÉ UTILIZANDO INTERNET DE LAS COSAS PARA LA SUPERVISIÓN Y EL CONTROL

Proyecto Integrador Profesional presentado por:

FRANCHERI DAVID ALEJANDRO

Ante la Facultad de Ingeniería de la Universidad Nacional del Comahue para acceder al título de:

INGENIERO ELECTRÓNICO

Dirección de tesis:

Director: ING. Daniel Simone

Neuquén, 6 de agosto de 2025

Resumen

Este trabajo aborda la problemática de la gestión del suministro y distribución de agua potable en la localidad de Aluminé, Argentina. El Proyecto Integrador Profesional (PIP) proporciona una solución automatizada para la gestión del llenado de los tanques, permitiendo activar o detener las bombas de agua en función de niveles preestablecidos. Esta automatización no solo reduce significativamente las pérdidas por desbordamiento y el consumo innecesario de energía eléctrica debido al funcionamiento inadecuado de las bombas, sino que también elimina la necesidad de intervención manual y contribuye al uso eficiente del recurso hídrico, favoreciendo la sostenibilidad ambiental. Para la automatización del sistema, dado el alcance del proyecto y las capacidades financieras de Aluminé, se optó por utilizar dispositivos inteligentes de bajo costo. En concreto, se emplearon placas de desarrollo comerciales prefabricadas de tipo NodeMCU que incorporan el SoC ESP32.

La comunicación entre los dispositivos ESP32, que controlan las bombas y monitorizan los tanques en distintas zonas, se gestionó mediante la plataforma de internet de las cosas (IoT) ThingsBoard. Esta plataforma facilita la supervisión del sistema de abastecimiento de agua. Los datos pueden visualizarse en cualquier navegador en Windows o en una aplicación específica para Android. En esta interfaz web, se podrán consultar datos como la cantidad de dispositivos conectados en cada zona, la cantidad de tanques y bombas que cada dispositivo tiene, si el dispositivo esta activo o no, y varios gráficos con información de consumo de agua, funcionamiento de las bombas, estado de los sensores en los tanques, entre otros.

La solución, además de ser de bajo costo, es fácilmente escalable en función de futuras ampliaciones que puedan ser necesarias para cubrir la creciente demanda de agua. Asimismo, su mantenimiento es mínimo, ya que solo requeriría reemplazar algún sensor o dispositivo en caso de desperfecto. La tecnología empleada en el proyecto es de amplia disponibilidad en el país, y el sistema de monitoreo y control puede ser fácilmente adoptado por el personal del municipio.

Palabras clave: (IoT, monitoreo y control, automatización inteligente, WiFi, Software).

Abstract

This work addresses the issue of potable water supply and distribution management in Aluminé, Argentina. The Professional Integrated Project (PIP) provides an automated solution for the management of tank filling, enabling the activation or deactivation of water pumps based on predefined level thresholds. This automation not only significantly reduces water losses due to overflow and unnecessary electrical energy consumption caused by improper pump operation, but also eliminates the need for manual intervention and promotes the efficient use of water resources, thereby contributing to environmental sustainability. For system automation, considering the project's scope and Aluminé's financial situation, low-cost intelligent devices were chosen. Specifically, commercially available prefabricated development boards, NodeMCU, equipped with the ESP32 SoC were used.

Communication between ESP32 devices for pump control and tank monitoring across different areas was managed through the Internet of Things (IoT) platform, ThingsBoard. This platform facilitates the observation of the water supply system's operation and can be accessed on Windows via any web browser or on Android through a dedicated app. The web interface provides data such as the number of devices in each operational area, the number of tanks and pumps managed by each device, device connectivity status, and various data graphs related to water consumption, pump operation, and tank sensor statuses.

In addition to being a low-cost solution, the system is easily scalable to accommodate future expansions that may be required to meet the growing demand for water. Moreover, maintenance requirements are minimal, as it would only involve replacing a sensor or device in the event of a malfunction. The technology employed in the project is widely available in the country, and the monitoring and control system can be easily adopted by municipal personnel.

Keywords: (IoT, monitoring and control, intelligent automation, WiFi, Software).

Dedicatorias y Agradecimientos

En primer lugar, deseo expresar mi más sincero agradecimiento a mis padres por su apoyo constante. Sin su acompañamiento incondicional, este logro no habría sido posible.

Agradezco también a los familiares que están aquí y a aquellos que ya no están con nosotros, pero cuyas enseñanzas continúan guiándome hasta el día de hoy.

Mis compañeros de carrera merecen un reconocimiento especial por los consejos y el apoyo brindado a lo largo del tiempo, ya sea de manera directa o indirecta, que han sido importantes para la realización de este trabajo.

Estoy muy agradecido con todas las personas que, ya sea a través de su motivación personal o su experiencia, dedicaron tiempo para aconsejarme y contribuyeron, quizás sin darse cuenta, a la realización de este proyecto.

Un agradecimiento especial va para mi tutor, el Ing. Daniel Simone, por la confianza y la orientación que me ofreció durante el desarrollo de este proyecto, facilitando y guiando su culminación.

Finalmente, quiero expresar mi gratitud a la Universidad Nacional del Comahue, y en particular a la Facultad de Ingeniería y sus docentes, por estos años de educación de alta calidad y por el papel fundamental que desempeñan como institución pública en la formación académica.

Índice general

Lista de figuras

Lista de tablas

1 Introducción y Fundamentación

1.1 Introducción

En las últimas décadas, los avances en automatización y en el Internet de las Cosas (IoT) han transformado profundamente la forma en que gestionamos y controlamos sistemas complejos. La automatización ha permitido optimizar procesos previamente dependientes de intervención humana constante, logrando una mayor eficiencia y reduciendo costos operativos. Por su parte, el IoT ha revolucionado el monitoreo y control de sistemas remotos, permitiendo la recopilación de datos en tiempo real y la implementación de decisiones automatizadas. Estos avances son fundamentales para mejorar la gestión de recursos en sectores como la energía, el agua y la infraestructura pública.

La incorporación de soluciones tecnológicas inteligentes no solo ha facilitado la supervisión de dispositivos, sino que ha impulsado la sostenibilidad, reduciendo el uso innecesario de recursos. Con la conectividad cada vez más accesible y el desarrollo de dispositivos inteligentes de bajo costo, la adopción de sistemas automatizados es una realidad alcanzable para muchas comunidades o negocios que buscan mejorar su eficiencia operativa y reducir su impacto ambiental.

En este contexto, el abastecimiento de agua potable en áreas remotas con infraestructura deficitaria o inexistente presenta un reto significativo, especialmente en localidades como Aluminé, Argentina, donde la gestión del recurso hídrico es vital para el bienestar de la población. En estas circunstancias, la operación manual para la carga de tanques conlleva riesgos como el desbordamiento, el funcionamiento ineficiente de las bombas y el consecuente alto consumo energético, además de una constante dependencia de supervisión humana.

La implementación de una solución automatizada que permita controlar y monitorear las bombas de manera remota ofrece ventajas como la minimización de las pérdidas de agua, la optimización del consumo de energía y la eliminación de la intervención directa. Al utilizar dispositivos inteligentes y una plataforma IoT, se podrá gestionar el sistema de manera más eficiente, proporcionando información en tiempo real sobre el estado de los tanques y el funcionamiento de las bombas. Esta tecnología responde a la necesidad de Aluminé de mejorar la gestión de sus recursos hídricos de manera sostenible, haciendo uso de herramientas tecnológicas accesibles.

La hipótesis de este proyecto es que la automatización del sistema reducirá significativamente el desperdicio de agua y el consumo eléctrico, garantizando un suministro más confiable y eficiente. Los datos obtenidos a través de la plataforma IoT también proporcionarán información valiosa para la futura ampliación del sistema de abastecimiento de agua, en caso de que sea necesario.

El proyecto enfrenta limitaciones económicas y tecnológicas en la región, lo que influye en la elección de las tecnologías y dispositivos accesibles. No obstante, la solución busca ser adaptable y fácil de implementar, permitiendo su adopción en otras comunidades con características similares.

1.2 Revisión bibliográfica

El estado del arte en la automatización del abastecimiento de agua y su integración con la tecnología IoT ha mostrado avances significativos en la optimización de recursos y el control remoto de infraestructuras. Diversos estudios y proyectos han abordado la necesidad de gestionar eficientemente sistemas de agua en áreas tanto urbanas como rurales, reduciendo pérdidas, mejorando la sostenibilidad y facilitando la toma de decisiones basada en datos.

Un ejemplo destacado es el trabajo: "*Making urban water smart: the SMART-WATER solution*" [8], el cual es innovador, ya que integra soluciones de Internet de las Cosas (IoT) con tecnologías de análisis de datos avanzadas, mejorando significativamente la gestión de redes de suministro de agua en entornos urbanos. Este sistema se destaca por su capacidad de monitorear y controlar el sistema de distribución de agua, permitiendo una supervisión en tiempo real que optimiza tanto el uso del recurso hídrico como la eficiencia energética.

Entre las principales funcionalidades de SMART-WATER, se encuentra la telemetría, que permite recopilar datos detallados sobre el estado de la red hídrica, como el caudal, presión, calidad del agua, y detectar fugas. Estos datos son transmitidos a un servidor en tiempo real a través de válvulas con conectividad inalámbrica LoRa o comunicación Modbus, a partir de los cuales se efectúa un análisis de datos para estudiar comportamientos de consumo y demanda. Asimismo, el control remoto de las válvulas mediante este tipo de comunicación reduce la necesidad de intervención manual y minimiza el riesgo de errores humanos en la operación.

Otra característica fundamental de la plataforma es su capacidad para visualizar patrones de consumo de agua, lo que permite tanto a las compañías proveedoras como a los usuarios domésticos y comerciales ajustar sus hábitos de uso para un consumo más eficiente. A través

de algoritmos predictivos y machine learning, SMART-WATER puede anticipar la demanda de agua basándose en factores históricos y condiciones climáticas, lo que permite una planificación más eficiente y la reducción de desperdicios en momentos de alta demanda o estrés hídrico.

Además, la implementación de modelos de predicción no solo ayuda a gestionar la oferta de agua en tiempo real, sino que también optimiza la infraestructura a largo plazo. Esto resulta en una mejor distribución de los recursos, evitando sobrecargas en el sistema y promoviendo una gestión más sostenible del agua.

Este tipo de soluciones se está implementando en diversas ciudades del mundo, demostrando su efectividad en la reducción de pérdidas de agua, aumento de la eficiencia energética, y mejora del servicio de abastecimiento. La plataforma SMART-WATER está diseñada para ser escalable y flexible, lo que la convierte en una opción viable tanto para grandes ciudades como para áreas con recursos limitados (Antzoulatos et al., 2020).

Otro caso, es el artículo *"IoT Based Smart Water Quality Monitoring: Recent Techniques, Trends and Challenges for Domestic Applications"* [9], el cual profundiza en las innovaciones tecnológicas aplicadas al monitoreo de la calidad del agua mediante el uso de Internet de las Cosas (IoT). El estudio se enfoca en las técnicas recientes para la recolección de datos en tiempo real, empleando sensores inteligentes capaces de medir parámetros clave como el pH, la turbidez, la temperatura y el oxígeno disuelto. Estos sensores están conectados a redes IoT que permiten no solo la recolección remota de datos, sino también la posibilidad de analizarlos y actuar en tiempo real para garantizar la calidad del agua en entornos domésticos.

La implementación de estos sistemas ha demostrado su eficacia para reducir costos operativos y optimizar el consumo de recursos. Sin embargo, el artículo identifica desafíos que limitan su adopción masiva, especialmente en regiones con menos recursos. Estos incluyen la duración de la batería de los sensores inalámbricos, la integridad de los datos durante la transmisión y la falta de estandarización en los dispositivos IoT. Además, se señala que las soluciones actuales suelen ser costosas y requieren de infraestructura avanzada, lo que dificulta su adopción en comunidades rurales o con infraestructura limitada.

El artículo propone enfoques para superar estos desafíos, como el desarrollo de sensores más eficientes en términos energéticos y con mayores capacidades de procesamiento, lo cual podría hacer que estas tecnologías sean más accesibles. Esto es especialmente relevante para áreas

como Aluminé, donde la implementación de tecnologías asequibles y sostenibles es clave para mejorar la gestión del agua potable y garantizar el acceso a recursos hídricos de calidad.

El estudio también destaca que la integración de plataformas IoT con inteligencia artificial (IA) y modelos matemáticos permite optimizar el monitoreo de la calidad del agua. Dado que no siempre es posible medir todos los parámetros de calidad con sensores comerciales, los investigadores han desarrollado métodos para inferir parámetros importantes utilizando un subconjunto de mediciones disponibles. Por ejemplo, mediante sensores que capturan datos como pH, turbidez y el potencial de reducción de oxidación (ORP), es posible estimar otros parámetros, como el cloro libre, utilizando algoritmos de aprendizaje automático (ML). Estos modelos matemáticos basados en ML, un subconjunto de la IA, permiten realizar predicciones precisas sin necesidad de medir directamente todos los factores, lo que simplifica y reduce los costos del sistema. De esta manera, la combinación de IoT y IA no solo mejora la precisión en la gestión de recursos hídricos, sino que también facilita la implementación de soluciones más sostenibles y accesibles en comunidades con recursos limitados.

Este enfoque es clave para garantizar la eficiencia y sostenibilidad de los sistemas, ya que permite optimizar la infraestructura existente y mejorar la calidad del agua monitoreada con un menor número de dispositivos (Jan, et al., 2021).

1.3 Fundamentación

El abastecimiento de agua potable es un servicio esencial para el desarrollo y bienestar de cualquier comunidad. En la localidad de Aluminé, las limitaciones operativas del sistema actual, basado en la activación y desactivación manual de las bombas, no solo generan un consumo energético innecesario, sino que además implican el desperdicio de un recurso natural finito, como es el agua proveniente del río Aluminé, destinada al abastecimiento de la población.

Frente a esta problemática, la incorporación de un sistema automatizado para el control de las bombas representa una solución viable, efectiva y sostenible. La automatización no solo garantiza un uso más racional del recurso hídrico y de la energía eléctrica, sino que también reduce la dependencia de la intervención humana, optimizando el tiempo y esfuerzo del personal encargado.

El uso de tecnologías accesibles y de bajo costo, combinadas con la integración de plataformas IoT para monitoreo y control remoto, permite modernizar la gestión del sistema de

abastecimiento, haciéndolo escalable y adaptable a futuras necesidades de la localidad. De esta manera, este proyecto no solo resuelve una deficiencia operativa concreta, sino que también contribuye a sentar las bases para un modelo de gestión más eficiente, sostenible y acorde a las demandas actuales.

1.4 Objetivos e hipótesis

El objetivo del proyecto integrador profesional es desarrollar un sistema de automatización basado en **Internet de las Cosas (IoT)** para mejorar la eficiencia operativa y la sostenibilidad del abastecimiento de agua potable en Aluminé, Argentina.

Algunos **objetivos específicos** del proyecto incluyen:

1. Promover la eficiencia energética en el sistema de bombeo reduciendo el consumo de energía eléctrica.
2. Fomentar el uso responsable del agua del rio Aluminé, creando condiciones para la sostenibilidad ambiental.
3. Monitoreo remoto: Diseñar y desarrollar un panel de control en una aplicación WEB donde se envíe y supervise la información del nivel de agua del tanque, el estado de la bomba y su control.
4. Construcción de un prototipo en una maqueta en pequeña escala que permita modelar el funcionamiento del sistema de control y el monitoreo diseñado.

2 Marco Teórico

2.1 Dispositivos inteligentes, SoC, SoM y placas de desarrollo

En el contexto de la automatización y el Internet de las Cosas (IoT), los **dispositivos inteligentes** se han convertido en componentes clave para gestionar y controlar sistemas complejos de manera eficiente. Estos dispositivos, basados en la integración de hardware y software, permiten realizar tareas como el monitoreo remoto, la recopilación de datos en tiempo real y la automatización de procesos. La mayoría de los dispositivos inteligentes que se emplean en proyectos de IoT y automatización están construidos en base a **System on Chip (SoC)** y **System on Module (SoM)** [15], que proporcionan la potencia de procesamiento y conectividad necesarias para vincularse con el medio.

2.1.1 System on Chip (SoC)

Figura 1: SoC ESP32-S2F (Fuente: ar.mouser.com)

El **SoC** es un circuito integrado que incluye en un solo chip todos los componentes fundamentales de un sistema de procesamiento. Estos componentes incluyen una **Micro Unidad de Control (MCU)**, memoria, controladores de entrada y salida, y módulos de comunicación como **WiFi**, **LoRa**, **Bluetooth** y **Ethernet**. Esta integración de múltiples funcionalidades en un solo chip permite reducir el tamaño y el consumo de energía de los dispositivos, haciéndolos ideales para aplicaciones embebidas y sistemas de bajo consumo. En la Figura 1 se puede apreciar un ejemplo destacado de **SoC**: el **ESP32** desarrollado por Espressif Systems [16], el cual es uno de los componentes más versátiles y potentes utilizados en proyectos de Internet de las Cosas (IoT). El ESP32 es un chip de bajo costo y alta eficiencia

energética que integra tanto conectividad WiFi como Bluetooth, lo que lo hace ideal para aplicaciones que requieren una comunicación remota confiable y constante. Cuenta con un microprocesador de simple/doble núcleo Xtensa LX6 de 32 bits que puede operar a una velocidad de hasta 240 MHz, lo que le otorga una capacidad de procesamiento notable para gestionar múltiples tareas simultáneamente. Además, dispone de una amplia gama de interfaces, tales como SPI, I2C, UART, PWM, ADC y DAC, lo que permite la integración con diversos sensores y actuadores.

El ESP32 es un SoC popularmente utilizado en placas de desarrollo debido a su balance entre coste y rendimiento, facilitando la creación de dispositivos inteligentes capaces de conectarse a redes WiFi o Bluetooth. Este SoC permite tanto el procesamiento de datos en el dispositivo como su transmisión a plataformas IoT, lo que lo hace adecuado para proyectos de monitoreo y control de sistemas en tiempo real.

2.1.2 System on Module (SoM)

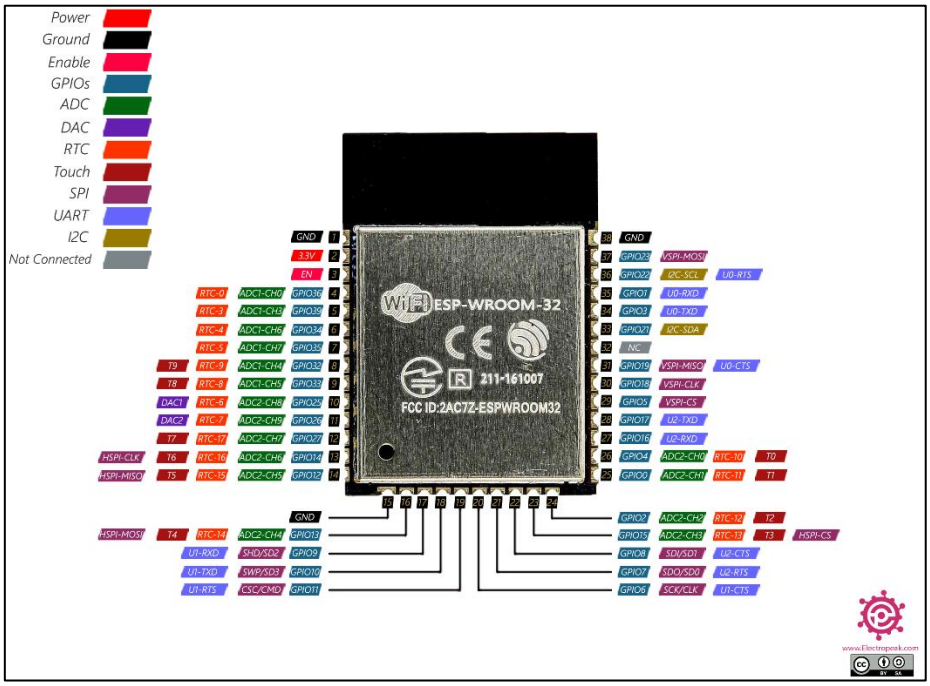


Figura 2: Módulo para ESP32-WROOM-32. (Fuente: electropeak.com)

Mientras que el **SoC** proporciona los componentes básicos de un sistema en un solo chip, el **SoM** es un módulo que integra el SoC junto con otros elementos necesarios para facilitar su uso en aplicaciones más grandes y complejas. Un SoM suele incluir no solo el SoC, sino también memoria adicional, fuentes de alimentación, interfaces de comunicación y otros componentes en una placa que se puede conectar fácilmente a otros sistemas. De esta forma, el SoM simplifica la integración de un SoC en sistemas embebidos más grandes y es

particularmente útil para aplicaciones industriales y de automatización, donde se requiere más robustez y conectividad.

Por ejemplo, el ESP32 puede formar parte de un SoM como el observado en la Figura 2, cuando se necesitan más recursos de conectividad o cuando se requiere conectar el sistema a una variedad de periféricos externos. Este enfoque modular permite que el SoM se utilice en una amplia gama de aplicaciones, desde sistemas de monitoreo ambiental hasta soluciones de automatización industrial.

Un aspecto clave es que los SoM también permiten una mayor flexibilidad en el diseño de hardware, ya que reducen la necesidad de integrar manualmente componentes adicionales en el diseño del sistema.

2.1.3 Placas de desarrollo

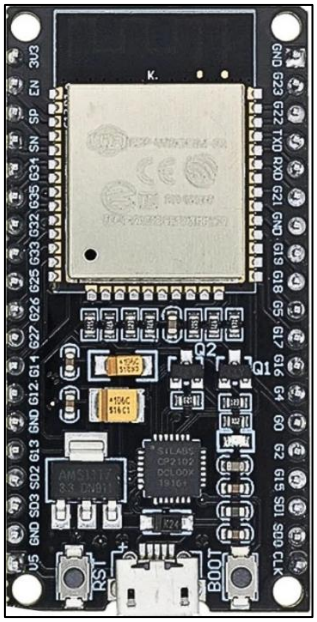


Figura 3: Placa de desarrollo NodeMCU de 38 pines para ESP32. (Fuente: arduino-projekte.info)

Para facilitar el uso de SoC y SoM en proyectos de ingeniería y desarrollo, se utilizan **placas de desarrollo** como las de la Figura 3, que son plataformas electrónicas que permiten la conexión de estos sistemas con el entorno físico, ya sea mediante sensores, actuadores o interfaces de comunicación. Las placas de desarrollo basadas en el ESP32, como el **NodeMCU**, permiten a los ingenieros y desarrolladores aprovechar las capacidades de estos sistemas integrados sin tener que diseñar desde cero una placa de circuito personalizada.

Las placas de desarrollo son esenciales en la fase de prototipado de proyectos, ya que proporcionan todas las conexiones necesarias para interactuar fácilmente con el hardware (por ejemplo, a través de un puerto serie USB) y software, permitiendo un desarrollo rápido y pruebas iterativas. Además, estas placas suelen incluir herramientas de programación y depuración, facilitando el diseño de sistemas inteligentes para aplicaciones específicas. Existen distintos modelos de estas placas para un mismo SoM, sin embargo, algunas características generales incluyen, led de salida, pines GPIO de entrada/salida, pines vinculados con funciones del SoC/SoM como ADC, DAC, PWM, comunicación I2C, SPI, UART, entre otros.

2.2 Plataformas IoT: Generalidades

Las plataformas IoT (Internet de las Cosas) permiten la integración y gestión de dispositivos conectados que recopilan y transmiten datos a través de redes. Estas plataformas facilitan el monitoreo, análisis y control de sensores, actuadores y otros dispositivos en tiempo real, proporcionando herramientas para la visualización de datos, generación de alertas y toma de decisiones automatizada. En el ámbito industrial y doméstico, las plataformas IoT ayudan a optimizar procesos, reducir costos operativos y mejorar la eficiencia de sistemas complejos, como redes de suministro de agua, energía o infraestructura de transporte.

En la Figura 4 se puede apreciar la arquitectura de una plataforma IoT, la cual presenta un diseño que abarca varios componentes y capas, cada uno con un propósito específico. Estos componentes trabajan en armonía para garantizar el funcionamiento fluido de dispositivos y aplicaciones IoT. En su núcleo, la arquitectura de una plataforma IoT consta de tres capas fundamentales [18].

- **Capa de dispositivo:** Describe los diferentes tipos de sensores y actuadores (temperatura, humedad, movimiento, etc.) y cómo se integran en la plataforma.
- **Capa de conectividad:** Abarca los protocolos de comunicación más utilizados en IoT (MQTT, CoAP, HTTP) y sus características. Menciona también las redes LPWAN (Low-Power Wide-Area Networks) como opción para dispositivos con bajo consumo energético.
- **Capa de aplicación:** Detalla las funcionalidades típicas de una capa de aplicación, como la visualización de datos en dashboards (paneles de control), el análisis de datos a través de herramientas de Business Intelligence y la integración con sistemas empresariales.

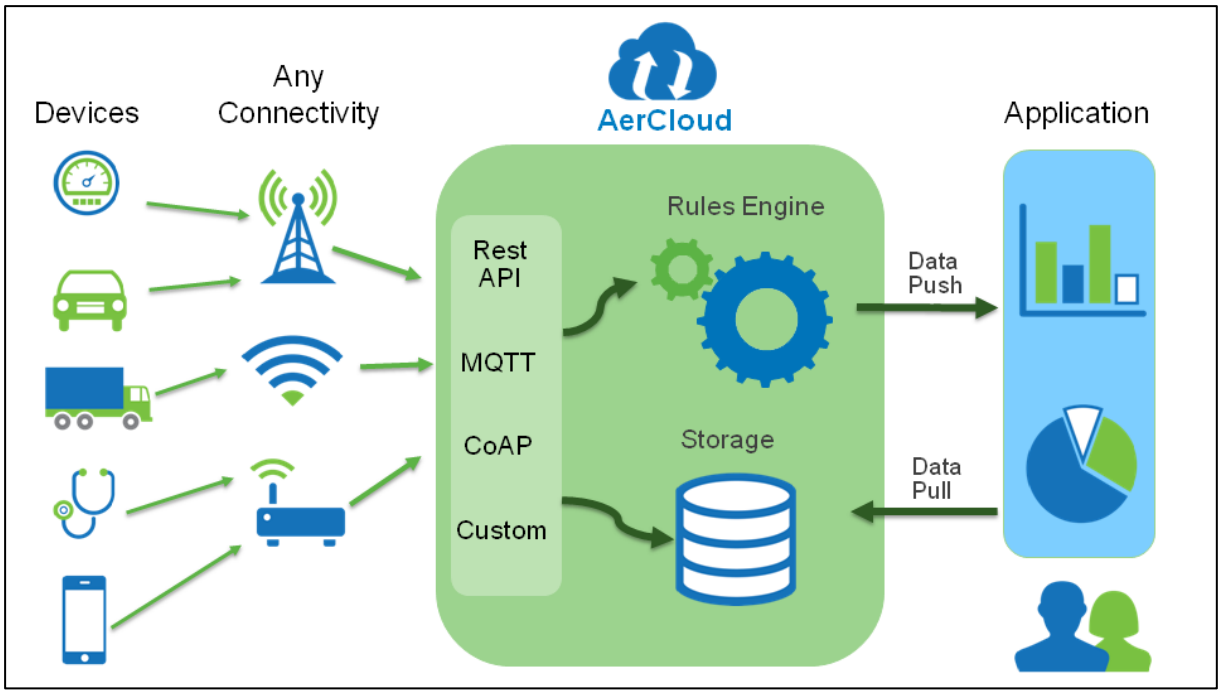


Figura 4: Arquitectura de una plataforma IoT genérica. (Fuente: aprendiendoarduino.wordpress.com)

Cada capa de la arquitectura de una plataforma IoT es esencial. Algunas de las razones por las que se necesita una arquitectura IoT sólida incluyen [19]:

Escalabilidad: Una plataforma IoT grande puede constar de cientos de dispositivos con diferentes aplicaciones y necesidades. Es muy importante evaluar las especificaciones de hardware y software y la infraestructura de red para asegurar que un sistema tan vasto pueda expandirse y escalarse con el tiempo.

Rentabilidad: Una arquitectura IoT bien optimizada no solo contribuye a reducir costos operativos, sino que también mejora la eficiencia energética al asegurar que la captura y transmisión de datos se realicen de manera eficiente. Esto implica capturar información en los intervalos adecuados, enviar únicamente los datos esenciales y gestionar de forma efectiva las capas de comunicación.

Interoperabilidad: Debe garantizar flexibilidad para adaptarse a los avances tecnológicos y facilitar la integración con otros sistemas. Es fundamental que se mantenga actualizada de manera continua, incorporando nuevas tecnologías y servicios externos, lo que permite una expansión progresiva de su funcionalidad y alcance.

2.3 ThingsBoard: Plataforma IoT

ThingsBoard es una plataforma IoT de código abierto diseñada para el monitoreo y control de dispositivos conectados, así como para la visualización y análisis de datos en tiempo real. Esta plataforma permite gestionar un amplio espectro de dispositivos IoT y es compatible con diversos protocolos de comunicación como **MQTT**, **HTTP** y **CoAP**, lo que la hace versátil para aplicaciones en múltiples industrias, incluidas la energía, el transporte y el agua.

2.3.1 Arquitectura

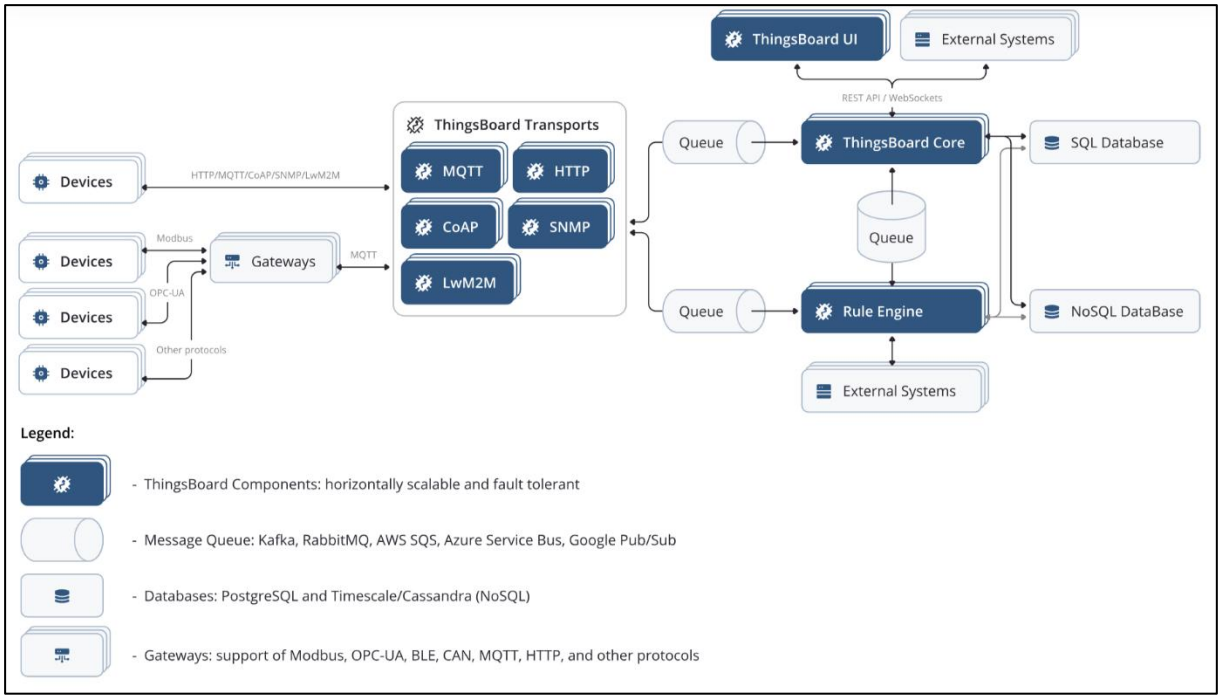


Figura 5: Arquitectura de la plataforma IoT ThingsBoard. (Fuente: thingsboard.io)

La arquitectura de ThingsBoard [21] se basa en un enfoque escalable y modular, permitiendo a los usuarios conectar cientos o incluso miles de dispositivos de manera eficiente. Entre las características más destacadas de ThingsBoard se encuentra su capacidad para añadir dispositivos, assets (activos) y clientes, procesar grandes volúmenes de datos, generar alertas basadas en eventos predefinidos, visualizar métricas a través de dashboards personalizables y controlar dispositivos mediante llamadas a procedimientos remotos o RPC, por sus siglas en inglés. Esta plataforma también permite la integración con servicios en la nube y bases de datos externas, facilitando la implementación de soluciones IoT en diferentes entornos.

En base a la arquitectura de la plataforma dada en la Figura 5, se caracterizan:

ThingsBoard Transports:

ThingsBoard utiliza diferentes protocolos para conectar y gestionar dispositivos. La capa de transporte de ThingsBoard soporta APIs basadas en MQTT, HTTP, CoAP y LwM2M para aplicaciones y firmware de dispositivos. Cada una de estas APIs están provistas por componentes del servidor y son parte de la "capa de transporte" de ThingsBoard. Una vez que un mensaje es enviado a través de la capa de transporte desde un dispositivo, se envía a una cola de mensajes "durable". Luego, el reconocimiento de recepción al dispositivo solo se confirma una vez que el mensaje ha sido procesado correctamente por la cola. Además, el transporte MQTT permite la integración de gateways que conectan múltiples dispositivos y sensores.

ThingsBoard Core:

El núcleo de ThingsBoard gestiona las llamadas **REST API** y las suscripciones **WebSocket**. También mantiene información actualizada sobre las sesiones activas de los dispositivos y su estado de conectividad. El sistema está basado en un modelo tipo "Actor", gestionando las entidades principales de la plataforma: propietarios (tenants) y dispositivos. Los nodos de la plataforma pueden unirse a un clúster, donde cada nodo gestiona una parte de los mensajes entrantes.

ThingsBoard Rule Engine:

El motor de reglas de ThingsBoard es el centro del sistema, procesando los mensajes entrantes. Este también utiliza un sistema tipo "Actor" para gestionar las cadenas y nodos de reglas. Los nodos pueden unirse a un clúster y gestionar particiones de los mensajes entrantes. El motor de reglas puede operar en dos modos: compartido, para gestionar mensajes de múltiples clientes, o aislado, para procesar mensajes de un perfil de cliente específico.

ThingsBoard Message Queue:

ThingsBoard admite múltiples implementaciones de colas de mensajes: **Kafka, RabbitMQ, AWS SQS, Azure Service Bus y Google Pub/Sub**. El uso de colas duraderas y escalables permite a ThingsBoard implementar control de retroalimentación (back-pressure) y balanceo de carga. El back-pressure es extremadamente importante en caso de picos de cargas.

ThingsBoard DataBase (SQL vs NOSQL vs Híbrida):

ThingsBoard utiliza bases de datos para almacenar entidades (dispositivos, assets, clientes, dashboards, etc.) y datos de telemetría (atributos, lecturas de sensores en series temporales, estadísticas, eventos). La plataforma admite tres opciones de bases de datos:

1. **SQL**: Almacena todas las entidades y la telemetría en una base de datos SQL. Los autores de ThingsBoard recomiendan usar **PostgreSQL**, que es la principal base de datos SQL compatible con ThingsBoard.

2. **NoSQL (obsoleto)**: Almacena todas las entidades y la telemetría en una base de datos NoSQL. Los autores de ThingsBoard recomiendan usar **Cassandra**, que es la única base de datos NoSQL compatible en este momento. Sin embargo, esta opción está en desuso en favor del enfoque híbrido debido a las muchas limitaciones de NoSQL en cuanto a transacciones y "uniones" requeridas para habilitar búsquedas avanzadas sobre entidades IoT.

3. **Híbrido (PostgreSQL + Cassandra)**: Almacena todas las entidades en la base de datos **PostgreSQL** y los datos de series temporales en **Cassandra**.

Implementaciones on-premise vs en la nube

ThingsBoard admite tanto implementaciones on-premise (en instalaciones locales) como en la nube. Con más de 5000 servidores de ThingsBoard en funcionamiento en todo el mundo, la plataforma se está ejecutando en producción en servicios de la nube como **AWS**, **Azure**, **Google Cloud Engine (GCE)**, y también en centros de datos privados. Es posible lanzar ThingsBoard en una red privada sin ningún acceso a internet, lo que permite a las organizaciones mantener el control total de sus datos.

2.3.2 Aspectos clave: Entidades y relaciones

Entidades:

- Propietarios (Tenants): Puede ser un individuo u organización que tiene o produce dispositivos y assets. Cada tenant puede tener varios administradores, millones de clientes, dispositivos y assets.

- Clientes (Customers): Puede ser un individuo u organización que compra o utiliza dispositivos y assets creados o prestados por un tenant. Cada cliente puede tener varios usuarios y millones de dispositivos y assets.
- Usuarios (Users): Los usuarios son capaces de visualizar dashboards y gestionar entidades.
- Dispositivos (Devices): Entidades IoT básicas que pueden producir telemetria, enviar atributos o gestionar llamadas RPC. (P.ej: sensor inteligente, ESP32, Raspberry Pi).
- Activos (Assets): Entidades IoT abstractas que se las puede relacionar con otros dispositivos o assets. (P.ej: Fabrica, campo, vehículo).
- Vistas de entidad (Entity Views): Útil si se desea compartir solo una parte de los datos de un dispositivo o asset a un cliente.
- Alarmas (Alarms): Eventos que identifican alertas con tus dispositivos, assets u otras entidades.
- Paneles de control (dashboards): Visualización de la información IoT a través de gráficos, tablas, mapas. También es posible controlar dispositivos desde la interfaz de usuario.
- Nodo de reglas (Rule node): Unidades de procesamiento de mensajes entrantes en una cadena de reglas.
- Cadenas de reglas (Rule Chain): Define el flujo de procesamiento de mensajes entrantes a través del motor de reglas (rule engine). Puede contener varios rule node y links hacia otros rule chain.

Cada entidad admite:

- Atributos (Attributes): Pares clave-valor (Key-Value) estáticos o semi-estáticos asociado a entidades. (P.ej: número serial, color, modelo, Firmware).
- Telemetría (Telemetry): Pares clave-valor dinámicos con el tiempo almacenables, y visualizables. (P.ej: temperatura, humedad, presión, nivel de agua).
- Relaciones (Relations): Vínculo directo entre entidades. (P.ej: Contiene, maneja, produce).

Algunas entidades admiten perfiles:

- Perfiles tenant: Características comunes para múltiples tenants: entidades, API y límites de mensajes, etc. Cada tenant solo puede tener un único perfil.

- Perfiles para dispositivos: Características comunes para múltiples dispositivos: procesamiento y configuraciones del transporte, etc. Cada dispositivo puede tener un único perfil.
- Perfiles para assets: Características comunes para múltiples assets: procesamiento y configuración. Cada asset puede tener un único perfil

Relaciones:

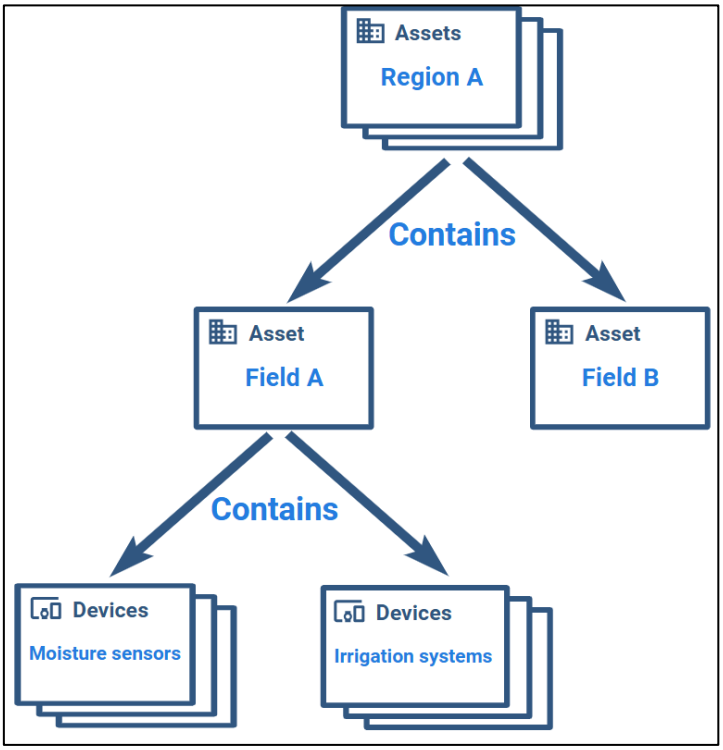


Figura 6: Ejemplo relaciones entre dispositivos y assets en ThingsBoard. (Fuente: thingsboard.io)

Las relaciones establecen vínculos entre entidades dentro de un mismo tenant. Cada relación tiene un tipo específico: puede ser "contains", "manages" o personalizado. Asimismo, cada relación posee una dirección que indica su naturaleza. Estas relaciones son útiles para modelar objetos físicos. Por ejemplo, en la Figura 6, la Región A (un asset) alberga dos campos de cultivo (también assets), los cuales están relacionados con la Región A. Además, el Campo de Cultivo A cuenta con sensores de humedad y un sistema de irrigación (dispositivos), que están integrados dentro de este campo (relación de tipo contains).

Los datos generados por los sensores pueden ser visualizados mediante gráficos interactivos, mapas o tablas, y los usuarios pueden establecer reglas de automatización que actúan en respuesta a eventos específicos, como cambios en las lecturas de sensores.

2.3.3 Atributos, telemetría y periodo de actividad

Atributos:

ThingsBoard permite asignar atributos personalizados a las entidades y gestionar estos atributos. Estos atributos se almacenan en la base de datos y pueden ser utilizados para la visualización y el procesamiento de datos.

Los atributos se manejan como pares clave-valor, la clave siempre es una cadena y, en esencia, es el nombre del atributo, mientras que el valor del atributo puede ser una cadena, booleano, doble, entero o JSON.

Hay dos tipos de atributos: del lado del servidor (Figura 7) y del lado del cliente (Figura 8).

Atributos del lado servidor:

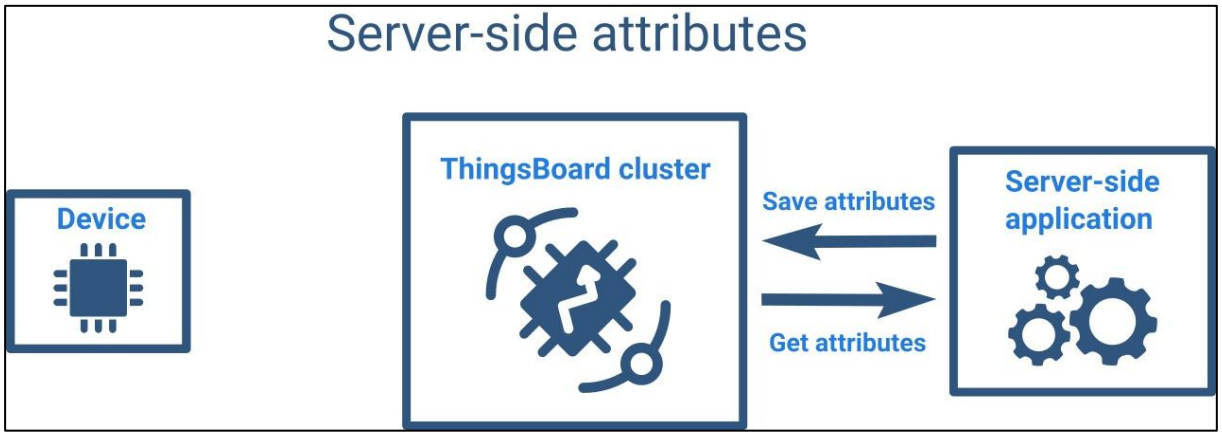


Figura 7: Atributos del lado del servidor en ThingsBoard. (Fuente: thingsboard.io)

Este tipo de atributos está permitido en casi todo tipo de entidad: Dispositivo, asset, cliente, tenant, user, etc. Los atributos del lado del servidor son aquellos que el usuario añade y configura desde el UI de Thingsboard.

Algunos ejemplos incluyen:

- Latitud y longitud para un asset de locación.
- El límite de temperatura para un dispositivo o asset.
- Alguna variable que se desee guardar en un dispositivo o asset, para luego modificarla en un rule chain.

Atributos del lado cliente:

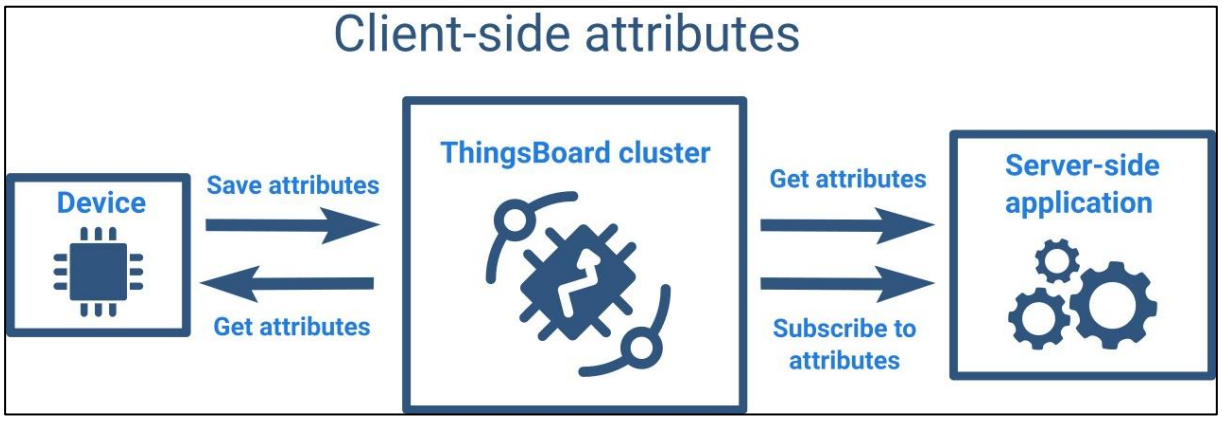


Figura 8: Atributos del lado del cliente en ThingsBoard. (Fuente: thingsboard.io)

Este tipo de atributos solamente lo tienen los dispositivos. Se los utiliza para que el dispositivo (cliente) envíe hacia la plataforma (servidor) pares clave-valor semi-estaticos. Desde el lado del servidor es posible "llamar" a los atributos para que el dispositivo los envíe vía UI/REST API pero estos son de solo lectura, el servidor no puede modificarlos.

Ejemplos comunes son los referidos a las características del dispositivo como, firmware actual, modelo, color, tamaño, estado actual, entre otros.

Telemetría:

ThingsBoard trata internamente la telemetría como pares clave-valor con una marca de tiempo (timestamp). Cada par clave-valor con una marca de tiempo se lo denomina punto de telemetría (data point). Al igual que los atributos, la clave siempre es una cadena y, en esencia, es el nombre del punto de telemetría, mientras que el valor puede ser una cadena, booleano, doble, entero o JSON. Se debe notar que si al enviar un dato de telemetría, este no tiene una marca de tiempo asignada (ts), entonces, se le asignara al punto de telemetría el timestamp actual del servidor.

Algunos ejemplos de datos de telemetría son, humedad, temperatura, presión, volumen, porcentaje de llenado, etc.

Almacenamiento de datos de telemetría

El administrador, es capaz de configurar ThingsBoard para almacenar los datos en bases SQL(PostgreSQL) o NoSQL(Cassandra). La utilización de bases de datos SQL es

recomendable para la gestión de proyectos con menos de 5000 puntos de telemetría por segundo. El uso de NoSQL se recomienda solo si el proyecto demanda volúmenes de datos por segundo muy elevados o si es necesaria una gran disponibilidad de los mismos en el tiempo.

Periodo de actividad:

ThingsBoard utiliza mensajes enviados por el dispositivo para determinar el estado de conexión que tiene con la plataforma. La plataforma reconoce como actividad alguno de los siguientes eventos:

- Conexión o desconexión de un dispositivo a la plataforma
- Enviar mensajes de telemetría o atributos a la plataforma mediane integraciones o protocolos de transporte.
- Enviar comandos RPC.
- Suscripción a actualizaciones de atributos.

Reportar el estado de actividad en la plataforma ocurre cuando alguno de estos eventos es ejecutado. De esta manera, un mensaje de conectividad “Connect event” se envía actualizando parámetros como lastConnectTime y lastActivityTime.

ThingsBoard trabaja con periodos de actividad, donde se reportan todas las actividades de un dispositivo, la duración de estos periodos es configurable al montar la plataforma en un servidor.

Si no es especificado como atributo de servidor (atributo escrito por el usuario desde la plataforma), un dispositivo tendrá como periodo de inactividad el mismo periodo de desconexión. El cual ocurre cuando no se recibe actividad por un periodo de 10 minutos en ThingsBoard Cloud.

Es posible escribir como atributo de servidor en cada dispositivo el parámetro “inactivityTimeout” con el tiempo en milisegundos para tomar un periodo de inactividad distinto. Por la lógica de la plataforma, el envío de un mensaje “Disconnect event” (al finalizar el plazo de desconexión) genera un mensaje de actividad del dispositivo, por lo tanto, desde ese momento, vuelve a correr el plazo de “inactivityTimeout” por segunda y última vez.

2.3.4 Motor de reglas (Rule Engine)

El Motor de Reglas es un marco fácil de usar para construir flujos de trabajo basados en eventos. Hay 3 componentes principales:

- Mensaje: cualquier evento entrante. Pueden ser datos que llegan de dispositivos, eventos del ciclo de vida del dispositivo, eventos de API REST, solicitudes RPC, etc.
- Nodo de Regla (Rule node): una función que se ejecuta en un mensaje entrante. Existen muchos tipos de nodos que permiten filtrar, transformar o ejecutar alguna acción en el mensaje entrante.
- Cadena de Reglas (Rule chain): los nodos están conectados entre sí mediante relaciones, por lo que el mensaje saliente de un nodo de regla se envía a los siguientes nodos de regla conectados, como si se tratase de un diagrama de bloques (nodos).

Casos de Uso Típicos

El Motor de Reglas de ThingsBoard es un marco altamente personalizable para el procesamiento complejo de eventos. Algunos casos de uso comunes que se pueden configurar a través de las Cadenas de Reglas de ThingsBoard:

- Validación y modificación de datos para la telemetría o atributos entrantes antes de guardarlos en la base de datos.
- Copiar telemetría o atributos de dispositivos a activos relacionados para poder agregar la telemetría. Por ejemplo, los datos de múltiples dispositivos pueden ser agregados en un activo relacionado.
- Crear/Actualizar/Limpiar alarmas basadas en condiciones definidas.
- Activar acciones basadas en eventos del ciclo de vida del dispositivo. Por ejemplo, crear alertas si un dispositivo está en línea/fuera de línea.
- Cargar datos adicionales necesarios para el procesamiento. Por ejemplo, cargar el valor del umbral de temperatura para un dispositivo que está definido en el atributo de Cliente o Inquilino del dispositivo.
- Activar llamadas a la API REST a sistemas externos.
- Enviar correos electrónicos cuando ocurre un evento complejo y usar atributos de otras entidades dentro de la plantilla de correo electrónico.
- Tener en cuenta las preferencias del usuario durante el procesamiento de eventos.

- Realizar llamadas RPC basadas en condiciones definidas.
- Integrarse con pipelines externos como Kafka, Spark, servicios de AWS, etc.

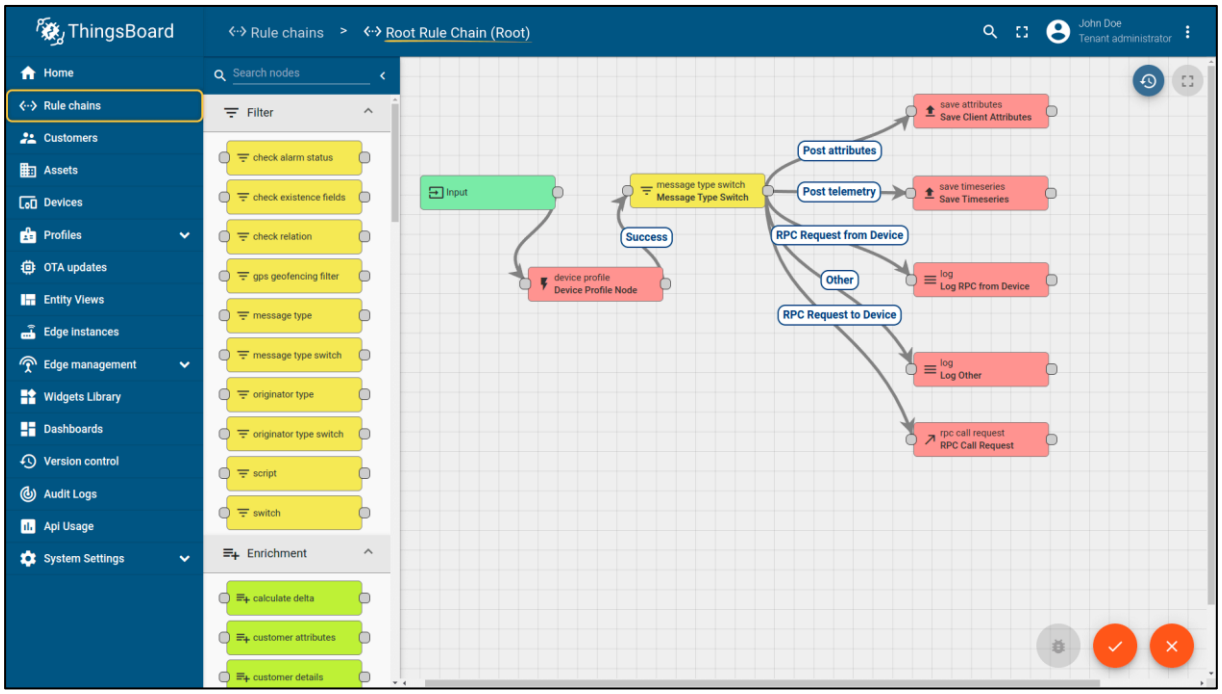


Figura 9: Interfaz WEB de Thingsboard en el Root Rule Chain (Cadena de reglas principal). (Fuente: thingsboard.io)

Existen diversos nodos para la elaboración de rule chains, algunos de estos se pueden observar en la Figura 9. Los nodos amarillos permiten la filtración de mensajes, los nodos verdes habilitan el enriquecimiento (añadido de información) en el cuerpo del mensaje o en los metadatos, los nodos celestes son de transformación del mensaje, los nodos rojos indican acciones a efectuar al mensaje, los nodos naranjas permiten la integración con sistemas externos a ThingsBoard (por ejemplo a otras bases de datos), los nodos grises permiten el flujo de información (por ejemplo el envío de mensajes a otro rule chain).

Algunos nodos relevantes utilizados en este proyecto son los siguientes:

- **Nodos Filtro:**

Asset profile switch: Este filtro funciona como switch de mensajes, permite dirigir los mensajes a distintos puntos, la conexión con los nodos subsiguientes se realiza colocando el nombre del asset profile.

Check relation presence: Busca la relación existente entre el originador del mensaje y la entidad que se defina en el filtro. Los parámetros de busca son “to originator” (relación hacia el originador) o “from originador” (relación desde el originador), luego se especifica el tipo de relacion existente (pueden ser las que son por defecto, manage, contains o puede ser una personalizada). Esto busca en términos generales todas las relaciones existentes, pero también es posible definir unívocamente la entidad con la cual existe la relación.

Message type switch: Otro filtro switch, en este caso, direcciona por tipo de mensaje, es decir, telemetría, actividad, atributo, RPC, etc.

Switch: Filtro genérico de tipo switch, se puede crear un script en javascript donde uno especifica las posibles salidas (uniones) con nodos subsiguientes.

- **Nodos de enriquecimiento:**

Originator atributes: Permite añadir al mensaje o a los metadatos del mensaje entrante atributos de tipo cliente, de tipo servidor (añadidos en la misma plataforma) o compartidos.

Originator telemetry: Permite añadir a los metadatos del mensaje entrante, valores de telemetría del originario del mensaje en un lapso de tiempo definido por el usuario. Se especifica el plazo de tiempo y se puede seleccionar agregar el primer valor del intervalo, el ultimo o “todos”. Esta última opción habilita la posibilidad de agregar información, es decir, añadir en los metadatos el promedio de valores en el lapso de tiempo fijado, o el máximo o el mínimo, etc.

- **Nodos de transformación:**

Change originator: Habilita el cambio de originador, esto es, el mensaje puede cambiar de entidad. La nueva entidad puede ser seleccionada mediante la búsqueda de relación entre el originario y la nueva entidad o se puede especificar directamente el nombre. También es posible que la nueva entidad sea un customer o un tenant.

Script: Permite cambiar por completo el cuerpo del mensaje, los metadatos o el tipo de mensaje utilizando javascript. También permite generar un nuevo tipo de mensaje. Si el cuerpo del mensaje o los metadatos o el tipo de mensaje no están especificados, el mensaje que devuelve el nodo tomara estos datos del mensaje entrante.

- **Nodos de acción:**

Device profile: Habilita las alarmas creadas en los device profile.

Save time series: Permite guardar en la base de datos de ThingsBoard Cloud pares clave:valor de telemetría. En primera instancia toma el TTL (time to live) de los metadatos del mensaje, si no lo encuentra, procede a tomar el valor del especificado en el nodo. Si se coloca "0" se toma el valor de configuración del Tenant.

Save attributes: Permite guardar lo atributos enviados desde un dispositivo en la base de datos.

Math function: Aplica una función matemática al valor del mensaje y guarda el resultado en los metadatos, en los mensajes, en un atributo o en un mensaje de telemetría. Pueden escogerse varios argumentos para la operación.

Delete attributes: Elimina atributos del originador del mensaje. Los atributos pueden ser del cliente o de tipo servidor.

2.4 Sensores para la medición del nivel de agua en sistemas hídricos

Los sensores y transductores que miden el nivel del recurso hídrico son fundamentales para la gestión eficiente del agua. Estos dispositivos permiten monitorear en tiempo real la cantidad de agua disponible en tanques, embalses y otros sistemas de almacenamiento.

2.4.1 Sensores ultrasónicos

Los sensores ultrasónicos [22] son ampliamente utilizados en robots, detección de obstáculos y desarrollo de prototipos debido a su relativo bajo costo y su capacidad para medir distancias sin contacto físico con el medio que se está analizando. Estos sensores funcionan mediante la emisión de ondas de sonido de alta frecuencia que rebotan en la superficie de un objeto y regresan al receptor del sensor. El tiempo que tarda la onda en viajar hacia el objeto y regresar se utiliza para calcular la distancia, basándose en la velocidad conocida del sonido en el aire.

A pesar de su bajo costo, las principales desventajas de este tipo de sensores son su rendimiento en zonas con objetos blandos, que absorben las ondas sonoras y dificultan su regreso al sensor, así como su efectividad en entornos con objetos pequeños y en gran número, donde el rebote de las ondas no es óptimo. Además, su precisión puede verse afectada en exteriores y particularmente en entornos líquidos, como en la medición de agua, ya que el vapor

puede causar imprecisiones. Asimismo, las mediciones pueden ser susceptibles a variaciones de temperatura si no están compensadas en la placa del sensor.

Existen diversos tipos de sensores ultrasónicos que varían en cuanto a su rango de medición y precisión. Algunos modelos están diseñados para aplicaciones de corto alcance, mientras que otros pueden medir distancias más largas.

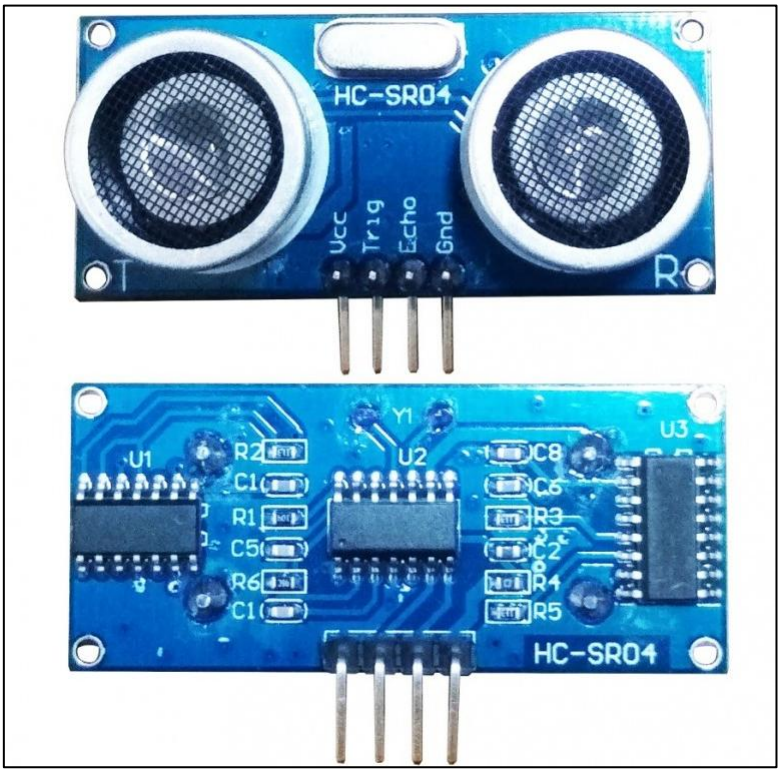


Figura 10: Sensor ultrasónico HC-SR04. (Fuente: duaitek.com.ar)

En la Figura 10 se puede observar un ejemplo específico de sensor ultrasónico: el HC-SR04 [23], ampliamente utilizado en aplicaciones de bajo costo y en proyectos educativos. Este sensor tiene un consumo de corriente de hasta 15 mA y puede medir distancias de entre 2 cm y 400 cm con una resolución de aproximadamente 0,3 cm. Su funcionamiento se basa en la emisión de pulsos ultrasónicos de 40 KHz desde su transmisor, los cuales rebotan en la superficie del objeto a medir, y son captados nuevamente por su receptor. La diferencia de tiempo entre la emisión y recepción del pulso es lo que permite calcular la distancia mediante una simple fórmula matemática:

$$Distancia = \frac{Tiempo\ de\ ida\ y\ vuelta * Velocidad\ del\ sonido\ en\ el\ aire}{2}$$

La división por dos es necesaria porque el tiempo medido corresponde al viaje de ida y vuelta del pulso.

2.4.2 Sensores láser

Los **sensores láser** [24] son dispositivos que permiten realizar mediciones precisas sin necesidad de contacto físico con el objeto. Estos equipos utilizan un láser, un sistema detector y un circuito de procesamiento para medir parámetros como la posición, el movimiento, el grosor o el diámetro de distintos elementos. Gracias a la alta linealidad del haz láser, estos sensores son capaces de registrar cambios mínimos con gran exactitud.

Su funcionamiento se basa principalmente en dos métodos: la **triangulación láser** y el **tiempo de vuelo (Time-of-Flight, TOF)**. La triangulación se emplea para distancias cortas y alta resolución. En este método, un láser proyecta un haz sobre la superficie del objeto, y el reflejo de esa luz incide sobre un detector, como un PSD o un CCD. A medida que varía la distancia del objeto, cambia el ángulo del haz reflejado, permitiendo calcular con precisión micrométrica la posición del mismo.

Por otro lado, el método de tiempo de vuelo es más adecuado para distancias largas. Consiste en medir el tiempo que tarda un pulso láser en ir hasta el objeto y volver al sensor. Conociendo la velocidad de la luz y usando sistemas de cronometraje de alta precisión, se determina la distancia recorrida. Aunque este método cubre mayores rangos, suele tener una resolución más baja en comparación con la triangulación.

Estos sensores son ampliamente utilizados en procesos industriales, control de calidad, robótica y otras aplicaciones donde se requiere medir con exactitud sin contacto directo.

La tecnología LiDAR (Light Detection and Ranging) permite determinar la distancia desde un emisor láser hasta un objeto o superficie utilizando un haz láser pulsado. Esta tecnología emplea miles de pulsos láser por segundo, para medir la disposición y la estructura del entorno [25]. Luego, para efectuar la medición, LiDAR utiliza el tiempo de vuelo (TOF), donde la distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada. En general, la tecnología LiDAR tiene aplicaciones en geología, sismología y física de la atmósfera. También se investiga su uso en vehículos, especialmente los autónomos [26].

A pesar de ser generalmente más costosos que los sensores ultrasónicos, los sensores láser ofrecen una precisión y versatilidad que justifican ampliamente su uso en diversas aplicaciones.

Un ejemplo de sensor laser con protección para exteriores es el LiDAR TF Mini Plus [27] de la Figura 11. El cual es un sensor que destaca por su capacidad de medir distancias desde 0.1 metros a 12 metros, lo que lo hace ideal para aplicaciones de corta y media distancia. Este sensor ofrece una exactitud de ± 5 mm, permitiendo obtener medidas confiables en diversas condiciones. Su frecuencia de muestreo permite realizar hasta 1000 lecturas por segundo, lo que facilita la captura de datos en tiempo real.

Figura 11: Sensor LiDAR TF Mini Plus. (Fuente: ar.mouser.com)

En términos de integración, el TF Mini Plus proporciona datos a través de salida serial UART o I2C, lo que simplifica su conexión con microcontroladores y otros sistemas de control. Funciona con una alimentación de 5 V, lo que lo hace compatible con la mayoría de las plataformas de desarrollo. Además, está diseñado para operar en un rango de temperatura de -20 °C a +60 °C, lo que lo hace adecuado para entornos exteriores. El LiDAR TF Mini Plus tiene un consumo de corriente promedio de 180 mA con picos de 500 mA.

En cuanto a su diseño, el sensor es compacto y ligero, con dimensiones reducidas, lo que facilita su integración en proyectos donde el espacio es limitado. También cuenta con una clasificación de protección IP65, lo que indica que es resistente al polvo y a salpicaduras de agua.

2.4.3 Sensores de presión hidrostática

La medición de nivel hidrostática [29] es una técnica utilizada para determinar el nivel de líquidos en tanques y otros recipientes.

Este método se basa en el principio fundamental de la **presión hidrostática**, que establece que la presión (P) ejercida por una columna de fluido es directamente proporcional a su altura (h), densidad (ρ) y por la fuerza de gravedad (g). De esta manera. la ecuación que la rige está dada por:

$$P = \rho * g * h$$

Por lo tanto, al medir la presión en el fondo de un tanque, se puede deducir el nivel del líquido, siempre que su densidad sea conocida y se mantenga constante.

Esta medición de presión se realiza típicamente mediante dispositivos especializados conocidos como transmisores de nivel hidrostáticos, como el transmisor de presión diferencial de la Figura 12.

Figura 12: Transmisor de presión diferencial. (Fuente: es.endress.com)

Se compensan factores como la presión atmosférica (en tanques abiertos) o la presión interna de tanques sellados (con medición diferencial).

Existen varios tipos de transmisores de nivel hidrostáticos, cada uno orientado a requisitos específicos de aplicación:

- **Transmisores de nivel sumergibles:**

Están diseñados para sumergirse completamente en el líquido. Se suspenden dentro del tanque y miden directamente la presión a la profundidad donde se encuentran, lo que los hace adecuados para tanques abiertos y cuerpos de agua. A menudo incorporan funciones como protección contra rayos para una mayor fiabilidad en entornos exigentes.

- **Transmisores montados externamente de presión relativa:**

Se montan directamente en el lateral o en la parte inferior del tanque. Se utilizan normalmente en tanques abiertos donde es posible el montaje directo.

- **Transmisores de presión diferencial:**

Se emplean para determinar el nivel en tanques cerrados y presurizados. Estos transmisores miden la diferencia de presión entre dos puntos. Un puerto se conecta al fondo del tanque, donde detecta la presión combinada del líquido y del gas que hay encima. El otro puerto se conecta a la parte superior del tanque, detectando solo la presión del gas. Al restar la presión del gas, el transmisor aísla la contribución de presión de la columna de líquido, permitiendo una medición precisa del nivel.

Las ventajas de utilizar estos instrumentos de medición, radican en su fiabilidad y estabilidad, ya que se trata de una tecnología madura y probada, con un rendimiento constante incluso en condiciones exigentes. Presenta una gran robustez, ya que no se ve afectada por la presencia de espuma, polvo o la viscosidad del líquido, lo que la hace adecuada para medios turbulentos o contaminados. Además, se destaca por su simplicidad tanto en la instalación como en la operación. Sin embargo, el uso de estos sensores también presenta algunas desventajas. No es apta para la medición de sólidos ni de líquidos muy viscosos que no fluyen libremente. Su precisión depende de que la densidad del líquido se mantenga constante, por lo que variaciones en este parámetro pueden generar errores en la medición. Asimismo, los sensores pueden ensuciarse o acumular residuos en entornos agresivos, lo que puede requerir mantenimiento periódico para asegurar su buen funcionamiento. Suelen ser bastante costosos.

2.4.4 Sensores de nivel de punto

Los interruptores de flotador [30], en una forma mecánica simple, han sido utilizados para el control de los flujos de agua en molinos y campos durante siglos y todavía hoy representan la tecnología más utilizada. Un cuerpo hueco (flotador), debido a su baja densidad y flotabilidad, se eleva o cae con el nivel ascendente y, respectivamente, descendente del líquido.

Los interruptores flotantes modernos, se utilizan para conmutar un circuito eléctrico. En su forma más simple, un interruptor de flotador consiste en un cuerpo flotante hueco con un imán incorporado, un tubo guía para guiar el flotador, unos collares o topes para limitar el recorrido del flotador en el tubo y un contacto de Reed ubicado en su interior. Los contactos reed están

formados por 2 láminas de material ferromagnético, herméticamente selladas dentro de una cápsula de vidrio. Dichas láminas están ligeramente separadas entre sí y se ponen en contacto en presencia de un campo magnético.

En el caso de un contacto reed normalmente abierto, al aplicar un campo magnético, las láminas entran en contacto. Cuando se produce el contacto entre las hojas, puede fluir una corriente a través de las hojas cerradas y se detectará una señal de conmutación. En el caso de un contacto normalmente cerrado, el contacto o circuito se interrumpe al aplicar un campo magnético. En la Figura 13 se puede apreciar una presentación de este tipo de sensores.

Figura 13: Sensor de nivel flotador. (Fuente: bloginstrumentacion.com)

2.5 Comunicaciones y transmisión de datos en sistemas IoT

En el ámbito del Internet de las Cosas (IoT), la comunicación eficiente y confiable entre los dispositivos es clave, especialmente en áreas remotas como Aluminé. Existen diversas tecnologías de comunicación inalámbrica [31], que se pueden utilizar para transmitir datos entre sensores, actuadores y plataformas IoT. Algunas de las tecnologías más relevantes incluyen WiFi, GSM, LoRa, y NB-IoT como las observadas en la Figura 14. Cada una de ellas tiene características particulares que las hacen más o menos adecuadas dependiendo de factores como la distancia, el consumo de energía, la accesibilidad y los costos.

NFC (Near Field Communication) es una tecnología de comunicación inalámbrica de corto alcance que permite la transferencia de datos entre dos dispositivos cuando se encuentran a pocos centímetros de distancia. Aunque su principal uso es en pagos móviles y la transferencia

de datos pequeños, su bajo consumo energético y simplicidad lo hacen útil para aplicaciones donde la seguridad y la inmediatez son cruciales. Sin embargo, su alcance extremadamente limitado (solo unos pocos centímetros) lo hace poco adecuado para aplicaciones que requieren una cobertura más amplia o transmisiones a mayor distancia. Durante la lectura o escritura NFC consume aproximadamente 40 mA, aunque solo por fracciones de segundo; en modo reposo (idle) el consumo baja a entre 3 mA y 5 mA, y en modo ultra ahorro puede llegar a menos de 5 µA [32].

Bluetooth es una tecnología de comunicación inalámbrica ampliamente utilizada en dispositivos personales, como teléfonos móviles, auriculares y dispositivos portátiles. Bluetooth opera en un rango de aproximadamente 10 a 100 metros, dependiendo de la clase del dispositivo. Su consumo energético es bajo, particularmente con versiones más recientes como Bluetooth Low Energy (BLE), lo que lo hace adecuado para aplicaciones donde el ahorro de energía es importante, como dispositivos portátiles y sensores. Sin embargo, aunque Bluetooth es muy eficiente para transmitir datos a corta distancia, no es ideal para aplicaciones en áreas remotas o de largo alcance. Una ventaja significativa es su masiva adopción en el mercado, lo que lo convierte en una opción accesible y económica para la mayoría de los usuarios. El consumo energético de la tecnología BLE varia acorde a los particulares del diseño del módulo, y el chip al cual se conecta, también depende de distancia a la fuente de conexión. En términos generales, el consumo en estado de reposo varia desde 1 µA a 10 µA, mientras que, durante la transmisión de datos, el consumo puede llegar hasta los 30 mA o 40 mA [33, 34].

WiFi es una de las tecnologías más extendidas debido a su alta velocidad de transmisión de datos, los bajos costos para la adquisición de hardware y la infraestructura ya existente en muchos lugares. Sin embargo, su alcance es limitado, generalmente unos pocos cientos de metros, lo que lo hace más adecuado para áreas urbanas o entornos cerrados. Además, su consumo energético es moderado, lo que puede ser un inconveniente para dispositivos IoT que dependen de baterías en áreas remotas. Con respecto al consumo energético, este depende de varios factores, incluyendo el modo de operación, la tasa de transferencia de datos, la distancia hasta el router o punto de acceso, y el tipo de tecnología utilizada (por ejemplo, 802.11 b/g/n/ac). En términos generales, cuando el dispositivo se encuentra en estado de operación activa, enviando o recibiendo datos, el consumo energético alcanza su valor máximo y puede variar entre 100 mA y 300 mA. En modo de reposo, en cambio, el consumo se reduce a un promedio de entre 10 mA y 50 mA. En el caso del ESP32, el consumo energético en modo activo, cuando transmite o recibe datos, varía entre 95 mA y 240 mA. [35, 36].

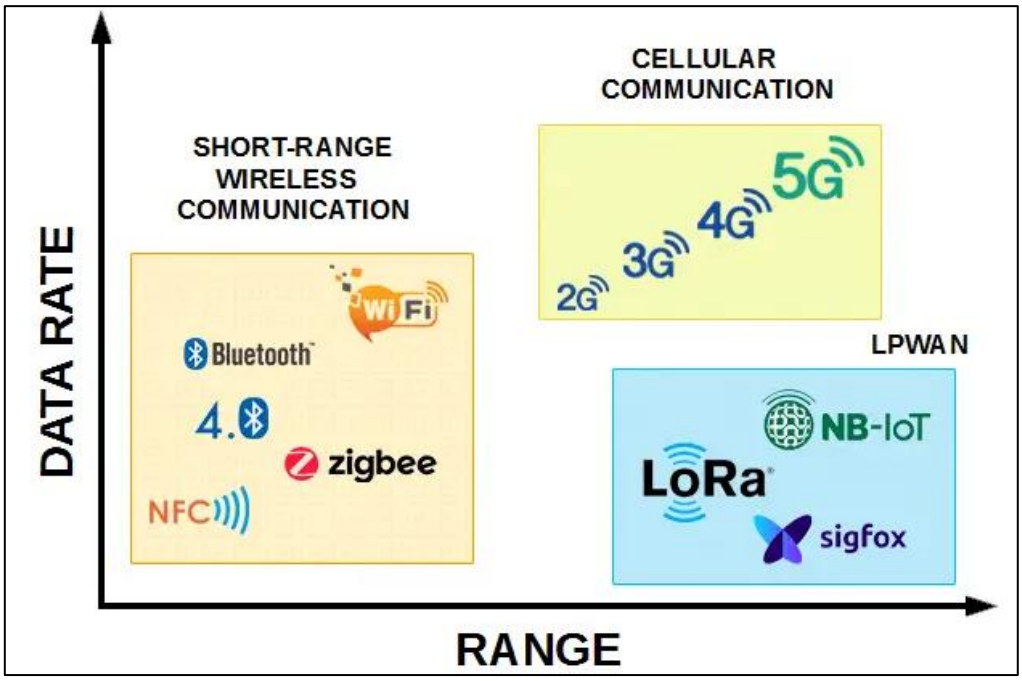


Figura 14: Relación distancia y tasa de datos en tecnologías de comunicación. (Fuente: mokolora.com)

GSM (y sus derivados como 3G, 4G y 5G) es ampliamente utilizado en dispositivos móviles y tiene una excelente cobertura en áreas rurales y remotas. Sin embargo, el consumo de energía es considerablemente alto, lo que puede limitar fuertemente la duración de las baterías en dispositivos IoT. Además, para el envío de grandes volúmenes de datos, se requiere una buena conectividad y acceso a tecnologías rápidas como 3G o superiores. En estos casos, también pueden representar una limitación los topes de datos móviles incluidos en los planes de telefonía. La infraestructura típica disponible hoy en día es 4G-LTE. En algunos lugares también puede encontrarse LTE-M, una tecnología basada en la infraestructura LTE, pero optimizada para un menor consumo energético, lo que la hace ideal para proyectos de IoT. En términos generales, el consumo energético típico en reposo ronda los 1,5 mA en GSM y puede llegar a 38 mA en LTE sin modos avanzados de ahorro. Durante la transmisión o recepción de datos, el consumo varía desde unos 110 mA en LTE-M hasta picos de 2 A en LTE convencional [37].

LoRa (Long Range) y LoRaWAN son tecnologías de comunicación inalámbrica de baja potencia y largo alcance, ideales para aplicaciones en áreas rurales o remotas. LoRa es particularmente atractivo porque ofrece un alcance de varios kilómetros con un consumo energético muy bajo, lo que permite que los dispositivos operen durante años con una sola batería. Esta tecnología es especialmente útil cuando se requiere transmitir pequeñas cantidades de datos de forma periódica y no en tiempo real. La principal limitación de LoRa es que, debido

a que el hardware disponible aún no está completamente masificado, los dispositivos propietarios pueden ser costosos. Sin embargo, existen opciones más económicas basadas en hardware abierto, como el uso de Raspberry Pi con concentradores LoRa, lo que permite su adopción incluso en proyectos con presupuestos limitados. El consumo energético aproximado de un módulo LoRa ronda entre los 5 mA y los 15 mA para la recepción y 120 mA para la transmisión [37].

Finalmente, **NB-IoT (Narrowband IoT)** es otra tecnología diseñada específicamente para aplicaciones IoT. Ofrece una buena cobertura en interiores y áreas rurales, así como un bajo consumo de energía. Sin embargo, al igual que GSM, requiere el uso de redes celulares, lo que puede incrementar los costos. Comparado con LoRa, NB-IoT es más adecuado para aplicaciones que requieren una mayor cantidad de datos o que operan en redes celulares existentes. El consumo energético es similar al de LTE-M, considerablemente inferior al de las redes celulares tradicionales, en torno a los 110 mA durante la recepción o transmisión de datos. [37].

La Tabla 1, ilustra algunos datos comparativos entre diversas tecnologías de comunicación inalámbrica utilizadas para proyectos IoT.

Wireless Standard	Power	Transmission Range (typical)	Data Rates
Bluetooth	Medium	1 to 100 m	1 to 3 Mbps
Bluetooth LE	Lower	>100 m	125 kbps to 2 Mbps
LoRaWAN	Low	10 km	0.3 to 50 kbps
NB-IoT	Low	<35 km	20 kbps to 5 Mbps
NFC	Low	<10 cm	106 to 424 kbps
Sigfox	Low	3 to 50 km	100 to 600 bps
6LoWPAN	Low	100 m	0 to 250 kbps
802.11/Wi-Fi	Medium	100 m to several km (with boosters)	10 to 100+ Mbps
802.15.4/Zigbee	Low	10 to 100 m	20 to 250 kbps
Z-Wave	Low	15 to 150 m	9.6 to 40 kbps

Tabla 1: Comparación entre distintas tecnologías de comunicación inalámbrica. (Fuente: hackaday.com)

3 Metodología y Desarrollo

3.1 Locaciones y diseño

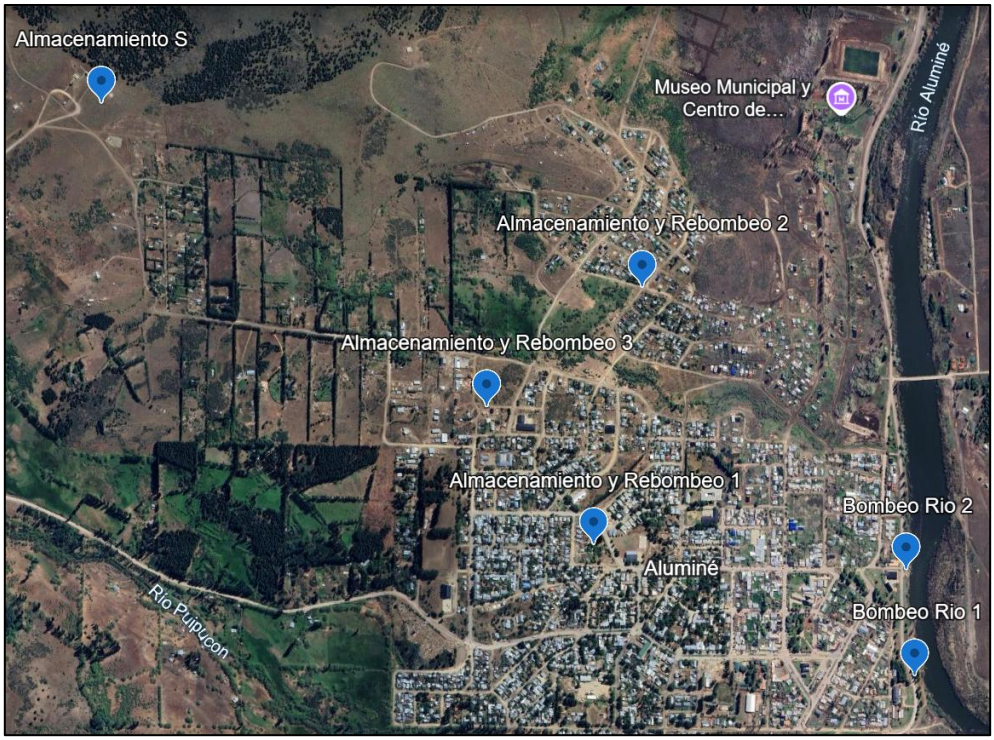


Figura 15: Ubicación aproximada de tanques y/o bombas en Aluminé. (Fuente: Google Earth)

El proyecto tiene como objetivo la automatización de la carga de tanques de agua en Aluminé, mediante la instalación del hardware necesario en cada una de las seis zonas señaladas en la Figura 15. En estas áreas, se presentan diferentes configuraciones: algunas cuentan únicamente con bombas, otras solo con tanques, y hay zonas que combinan ambos. Las áreas de Bombeo Río 1 y 2 son donde se capta y bombea el agua del río Aluminé hacia los espacios de almacenamiento y rebombeo 1, 2 y 3. En algunas de estas zonas, se encuentran plantas potabilizadoras y se almacena el agua para el posterior rebombeo. Por último, existe un nivel apartado dedicado exclusivamente al almacenamiento, donde se sitúa un tanque de grandes dimensiones.

Para desarrollar el trabajo, se consideraron varias alternativas en cuanto al hardware necesario para el abastecimiento automatizado de agua en los tanques de Aluminé. Un proyecto de esta índole podría implementarse mediante PLC en cada zona de almacenamiento y rebombeo de agua, con conectividad a internet, y mediante un sistema SCADA, se podría visualizar el funcionamiento completo del sistema en tiempo real. Sin embargo, la principal dificultad de la implementación de esta solución en Aluminé radica en los aspectos económicos.

El pueblo no puede costear un software de este tipo ni la considerable suma de dinero que demandan los PLC con una cantidad "mediana" de entradas y salidas. Por ello, esta solución de automatización fue descartada.

Otra alternativa, considerablemente más económica, era la utilización de hardware con un módulo de comunicación inalámbrica LoRa. Sin embargo, el inconveniente de esta opción radicaba en asumir el costo de un gateway, ya fuera propietario o de tipo open-hardware (más económico), lo cual llevó a descartar esta alternativa. A pesar de esto, un proyecto de esta índole podría llevarse a cabo utilizando esta tecnología de comunicación inalámbrica mediante la adquisición de un gateway open-hardware, que resulta más accesible en términos económicos.

Por lo tanto, se decidió realizar el proyecto con hardware ampliamente desarrollado y accesible en el mercado, utilizando placas de desarrollo comerciales con módulos de tecnología WiFi y routers en cada zona donde habría tanques y/o bombas. En principio, en Aluminé existen proveedores de internet, pero los costos pueden ser elevados o no tener cobertura para la totalidad de las zonas, en esta situación es posible que se desee adquirir modem/router con chips 4G para permitir la comunicación con Internet, dado que desde 2022 Movistar ya provee esta conexión en Aluminé.

En cuanto al hardware utilizado, se optó por emplear placas de desarrollo comerciales NodeMCU ESP32 debido a su generosa cantidad de entradas y salidas, la posibilidad de conexión WiFi gracias al SoC que presentan, la simplicidad de programación mediante el IDE de Arduino y el acceso a numerosas bibliotecas desarrolladas por la comunidad, entre otras razones.

Respecto a los sensores, se consideraron dos tipos de mediciones. En primer lugar, se pensó en una medición de nivel de punto mediante sensores flotadores, que se ubicarían en un nivel inferior y superior en cada tanque. A partir del accionamiento de estos, se generaría la activación o el apagado de las bombas, ya sea localmente o de forma remota (a través de la plataforma IoT), según corresponda. Por otro lado, resulta interesante conocer con mayor exactitud el nivel de agua en tiempo real dentro de cada tanque. Para ello, se podrían emplear sensores laser como el LiDAR TF Mini Plus o semejantes (la prueba se llevó a cabo con sensores ultrasónicos HC-SR04, por su accesibilidad y bajo costo), a través de los cuales sería posible medir el porcentaje de llenado de cada tanque en función de sus dimensiones y la ubicación de los sensores flotadores.

Para efectuar las pruebas, en primera instancia, luego de programar el ESP32 a través de Arduino IDE, se utilizó una protoboard con los sensores señalados. Luego, se soldó en una placa PCB experimental pines hembra Dupont para la colocación y extracción del ESP32 con simplicidad y también se soldó borneras para vincular los pines del mismo con los sensores, en un diseño que podría ser el ubicado en cada una de las seis localidades de Aluminé. En esta configuración, no todos los pines del dispositivo están vinculados mediante soldadura; solo algunos más que la cantidad utilizada en las pruebas.

La placa experimental se colocó en una caja estanca para exteriores, la cual se modificó para incluir una pantalla LCD que permita la visualización local de la información del ESP32, junto con un pulsador que habilite el cambio de menús/pantallas. La caja tiene perforaciones en un lado para el ingreso y egreso de cables a los sensores de cada tanque. Además, presenta el ingreso de energía DC de entre 5 V y 12 V mediante un módulo con entrada redonda hembra DC.

Con respecto al accionamiento de las bombas mediante los ESP32 en Aluminé, se utilizaría un esquema de control similar al de la Figura 16. A través de la salida correspondiente del ESP32 (-S), se accionaría un relé (-K) que cerrará el circuito para alimentar la bobina del contactor. Una llave conmutadora (-S3,-S4) permitirá habilitar el funcionamiento automatizado (mediante el ESP32) o el manual, que es el modo en que funciona actualmente. Esta implementación ofrecerá la posibilidad de cambiar a la función manual en caso de que surja algún desperfecto o problema con el automatismo.

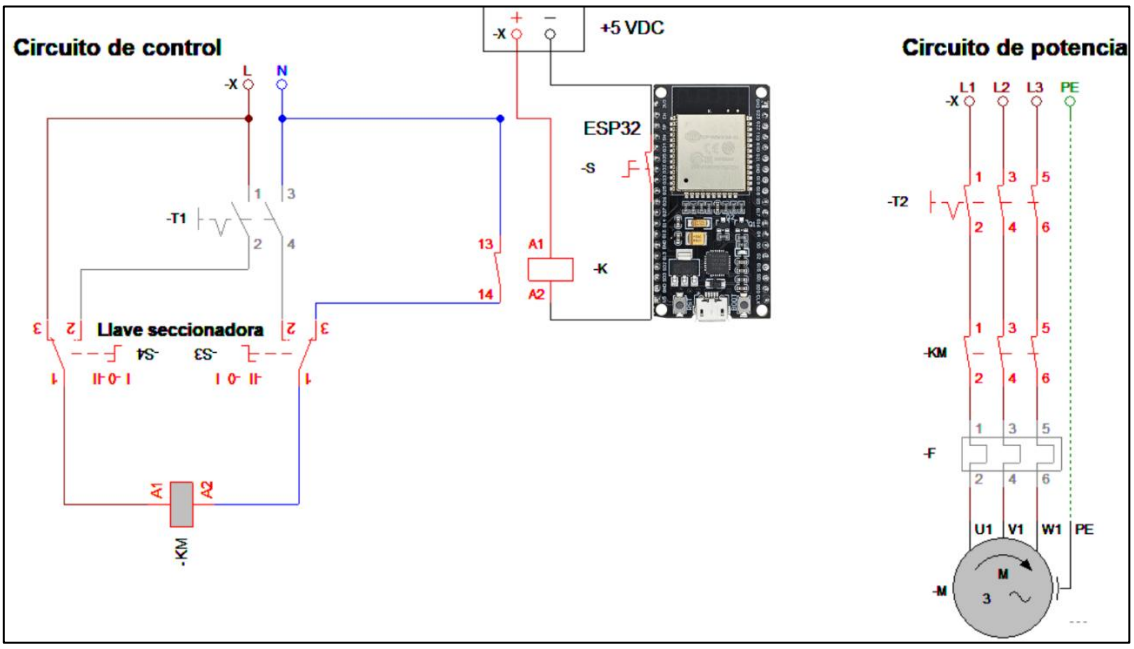


Figura 16: Esquema eléctrico para el control de las bombas en Aluminé.

3.2 Desarrollo

En este subcapítulo se detallará el trabajo realizado en base a los lineamientos, el diseño, el hardware y software presentado en el apartado anterior.

3.2.1 Programación del automatismo

En este apartado se describirá el código desarrollado en el ESP32 para la automatización de bombas en Aluminé. El código que se explicara fue creado considerando las mayor flexibilidad y escalabilidad posible, esto es, es un programa que con pocas líneas de código se lo puede escalar en caso de añadir más tanques y/o bombas en cada zona. También, mediante una ligera modificación, será posible adaptarlo a cada zona, es decir, si el ESP32 necesita cubrir más o menos tanques y/o más o menos bombas. Además, el código está debidamente comentado en su total extensión para generar mayor comprensión al usuario que lo edite.

El ESP32 es un SoC desarrollado por la empresa Espressif Systems y tienen un entorno de desarrollo específico para sus productos. Sin embargo, como fue mencionado, es posible programarlo en Arduino IDE mediante bibliotecas. En específico, Espressif Systems desarrollo la biblioteca necesaria para que se pueda utilizar en Arduino IDE y desde allí programarlo. Esto es una ventaja considerable, debido a que Arduino IDE tiene una interfaz amigable y sencilla, además de que permite la integración con otros sensores y dispositivos mediante bibliotecas desarrolladas por los fabricantes o por la comunidad, lo que simplifica considerablemente los códigos a desarrollar.

Sin embargo, una de las principales desventajas del Arduino IDE, particularmente en proyectos de mediana o gran envergadura, es que las herramientas de depuración (debugging) son útiles solo en placas que utilizan microcontroladores SAMD. Las versiones más nuevas del IDE incorporan un sistema de depuración gráfico con soporte para breakpoints, visualización de variables y ejecución paso a paso, pero este soporte no está disponible para todas las placas. Algunas placas específicas, como la Arduino Zero, permiten utilizar el depurador integrado sin hardware adicional; sin embargo, en otras placas basadas en SAMD se requiere hardware adicional como sondas JTAG, J-Link o Atmel-ICE.

La placa utilizada en este proyecto, NodeMCU ESP32 DevKit, posee soporte para depuración por hardware, pero necesita una interfaz JTAG externa y el uso de entornos como PlatformIO con VS Code, o directamente el ESP-IDF (entorno oficial de desarrollo provisto por Espressif Systems) para realizar una depuración más avanzada. En caso de continuar

utilizando el Arduino IDE con placas sin soporte nativo de depuración, esta se limita a métodos tradicionales como el uso del monitor serial y sentencias Serial.print(), lo cual puede dificultar el diagnóstico de errores complejos.

Otro punto a considerar es que el Arduino IDE está diseñado para simplificar el desarrollo, por lo tanto, oculta muchas complejidades del hardware. Esto lo vuelve ideal para proyectos simples, pero limita el acceso a configuraciones avanzadas del sistema, como el manejo detallado de tareas en FreeRTOS (sistema operativo del ESP32), la gestión fina del consumo energético, configuraciones personalizadas del WiFi, Bluetooth, uso de periféricos avanzados o el aprovechamiento completo de los dos núcleos del ESP32. No obstante, a pesar de estas limitaciones, el código desarrollado y el grado de control alcanzado sobre el dispositivo resultaron suficientes para satisfacer los requerimientos funcionales del proyecto.

- **Bibliotecas y definiciones:**

La primera parte del código incluye los archivos headers <.h> de las bibliotecas necesarias para el funcionamiento de la pantalla LCD y del módulo WiFi del ESP32. Luego, las siguientes tres bibliotecas están relacionadas con la vinculación del ESP32 con ThingsBoard, la plataforma IoT utilizada en el proyecto. <ThingsBoard.h> es el header de la biblioteca principal de ThingsBoard, mientras que las bibliotecas siguientes también están vinculadas, ya que habilitan la comunicación con la plataforma mediante el protocolo MQTT y permiten el manejo de archivos JSON para el envío de datos en ese formato, especialmente en los casos de llamadas a procedimientos remotos, o RPC (Remote Procedure Calls).

"LiquidCrystal_I2C lcd(0x27, 16, 2)" define la dirección de comunicación con la pantalla LCD y sus dimensiones (16 caracteres x 2 filas).

Luego, se definen parámetros importantes para conectar el ESP32 con ThingsBoard. Primero, "TB_Server" es la dirección del servidor a la cual se conecta el dispositivo. Debido a que este proyecto utiliza la opción gratuita de la plataforma, se empleó la versión demo. En condiciones normales, lo más conveniente sería elegir el plan más económico de ThingsBoard en la nube (que cubriría adecuadamente las necesidades del proyecto), en cuyo caso la dirección cambiaria ligeramente a la expresada en la Figura 17. Por otro lado, "TOKEN" es un código alfanumérico único que ThingsBoard asigna al añadir un nuevo dispositivo desde la plataforma. Este TOKEN debe copiarse desde la plataforma y colocarse como credencial de autenticación en la programación del dispositivo. De esta manera, cada dispositivo conectado necesita su

propio token para autenticarse, por lo que, si hipotéticamente existieran diez ESP32 instalados en Aluminé, se generarían diez tokens distintos, uno por cada dispositivo.

```
1  /*--------------------Bibliotecas y definiciones----------------------*/
2  #include <LiquidCrystal_I2C.h>      //Habilita bibliotecas para la pantalla
   LCD
3  #include <WiFi.h>                   //Habilita bibliotecas WiFi
4  #include <ThingsBoard.h>            //Habilita bibliotecas para comunicacion
   con Things Board IoT Platform
5  #include <Arduino_MQTT_Client.h>    //Habilita el uso del protocolo MQTT
6  #include <ArduinoJSON.h>            //Habilita el uso de bibliotecas para el
   tratamiento de archivos JSON
7  LiquidCrystal_I2C lcd(0x27, 16, 2); // Configura direccion del LCD a 0x27
   para  16 caracteres y 2 lineas
8
9  #define TB_SERVER "demo.thingsboard.io"
10 #define TOKEN "i261nbmwkv4gyypjji5i"
11
12 constexpr uint16_t MAX_MESSAGE_SIZE = 256U;    //Constante unsigned int de
   16 bits para almacenar la longitud maxima de los mensajes por MQTT
13 WiFiClient espClient;                          //Crea el cliente Wifi
14 Arduino_MQTT_Client mqttClient(espClient);     //Conecta el cliente WiFi al
   cliente MQTT
15 ThingsBoard tb(mqttClient, MAX_MESSAGE_SIZE);  // Inicializa la instancia
   ThingsBoard con el tamaño de los mensajes
16
17 hw_timer_t *timer = NULL;
18 portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
```

Figura 17: Bibliotecas y definiciones iniciales del código del automatismo.

“Max_Message_Size” define el tamaño máximo de los mensajes enviados a la plataforma, mientras que “WiFiClient” crea el cliente WiFi, “Arduino_MQTT_Client” conecta dicho cliente WiFi al cliente MQTT, y “Thingsboard tb()” inicializa una instancia en ThingsBoard con este cliente MQTT y el tamaño de mensaje establecido, completando así los parámetros de conexión a la plataforma.

Finalmente, “hw_timer_t” determina la creación de un tipo de variable para configurar un temporizador de nombre “timer”. El mismo se utilizará para enviar información periódica a la plataforma y también para llevar el conteo de tiempo necesario para habilitar nuevamente los intentos de reconexión al WiFi tras haber alcanzado el número máximo de intentos fallidos. Luego, se añade la variable “portMUX_TYPE” que se utiliza para gestionar la sincronización entre el bucle principal del programa y el ISR (rutina de interrupciones).

Por último, cabe mencionar que existe otro método de creación de dispositivos en la plataforma. ThingsBoard permite adicionar dispositivos mediante una funcionalidad llamada "provisionamiento de dispositivos" (Device Provisioning). Esta modalidad consiste en aprovisionar dispositivos a partir de la definición de un nombre de dispositivo, junto con una clave de provisión y una clave secreta, las cuales se generan al habilitar el provisionamiento en el perfil de dispositivo (device profile) en la plataforma. Estos tres valores se deben incluir en el código del automatismo, y mediante una comunicación inicial con la plataforma, ThingsBoard valida la solicitud. Si el nombre no está duplicado y las claves son correctas, la plataforma responde con un objeto que indica el estado de la solicitud "provisionDeviceStatus", el tipo de credencial generada "credentialsType" y la credencial misma, que en este caso es un "accessToken". Este TOKEN puede luego almacenarse y utilizarse como credencial de autenticación para establecer y mantener la conexión del dispositivo con la plataforma, permitiendo continuar con la ejecución normal del automatismo. Esta modalidad resulta útil para registrar múltiples dispositivos sin necesidad de crear uno por uno desde la interfaz web, facilitando así el despliegue a gran escala.

- **Parámetros configurables:**

A continuación, en el código se presentan los parámetros que deben modificarse para cada ESP32 en particular, según la zona en la que se encuentre y la cantidad de bombas y/o tanques que tenga vinculados en ese momento.

Dado que en cada zona existirá una red local, será necesario especificar los parámetros de conexión SSID y contraseña para permitir que el ESP32 se conecte a Internet. La constante, "tiempoEnvioDatos" es el intervalo en segundos para enviar datos a la plataforma IoT. Este valor depende en gran medida de la velocidad de variación en el nivel de agua de los tanques de la zona, y puede ajustarse, por ejemplo, a un minuto o a cinco minutos si el tanque es grande.

Los parámetros "tiempoCheckeoWifi" y "maxIntentos" están relacionados con la conexión WiFi. El primero, define el intervalo en segundos para verificar la conexión; al finalizar el tiempo especificado, el dispositivo reiniciará los intentos de reconexión, independientemente de si el ESP32 está conectado o no a la red WiFi. Por otro lado, "maxIntentos" indica la cantidad máxima de intentos permitidos para reconectar al WiFi.

Entre las líneas 27 y 30 se encuentran las especificaciones de los tanques y bombas vinculados al dispositivo y su numeración en la zona donde se ubica. Los macros definidos

como “numeromaxtanques” y “numeromaxbombas” indican la cantidad de bombas y tanques vinculados al ESP32; en caso de no haber tanques o bombas, se debe asignar el valor 0. Luego, estos macros serán asignados a constantes que se utilizarán en funciones del código.

```
21  /*------------------------PARAMETROS CONFIGURABLES-----------------------*/
22  const char ssid[] = "ProyectoTesis";        // SSID del WiFi de la
    zona
23  const char contrasenaWiFi[] = "Tesis4433"; // Contraseña de la red
    Wifi
24  const int tiempoEnvioDatos = 10;           // Tiempo de refresco en
    segundos para enviar la informacion a la plataforma IoT
25  const int tiempoCheckeoWifi = 90;          // Tiempo Wifi en segundos para
    revisar la conexion una vez caida
26  const int maxIntentos = 6;                 // Variable que almacena el
    numero maximo de intentos de reconexion a WiFi
27  #define   numeromaxtanques 2               // Define el numero de tanques
    vinculados al dispositivo
28  #define   numeromaxbombas 0                // Define el numero de bombas
    vinculadas al dispositivo
29  const int numeroTanqueInicial = 1;         // Tanque inicial vinculado al
    dispositivo en funcion de los tanques de la zona
30  const int numeroBombaInicial = 0;          // Bomba inicial vinculada al
    dispositivo en funcion de las bombas de la zona
31  const int cantidadBombasIndependientes = 0; // Cantidad de bombas
    independientes vincualadas, (son accionadas de manera remota)
32  const int tiempoMenuLCD = 7;               // Tiempo general en el menu del
    LCD en segundos (excepto el menu del estado WiFi)
33  const int tiempoMenuWifi = 4;             // Tiempo en el menu del LCD para
    la informacion del estado de WiFi e IoT en Segundos
34  /*-----------------------------------------------------------------------*/
```

Figura 18: Parámetros configurables para cada ESP32.

“numeroTanqueInicial” y “numeroBombaInicial” son constantes importantes que indican, en función de la cantidad de bombas y tanques en la zona, el número inicial de tanque y bomba vinculado al dispositivo en cuestión. Esto se ilustra mejor en la Figura 19, que detalla dos zonas hipotéticas con dos dispositivos en cada una. En la Zona 1, existen cinco tanques y dos bombas. Basándose en las entradas y salidas del ESP32, una distribución adecuada podría ser: tres tanques vinculados al dispositivo 1 (tanques con ID: 1, 2 y 3) y dos tanques (ID: 4 y 5) junto con dos bombas (ID: 1 y 2) vinculados al dispositivo 2. En este caso, para el dispositivo 1 numeroTanqueInicial = 1, mientras que para el dispositivo 2, numeroTanqueInicial = 4. “numeroBombaInicial” se asignaría de la misma manera; sin embargo, si un dispositivo no tiene bombas vinculadas (como en el caso hipotético de la Zona 1), este parámetro se volvería irrelevante. Esta metodología de asignación de ID simplifica el envío de datos al servidor de

ThingsBoard, ya que evita que lleguen dos o más mensajes con el mismo ID en una misma zona, evitando así la sobreescritura de información.

El último parámetro relacionado con las bombas es “cantidadBombasIndependientes”, que indica si existen bombas vinculadas al ESP32 que se activan de forma remota mediante comandos RPC a través de ThingsBoard.

Los dos últimos parámetros, “tiempoMenuLCD” y “tiempoMenuWifi”, determinan el tiempo en segundos para la visualización en la pantalla LCD de la información de las bombas y tanques (primer parámetro) y el estado de conexión a la red local mediante WiFi y al servidor de ThingsBoard (segundo parámetro).

Es importante tener en cuenta que las siguientes secciones del código están desarrolladas considerando los parámetros de la Figura 18. En este caso específico, el dispositivo programado posee dos tanques y ninguna bomba.

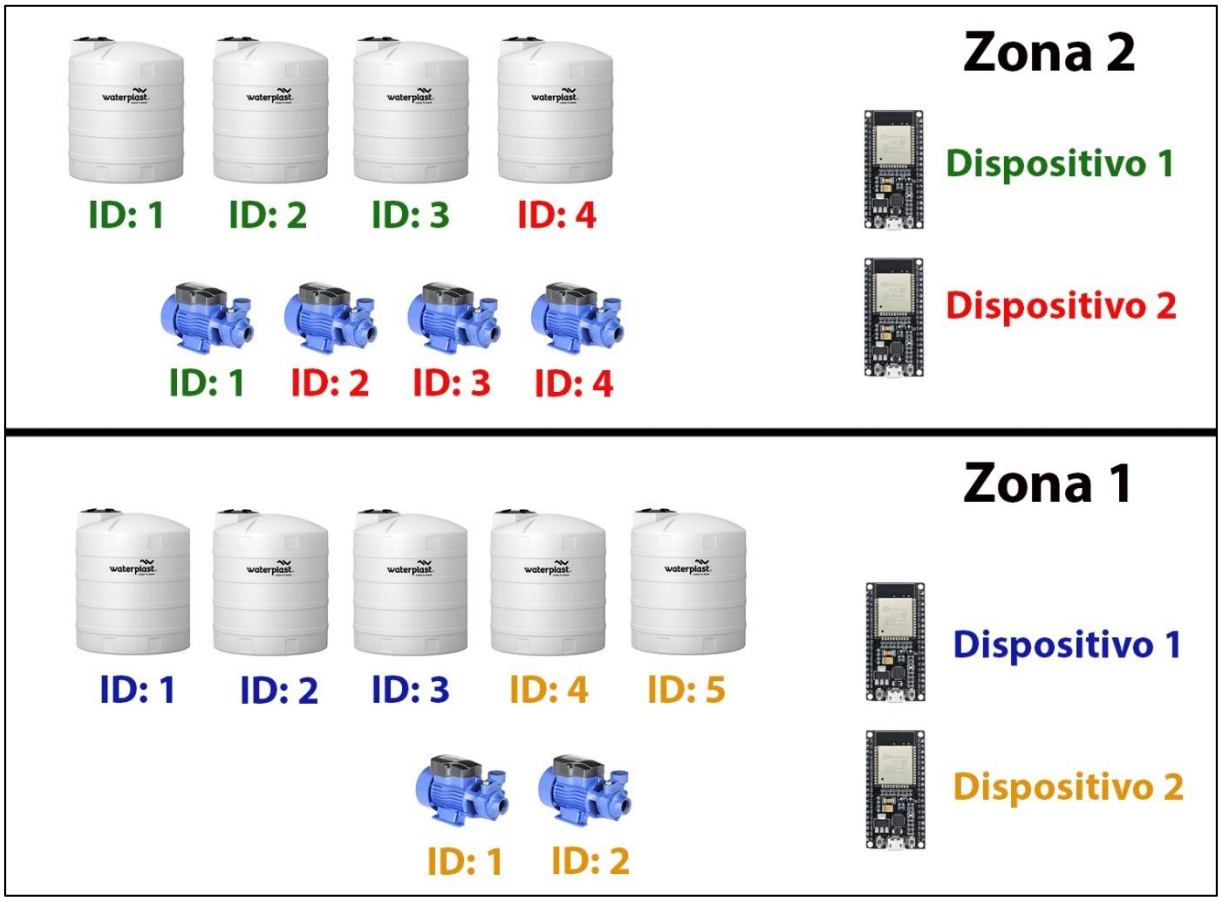


Figura 19: Identificación y distribución de bombas, tanques y dispositivos hipotética en dos zonas de Aluminé.

- **Variables configurables:**

```
36  /*----------------------INICIO DE VARIABLES CONFIGURABLES-----------------*/
37
38  /*
39  Como condicion de diseño si en la zona hay algun tanque dependiente de una
    bomba, se deberan colocar aquellos pines de los sensores de
40  esos tanques PRIMERO en "ASIGNACION DE PINES DE LOS SENSORES DE NIVEL PARA CADA
    TANQUE Y ARREGLOS DE IDENTIFICACION" y en
41  "ASIGNACION DE PINES PARA LAS BOMBAS Y ARREGLOS DE BANDERAS DE IDENTIFICACION DE
    OPERACION". De manera analoga en la seccion
42  "ASIGNACION DE PINES PARA LOS DETECTORES ULTRASONICOS DE NIVEL EN LOS TANQUES"
    se deberan colocar los pines correspondientes
43  a cada tanque en orden creciente
44  */
45
46  const int pinPulsadorExt = 25;  //Pin del pulsador externo
47
48  // Asignacion de pines de los sensores de nivel para cada tanque y arreglos de
    identificacion
49  const int pinSensorNivelAltoTanque1 = 16;  //Pin del sensor de nivel superior
50  const int pinSensorNivelBajoTanque1 = 17;  //Pin del sensor de nivel inferior
51  const int pinSensorNivelAltoTanque2 = 32;  //Pin del sensor de nivel superior
52  const int pinSensorNivelBajoTanque2 = 33;  //Pin del sensor de nivel inferior
53  int sensorNivelAlto[] = {pinSensorNivelAltoTanque1, pinSensorNivelAltoTanque2};
    //Se debe añadir en el arreglo separados por comas, si hay mas tanques.
54  int sensorNivelBajo[] = {pinSensorNivelBajoTanque1, pinSensorNivelBajoTanque2};
    //Se debe añadir en el arreglo separados por comas, si hay mas tanques.
55  /*
56  //Asignacion de pines para las bombas y arreglos de banderas de identificacion
    de operacion
57  const int pinRele1 = 4;        //Pin de salida hacia activacion de Bomba 1.
58  const int pinRele2 = 26;       //Pin de salida hacia activacion de Bomba 2.
59  int pinRele [] = {pinRele1, pinRele2};   //Se debe añadir en el arreglo
    separados por comas, si hay mas bombas.
60  */
```

Figura 20: Variables configurables para cada ESP32 (Parte 1).

Como fue mencionado, existen dos posibilidades respecto a la relación de las bombas vinculadas con el dispositivo. Un dispositivo puede poseer bombas tanto “dependientes” como “independientes”, es decir, bombas que se activen mediante tanques vinculados al mismo dispositivo (activación local) o bombas que se activen remotamente mediante comandos RPC por tanques desde otra zona (activación remota). Como condición de diseño, según lo expresado al comienzo de la Figura 20, se deberán colocar primero en los arreglos, aquellos ID’s de menor denominación, esto es, los tanques y bombas de activación local. De esta manera, considerando

un caso hipotético como el de la Figura 19, suponiendo que en la configuración de la Zona 2 el dispositivo 2 tenga dos bombas independientes y una local, la bomba con ID = 2 deberá ser la bomba local que cargue el tanque con ID = 4.

En la Figura 20 se definen variables que identifican los pines del ESP32. Primero se define el pin del pulsador, luego se especifican, en caso de que existan tanques, los pines para los sensores de nivel flotador (inferior y superior). Por último, se agrupan las variables asignadas a los pines en arreglos que serán utilizados en el programa.

```
61  // Asignacion de pines para los detectores ultrasonicos de nivel en los tanques
62  const int pinTrigTanque1 = 23;
63  const int pinEchoTanque1 = 5;
64  const int pinTrigTanque2 = 18;
65  const int pinEchoTanque2 = 19;
66  int pinTrigTanques[] = { pinTrigTanque1, pinTrigTanque2 };  // Se debe añadir en
    el arreglo separados por comas, si hay mas tanques.
67  int pinEchoTanques[] = { pinEchoTanque1, pinEchoTanque2 };  // Se debe añadir en
    el arreglo separados por comas, si hay mas tanques.
68
69  // Parametros dimensionales de los tanques y arreglos de identificacion
70  // TANQUE 1
71  const int alturaMaximaTanque1 = 500;       // Se debe colocar la altura en CM
72  const int distanciaSensorSuperiorDTechoTanque1 = 50;    // Distancia desde la
    tapa del tanque hasta el sensor de nivel superior en CM
73  const int distanciaSensorInferiorDPisoTanque1 = 40;      // Distancia desde el
    piso del tanque hasta el sensor de nivel inferior en CM
74  const int diametroTanque1 = 200;           // Se debe colocar el diametro en CM
75  // TANQUE 2
76  const int alturaMaximaTanque2 = 45;        // Se debe colocar la altura en CM
77  const int distanciaSensorSuperiorDTechoTanque2 = 10;  // Distancia desde la tapa
    del tanque hasta el sensor de nivel superior en CM
78  const int distanciaSensorInferiorDPisoTanque2 = 10;   // Distancia desde el piso
    del tanque hasta el sensor de nivel inferior en CM
79  const int diametroTanque2 = 30;            // Se debe colocar el diametro en CM
80  // Arreglos de identificacion (colocarlos en forma ascendente y ordenada)
81  const int alturaMaximaTanques[] = { alturaMaximaTanque1, alturaMaximaTanque2 };
82  const int distanciaSensorSuperiorDTecho[] = {
    distanciaSensorSuperiorDTechoTanque1, distanciaSensorSuperiorDTechoTanque2 };
83  const int distanciaSensorInferiorDPiso[] = {
    distanciaSensorInferiorDPisoTanque1, distanciaSensorInferiorDPisoTanque2 };
84  const int diametroTanques[] = { diametroTanque1, diametroTanque2 };
85
86  /*-----------------------FIN DE VARIABLES CONFIGURABLES----------------------*/
```

Figura 21: Variables configurables para cada ESP32 (Parte 2).

De manera análoga, en el caso de que existan bombas, se deben declarar e inicializar las variables con los pines que se utilizarán para el accionamiento de las mismas. Al final, se deben agrupar en arreglos.

En la Figura 21 se encuentran el final de las variables configurables. Entre las líneas 62 y 65 se definen los pines correspondientes a los sensores ultrasónicos. El sensor ultrasónico utilizado, HC-SR04 tiene dos pines (además de VCC y GND): uno es el pin trigger que se encarga de enviar el pulso ultrasónico y el otro es el pin echo, que se encarga de recibirlo. Estos pines se definen primero y luego se colocan las variables que los identifican dentro de los arreglos de las líneas 66 y 67.

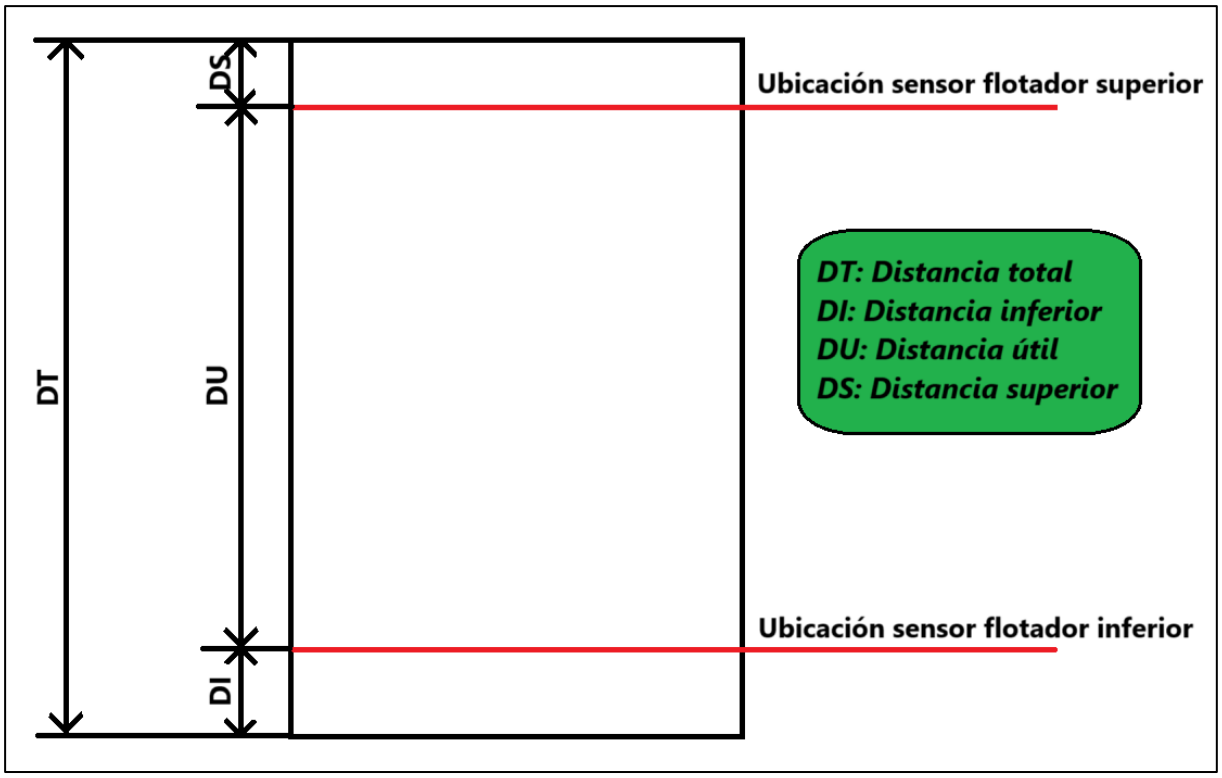


Figura 22: Consideraciones longitudinales para la colocación de los sensores en los tanques.

Por último, desde la línea 69 hasta la 86 se declaran e inicializan variables que almacenan las dimensiones físicas de cada tanque en centímetros, de manera que tanto el porcentaje de llenado como el volumen de agua de cada tanque se definan en términos de una cantidad de agua “útil”, como se indica en la Figura 22. Las dimensiones que se deben indicar incluyen la altura total, la distancia desde el techo del tanque hasta el sensor flotador superior, la distancia desde el piso del tanque hasta el sensor flotador inferior y el diámetro del tanque. Luego, al igual que en otros casos, estas variables se agrupan en arreglos que serán utilizados en el código.

Cabe mencionar que, en caso de que no existan tanques o bombas vinculadas al ESP32, se puede optar por comentar, eliminar o dejar sin modificar las "variables configurables" indicadas en las Figuras 20 y 21. Lo más habitual es que, si en determinada zona no hay tanques, ni siquiera se declaren las variables asociadas a los pines de los sensores flotadores, los sensores ultrasónicos o las dimensiones de los tanques. De manera análoga, si el dispositivo no tiene bombas vinculadas, tampoco se declararían las variables asociadas a los pines de salida destinados al control de los relés, que habilitan su accionamiento.

Esto es posible gracias al uso de directivas condicionales del tipo #if / #endif en el código, que permiten compilar (o no) determinadas secciones del programa en función del valor de ciertos macros, en este caso, "numeromaxtanques" y "numeromaxbombas". De este modo, si numeromaxtanques > 0, se habilitan los segmentos del código correspondientes a los tanques, y lo mismo ocurre con el macro "numeromaxbombas" para las bombas.

Por ello, en las siguientes secciones del código podrá observarse que algunas funciones están contenidas dentro de este tipo de condicionales.

- **Variables globales:**

La Figura 23 presenta variables globales utilizadas para almacenar información compartida entre varias funciones. Algunas de ellas, como "Tanque" y "Bomba", se emplean como punteros para indicar el número de tanque o bomba en la pantalla LCD o como variable de ingreso a varios arreglos y condicionales. Otra, como "intentos", se usa para contar la cantidad de intentos de reconexión a la red WiFi local.

También se incluyen banderas que registran el estado de ciertas condiciones del programa, como "pulsador", que habilita la visualización local de información; "banderaInterrupcion", que señala que se ha ingresado a la rutina de interrupciones mediante el pulsador; o "finBomba", que indica que el puntero ha llegado a la última bomba en el menú de visualización local.

Además, hay variables utilizadas como factores de conversión, y constantes como "numeroMaxTanques" y "numeroMaxBombas", que toman los valores definidos por los macros de los "parámetros configurables". Estas constantes se emplean tanto para enviarlas como atributos a la plataforma como para definir los límites de distintos bucles dentro del código.

```
89  /*-------------------------VARIABLES GLOBALES-----------------------*/
90  const int numeroMaxTanques = numeromaxtanques;     // Asignacion del
    Macro a la variable numeroMaxTanques
91  const int numeroMaxBombas = numeromaxbombas;       // Asignacion del
    Macro a la variable numeroMaxBombas
92  bool cargaBombas[numeroMaxBombas + 1] = { false }; // Bandera de
    encendido/apagado de las bombas.
93  const float conversionVelocidadSonido = 58.3;  // Factor que representa
    la velociadad del sonido en Cm/us, divida en la mitad e invertida (^-1)
94  const float volLitros = 0.001;                     // Factor de
    conversion para pasar de CM^3 a Litros
95  unsigned long tiempo = 0;                          // Variable para
    registrar los tiempos muertos de la pantalla LCD
96  unsigned long tiempoUltimoRebote;                  // Variable para
    registrar tiempo de rebote del pulsador externo en interrupcion
97  bool checkTimerDatosOk = false;                    // Variable que se
    utiliza para habilitar o denegar el envio de datos a la plataforma IoT
98  bool enviarUnaVez = false;                         // Variable para
    enviar atributos a la plataforma ThingsBoard una unica vez.
99  int estadoDeConexionIoT = 0;                // Variable que se utilizara
    para presentar en el LCD el estado de la conexion con el servidor IoT
100 volatile int checkReconexionWifi = 0;           // Variable que
    habilita el reset de variables que controlan la conexion WIFi.
101 int lectura = LOW;                              // Variable que recibe
    la lectura de la entrada del pulsador externo (para el debouncing)
102 int distanciaCm;                               // Distancia en centimetros
    desde el sensor ultrasonico hasta el nivel de agua del tanque
103 int longitudPorcentaje;  // Cantidad de caracteres del porcentaje de
    llenado
104 float porcentaje;        // Nivel de porcentaje de llenado del tanque
105 float volumen;           // Volumen de agua en el tanque
106 volatile int intentos = 0;  // Variable que almacena el numero de
    intento de conexion WiFi
107 volatile int pulsador = 0;  // Variable que representa el pulsado del
    pulsador externo
108 volatile int Tanque = 0;    // Variable que representa el numero de
    tanque que se muestra en la pantalla LCD
109 volatile int Bomba = 0;     // Variable que representa el numero de
    bomba que se muestra en la pantalla LCD
110 volatile int banderaInterrupcion = 0;      // Bandera de indicacion de
    ingreso en interrupcion por pulsador
111 volatile int banderaInterrupcionWifi = 0;  // Bandera de indicacion de
    ingreso en visualizacion WiFi
112 volatile int finBomba = 0;                 // Bandera de Indicacion de
    ultima bomba al ingresar por interrupcion del pulsador
```

Figura 23: Variables globales utilizadas en el código.

- **Funciones:**

Existen varias funciones desarrolladas en el programa. A continuación, se describirá brevemente el funcionamiento de cada función, dejando los detalles del código para el "Anexo C".

En la Figura 24 se presentan tres funciones. Las dos primeras, "interrupcionPulsador" e "interrupcionSensor", son funciones de interrupción alojadas en la RAM del ESP32 (IRAM_ATTR) para una respuesta más rápida. La primera de ellas gestiona las interrupciones del pulsador externo, que se utiliza para la visualización local de la información en la pantalla LCD. En este sentido, la pulsación del pulsador externo permite navegar entre los distintos menús: el porcentaje de llenado de cada tanque, el estado de cada bomba, el estado de la conexión a la red local WiFi y el estado de conexión con el servidor ThingsBoard.

```
115   // Funcion interrupcion para el pulsador externo
116 > void IRAM_ATTR interrupcionPulsador() {…}

136   #if numeromaxtanques > 0
137     // Funcion interrupcion para los sensores de nivel "flotadores" de los
        tanques
138 >   void IRAM_ATTR interrupcionSensor() {…}

156   #endif
157
158    // Funcion para evitar el bouncing (rebote indeterminado entre 0 y 1)
       del pulsador externo
159 > void eliminarRebotePulsador() {…}
```

Figura 24: Funciones interrupción del pulsador externo y de los sensores flotadores, función para la mitigación del efecto rebote en el pulsador.

La segunda función de interrupción se compila si hay tanques y se activa cuando alguno de los sensores flotadores de los tanques es accionado. Es decir, si se activa el sensor flotador inferior, se procede a encender las bombas locales (si hay bombas dependientes), y si se activa el sensor flotador superior, se apagan. Al finalizar la función, se activa la variable que permite enviar datos a la plataforma IoT. Esto es especialmente útil para aquellos ESP32 cuyos tanques no tienen bombas vinculadas de forma local, sino que requieren el encendido de una bomba controlada por otro ESP32.

Por último, la función "eliminarRebotePulsador" permite eliminar el rebote del pulsador por software.

```
179   #if numeromaxbombas > 0
180     // Funcion encendido de bombas locales
181   >   void encendidoLocalBombas(int j) {…}

188     // Funcion apagado de bombas locales
189   >   void apagadoLocalBombas(int j) {…}
195   #endif

197   // Funcion Delay (retardo) en milisegundos
198   > void retardoDeTiempo(int tiempoEsperadoMs) {…}
```

Figura 25: Funciones de encendido y apagado local y función delay mejorada.

Las funciones "encendidoLocalBombas" y "apagadoLocalBombas" de la Figura 25 compilan si el dispositivo tiene bombas vinculadas y son responsables del encendido y apagado local de las bombas en caso de que el ESP32 tenga bombas dependientes. Estas funciones son llamadas por la función "interrupcionSensor", mencionada anteriormente, según se active el sensor flotador inferior o el superior.

La función "retardoDeTiempo" es una función creada para generar un retardo (delay) en milisegundos, con la ventaja de que al utilizar el pulsador generándose una interrupción, se permita terminar con la función delay. Algo que no es posible utilizando la función por defecto de las bibliotecas de Arduino, donde la función delay no puede interrumpirse.

La Figura 26 muestra funciones relevantes para la medición del volumen de agua y el porcentaje de llenado de cada tanque. La función "retornoUltrasonico" se encarga, con base en los pines asignados a las variables Trigger y Echo, de enviar un pulso mediante el primero y recibirlo mediante el segundo. A partir de esto, se deduce el tiempo transcurrido desde que la onda es enviada hasta que es recibida; luego, este tiempo se divide a la mitad (para obtener solo el viaje de ida) y se convierte en una distancia en centímetros.

La función "calculoNivelTanque" se compila si hay tanques y toma como índice el valor de la variable "Tanque" y lo utiliza para buscar en los arreglos correspondientes del tanque específico, como se indica en la Figura 21 (Parte 2). En este contexto, arreglos como "alturaMaximaTanques", "distanciaSensorInferiorDPiso", "distanciaSensorSuperiorDTecho" y "diametroTanques" desempeñan un rol protagónico. Además, esta función llama a la función anterior "retornoUltrasonico", y a partir de la distancia en centímetros devuelta y de la información proporcionada por los demás arreglos, se obtiene el volumen de agua disponible y el porcentaje de llenado, en base a las características físicas de cada tanque.

```
209   // Funcion retorno en CM de distancia a objeto (por ultrasonico)
210 > int retornoUltrasonico(int Trigger, int Echo) {…}

222   #if numeromaxtanques > 0
223     // Funcion de calculo en porcentaje del nivel de agua del tanque
224 >   void calculoNivelTanque(int Tanque) {…}
241   #endif

243   // Funcion informacion de nivel de tanques en display LCD
244 > void infoTanquesLCD(int Tanque, int tanqueInicial, float porcentaje, int
      longitudPorcentaje) {…}

264   // Funcion informacion de estado de bombas en display LCD
265 > void infoBombasLCD(int Bomba, int bombaInicialZona) {…}
```

Figura 26: Función medición del retorno ultrasónico, cálculo del nivel de agua y funciones de visualización en LCD (bombas y tanques).

Por último, las funciones “infoTanquesLCD” e “infoBombasLCD” permiten ubicar el puntero y escribir en la pantalla LCD respecto al porcentaje de llenado de cada tanque y al estado de cada bomba.

La Figura 27 presenta las últimas funciones antes de la función “setup” y “loop”, que son las funciones por defecto de todo programa. “conexionWiFi” y “checkConexionTB” permiten visualizar en la pantalla LCD el estado de la conexión WiFi y el estado de conexión con el servidor de ThingsBoard, respectivamente. El parámetro de entrada representa un “Parámetro Configurable” de la Figura 18 y, por ende, puede establecerse el tiempo más conveniente para la visualización en segundos.

La función “envioDatosThingsBoard” es extensa. Su objetivo es enviar a la plataforma no solo los datos de telemetría del dispositivo, sino también los atributos. La transmisión de datos a la plataforma se realiza mediante el protocolo MQTT, enviando la información en pares clave-valor (key:value) en formato JSON. Para simplificar el código y evitar que el usuario intervenga cada vez que se configura un dispositivo con distinta cantidad de bombas y tanques, se crea un objeto JSON que toma los datos de las variables de los tanques y las bombas, y luego se serializa para enviarlo a la plataforma. Los datos de telemetría se envían periódicamente en un intervalo definido por el usuario, mientras que los atributos se transmiten solo una vez, al iniciar la conexión con el servidor o cuando esta se reinicia. Esto evita enviar información redundante, optimizando el uso de mensajes y respetando el límite mensual de mensajes permitido por la plataforma según el plan de suscripción. Los atributos enviados a la plataforma incluyen la cantidad de bombas y tanques vinculados al dispositivo, mientras que los datos de telemetría enviados abarcan el porcentaje de llenado de cada tanque, el estado de los sensores flotadores

(inferior y superior) de cada tanque, el volumen de agua de cada tanque y el estado de las bombas.

```
281    // Funcion de informacion de estado de conexion WiFi en display LCD
282  > void conexionWiFi(int tiempoPanelEnS) {…}

316    //Funcion para revisar la conexion a ThingsBoard en el menu del LCD
317  > void checkConexionTB(int tiempoPanelEnS) {…}

346    //Funcion para enviar datos a ThingsBoard referentes a los tanques y
         bombas locales
347  > void envioDatosThingsBoard() {…}

392  #if numeromaxbombas > 0
393    //Funcion RPC de biblioteca ThingsBoard para procesar las solicitudes RPC
         referidas a la activacion de las bombas independientes
394  > void procesarSetEstadoBomba(const JsonVariantConst &data, JsonDocument
         &response) {…}

405    // Define el array de callbacks RPC que apunta a la funcion de
         RPC_Response
406    const std::array<RPC_Callback, 1U> callbacks = {…}

409  #endif
411    // Establece la funcion que se ejecuta cuando desbordarda el timer

412  > void IRAM_ATTR timerEnvioDatos() {…}
```

Figura 27: Funciones de visualización LCD (WiFi y servidor IoT), envío de datos a ThingsBoard y funciones de recepción de mensajes RPC, función timer.

La función “procesarSetEstadoBomba” se compila si hay bombas y permite activar o desactivar las mismas al recibir un comando RPC en el dispositivo. Esta función recibe dos parámetros a través del mensaje RPC: el ID de la bomba (correspondiente al ID de la bomba en la zona) y el estado de la bomba (encendido o apagado). A partir de esta información, se controla la bomba correspondiente y se devuelve un mensaje con el estado actualizado.

La expresión dada en la línea 406, también se compila si hay bombas y define un arreglo de callbacks RPC, que actúa como un puente entre los mensajes RPC enviados desde la plataforma y la función de procesamiento especificada, en este caso, “procesarSetEstadoBomba”. Este arreglo de callbacks (que tiene un único elemento) especifica un "método" que debe coincidir con el método enviado en el mensaje RPC, y asocia este método a la correspondiente función de procesamiento.

Por último, la función “timerEnvioDatos” gestiona la operación cuando el timer desborda. Dentro de esta rutina, se habilita el envío de información a la plataforma.

```
421  > void setup() {…}

456  > void loop() {
457      retardoDeTiempo(10);
458  >   while (pulsador == 1) {…}
496  >   if (checkReconexionWifi >= tiempoCheckeoWifi / tiempoEnvioDatos) {…}
501  >   if (WiFi.status() != WL_CONNECTED && intentos <= maxIntentos) {…}
510  >   if (WiFi.status() == WL_CONNECTED) {…}
537       tb.loop();
538  }
```

Figura 28: Funciones Setup y Loop del programa.

Las últimas dos funciones del programa visualizadas en la Figura 28 son las de setup y loop que son las generales de cualquier programa en el IDE de Arduino. La función setup se ejecuta una única vez cuando el ESP32 se inicializa por primera vez, mientras que la función loop se repite indefinidamente.

La función “setup” se podrá ver en su total extensión en el “Anexo C”, pero principalmente es la encargada de configurar los pines de entrada y salida del ESP32. En esta función se definen como entradas, el pin del pulsador externo, los sensores flotadores inferiores y superiores y los pines Echo de los sensores ultrasónicos para cada tanque. Se definen como salida, los pines Trigger de cada tanque y las bombas en caso de estar vinculadas al ESP32. Además, en esta función se deben asignar las interrupciones a los pines correspondientes y se debe especificar como se gatilla cada interrupción. Otras acciones que se ejecutan en esta rutina es la inicialización de la pantalla LCD, el inicio de la bomba local correspondiente si el nivel de agua en un tanque está bajo el sensor inferior y la configuración del temporizador.

Por otro lado, la función “loop”, que también se podrá visualizar en el “Anexo C”, tiene cuatro subrutinas. La primera está dada cuando se presiona el pulsador externo, y la lógica de la sucesión de las pantallas está en la línea 458. Al presionar el pulsador externo, la subrutina se ejecuta y recién finaliza al pasar por todas las pantallas. Pasar de una pantalla a la siguiente puede realizarse automáticamente (tiempo dado en los “parámetros configurables”) o manualmente, mediante el pulsador. Mientras la visualización local este corriendo, el temporizador se congela y no será posible enviar información a la nube, esto se realiza porque de lo contrario puede producirse un conflicto al enviar los datos y volver a la visualización local.

Lo expresado en la línea 496 es la revisión del tiempo para reinicializar los intentos de reconexión al WiFi local.

En la línea 501 se realiza la conexión al WiFi, y se incrementa un intento, si el ESP32 no se puede conectar a la red local. Luego, si la cantidad de reconexiones supera “maxIntentos”, dejara de intentar conectarse y recién se habilitará las nuevas reconexiones cuando se reinicialice los intentos de reconexión dados por la línea 496.

Por último, lo expresado en la línea 510 habilita la conexión al servidor IoT, se suscribe a los mensajes RPC y si está habilitado, envía los mensajes a la plataforma IoT.

La línea 537 permite que la biblioteca ThingsBoard con todas sus funciones mantengan vivo el enlace entre el dispositivo y la plataforma.

3.2.2 Desarrollo del panel de control en ThingsBoard

En la siguiente sección se desarrollará el diseño y elaboración de la instancia creada en ThingsBoard para el proyecto.

- **Dispositivos y assets**

Antes de comenzar a desarrollar el panel de control, primero se deben añadir dispositivos y crear los assets (locaciones) respectivos. Estas interfases pueden visualizarse en las Figuras 29 y 30.

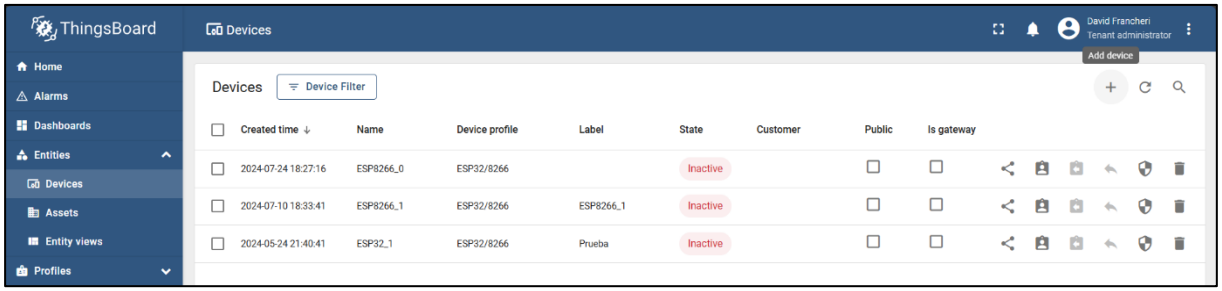


Figura 29: Interfaz en ThingsBoard para la creación y el control de dispositivos.

Al añadir dispositivos se le da un nombre, un label y un “device profile”, en este caso, el perfil es el mismo para todos los dispositivos que se crearían, pero si la naturaleza de los dispositivos fuera otra, podrían crearse distintos perfiles y asignarlos de manera adecuada. La creación del dispositivo ofrecerá un token que es el que añadimos en la programación del ESP32.

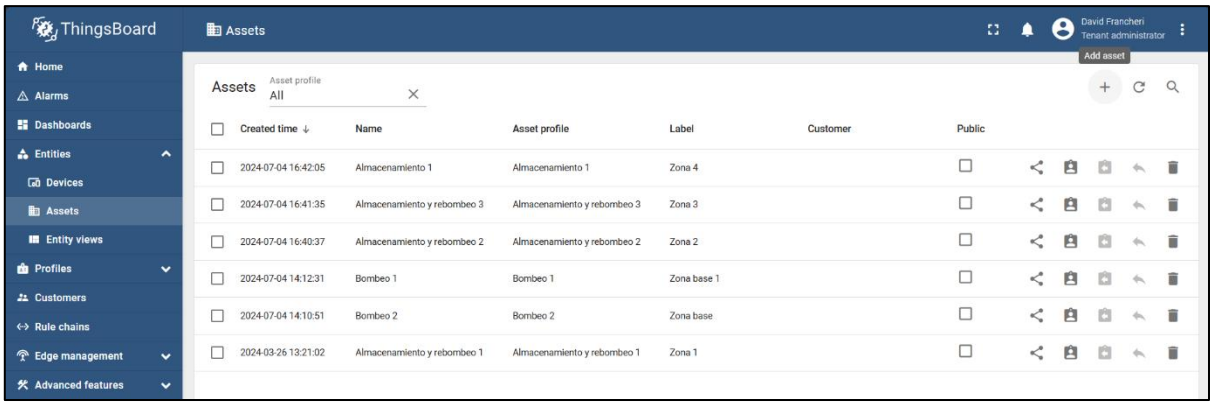


Figura 30: Interfaz en ThingsBoard para la creación y el control de assets.

De manera análoga, se define un nombre, un label y un “asset profile” para los assets. Para este proyecto, si es necesario la creación de estos perfiles, los cuales se crean en semejanza con los assets, es decir, existirá un perfil por cada asset. Esto se hace para que diversos widgets dentro del panel de control se puedan visualizar correctamente.

A los assets se les pueden asignar atributos de servidor, definidos manualmente por el usuario en la plataforma. En este proyecto, a cada asset se le añadió un par clave-valor con la latitud y longitud, con el objetivo de representarlos posteriormente en un widget de mapa. Además, se incorporó una clave denominada "connectedDevices" con valor nulo, pensada para reflejar en tiempo real los dispositivos conectados en la zona correspondiente. Finalmente, se agregaron los atributos "Bombas" y "Tanques", también con valores nulos, que permiten indicar —en el dashboard principal— qué bombas y tanques se encuentran "en línea" en función del estado de conexión de los dispositivos.

- **Alarmas**

Las alarmas en ThingsBoard se pueden agregar a través de los “device profile” o mediante Rule Chain. Para mayor simplicidad, los desarrolladores de la plataforma diseñaron la primera de estas opciones, cuando la especificidad de las alarmas no es muy alta. Para este proyecto es suficiente.

Las alarmas pueden clasificarse en cinco grados de severidad: Critico, mayor, menor, advertencia e indeterminado. Pueden activarse en condiciones temporales definidas, o en cualquier momento. Además, permiten activarse en distintas condiciones, lo cual puede implicar, una activación simple como por ejemplo que un par clave:valor pase de 0 a 1, o puede ser una activación por repetición, que luego de “x” veces de pasar de 0 a 1 se active o puede

activarse por duración, es decir, si el estado de 0 a 1 se mantiene durante “x” segundos, minutos o horas.

Para este PIP se añadieron las alarmas de un tanque, pero en Aluminé, deberían añadirse una cantidad de alarmas equivalente a la cantidad máxima de tanques que exista en alguna de las 6 zonas. La Figura 31 ilustra alarmas creadas para el “Tanque 1”. Se debe tener en cuenta que, dado que el perfil es único para todos los ESP32, no hay problema en crear alarmas para tanques que el dispositivo no tenga, ya que las alarmas se activan a partir de datos de telemetría, si los datos no son enviados desde ese dispositivo, entonces ninguna alarma se activara.

En el caso de la Figura 31, las alarmas creadas “Nivel de desborde” y “Nivel de vaciado” están relacionadas a los datos de telemetría del estado de los sensores flotadores superior e inferior respectivamente.

Acorde a la Figura 32, la alarma del nivel de desborde con severidad critica se puede activar si el estado de la clave (tag) “EstadoSensorSuperiorTanque1” es true (este par clave:valor es un booleano) durante 1 minuto, mientras que alcanza una severidad de advertencia si cambia a true en forma simple. Esto se consideró porque al aumentar el nivel de agua del tanque, el oleaje del agua puede generar un momento de “falso desborde”.

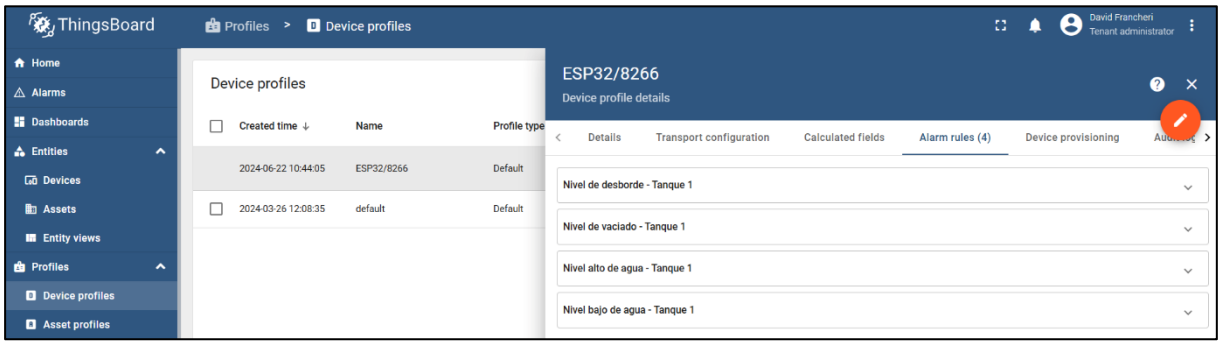


Figura 31: Creación de alarmas dentro de “device profiles”.

Abajo se debe definir la condición de limpieza de la alarma, la cual ocurre si el estado del sensor superior vuelve a false luego de 1 minuto. Las alarmas podrán visualizarse en el panel de control y pueden verificarse y eliminarse.

De manera análoga, pero con el sensor flotador inferior se planteó la alarma del vaciado del tanque. Las alarmas de “Nivel alto de agua” y “Nivel bajo de agua” son alarmas que avisan cuando el porcentaje de llenado del tanque está en un nivel alto (hay severidad mayor al 95% y severidad menor al % 90) y cuando el porcentaje de llenado está en un nivel bajo (severidad mayor al 5% y severidad menor al 10%).

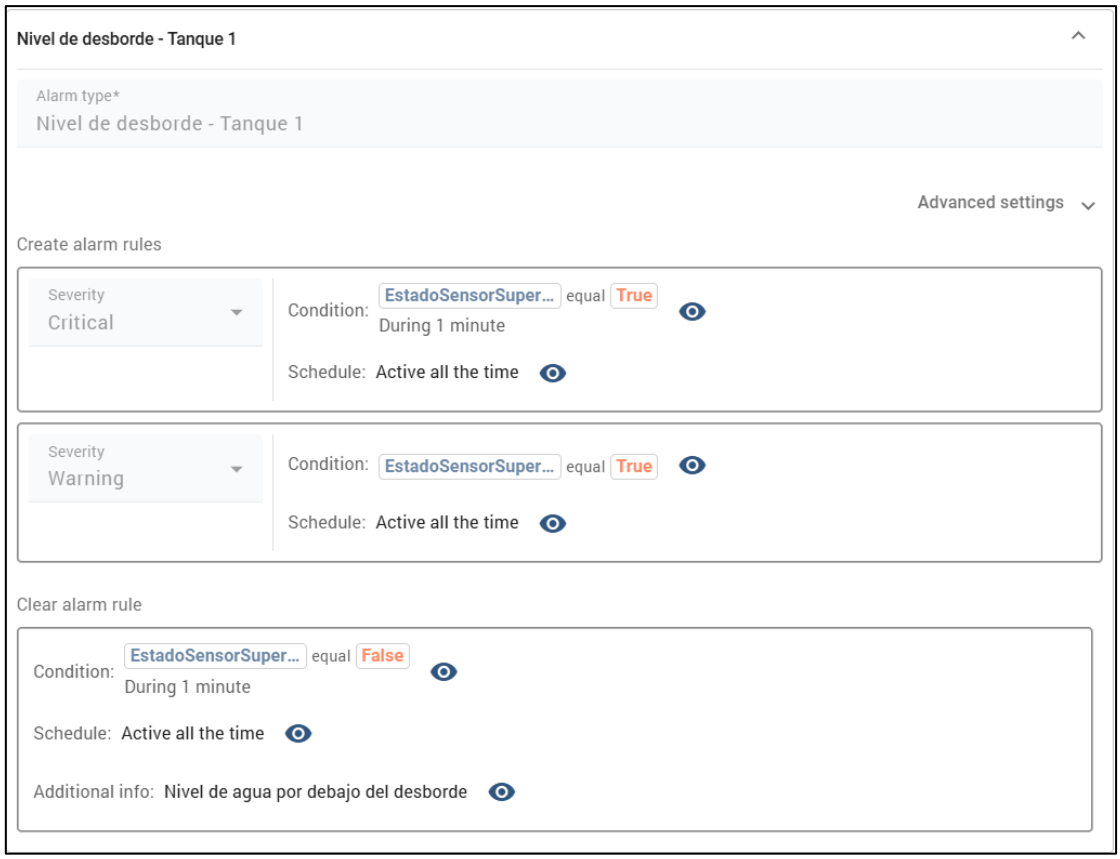


Figura 32: Reglas para la Alarma "Nivel de desborde - Tanque 1".

- **Dashboards**

En la Figura 33 se puede observar la interfaz para añadir dashboards (paneles de control), que es la herramienta visual utilizada en las plataformas IoT para la visualización gráfica de todos los datos de telemetría enviados por los dispositivos inteligentes.

Para este proyecto, se creó un único dashboard llamado "Principal". El dashboard "Home" fue generado por la cuenta demo (junto con otros que posteriormente fueron eliminados) y contiene algunos consejos.

Antes de visualizar las distintas instancias dentro del dashboard, es importante mencionar la visibilidad que tendrá, es decir, la posibilidad de que otros usuarios accedan a la visualización del mismo. La cuenta actual utilizada es una cuenta de tipo "tenant", lo que le otorga control total sobre la edición y visualización.

Para permitir que otros usuarios visualicen el dashboard, se puede crear un "customer" y especificar datos como el correo electrónico. Esto enviará un enlace a esa cuenta, y el usuario podrá crear su contraseña desde el enlace recibido. La cuenta customer es limitada: solo podrá visualizar alarmas y aquellos elementos que el tenant le asigne. En este caso, será necesario

asignarle el dashboard, los dispositivos y los assets. De este modo, el customer podrá visualizar toda la información disponible en el dashboard, se debe mencionar que la suscripción mensual más económica de ThingsBoard Cloud solo admite la creación de dos de estas cuentas.

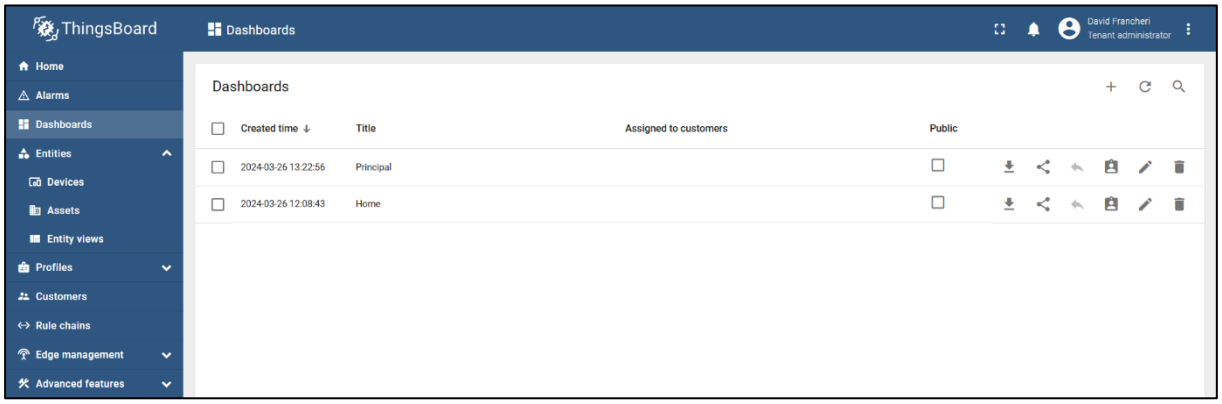


Figura 33: Creación de dashboards (panel de control) y asignaciones de visibilidad.

Otra forma de compartir el dashboard con otros usuarios es hacerlo público. Esto implica hacer público el dashboard y todos los dispositivos y assets asociados. Al hacer el dashboard público, se generará un enlace que podrá ser compartido con cualquier persona, incluso si no tiene cuenta en ThingsBoard. Esta opción es menos segura, pero más sencilla y rápida que la creación de un customer, además de que se puede compartir con cuantas personas se quiera.

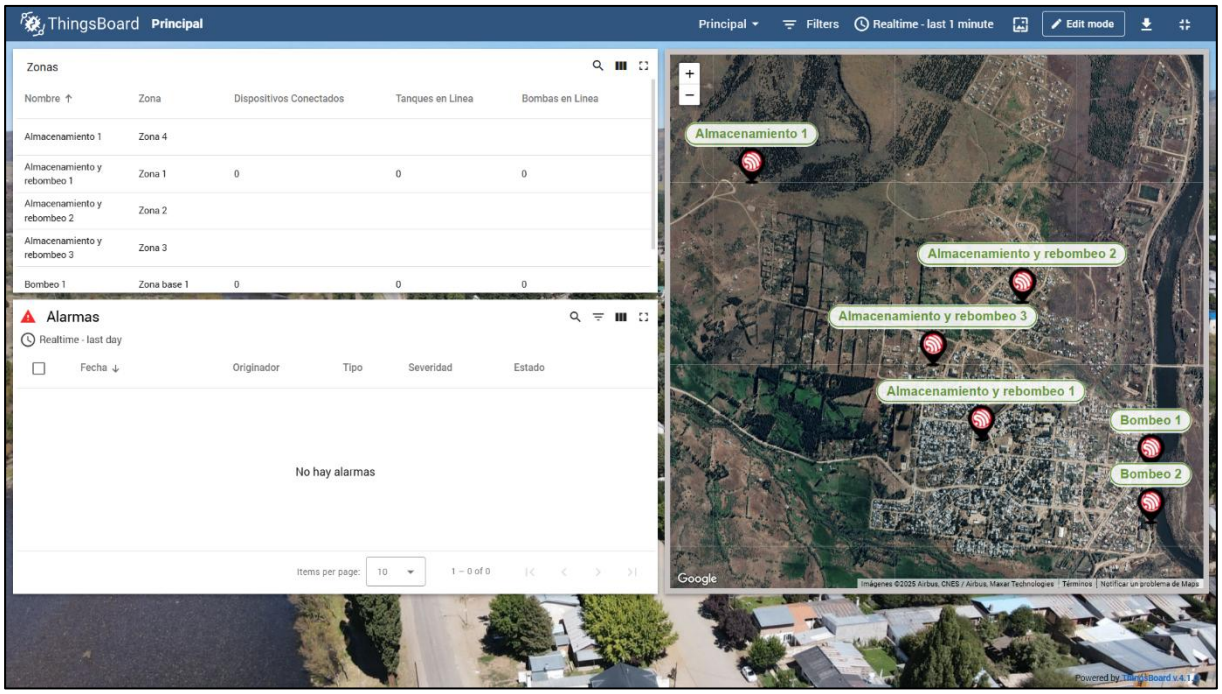


Figura 34: Pantalla principal del panel de control para el proyecto.

La Figura 34 ilustra la pantalla de inicio del panel de control del proyecto. El botón “edit mode”, ubicado en la esquina superior derecha, solo está habilitado para el usuario tenant. No está disponible ni para el usuario customer ni en el modo público.

El panel de control creado tiene tres niveles. El mostrado en la Figura 34 corresponde al nivel más general. En este, se visualiza un mapa satelital de la zona de Aluminé, con la ubicación aproximada, en latitud y longitud, de las locaciones que tienen bombas y/o tanques. A la izquierda se observa una tabla de entidades; en este caso, las entidades son assets (las zonas), y se puede visualizar información relevante de cada una.

Cada zona tiene un indicador de los dispositivos conectados. Cuando un dispositivo aparece como “en línea”, es decir, cuando envía telemetría a ThingsBoard, y debido a una regla dentro del rule chain, primero se filtran los mensajes por zona y luego se incrementa la variable “connectedDevices” en el asset correspondiente. Cuando un dispositivo pierde conexión con la plataforma, la variable “connectedDevices” decrece, lo que indica la pérdida de conexión de ese dispositivo. Además de este indicador, en la tabla también se pueden visualizar la cantidad de bombas y tanques en línea vinculados a los dispositivos de cada zona. Esta información proviene de los atributos enviados por cada ESP32 y se actualizan en tiempo real con la conexión o desconexión de dispositivos.

Por último, en la parte inferior izquierda, se visualizan las alarmas, que son globales para todas las zonas y dispositivos. En esta ventana se muestran los estados de alarma de cada uno de los dispositivos distribuidos por Aluminé.

Para analizar la información de una zona específica, se puede hacer clic en el icono del mapa, como se ilustra en la Figura 35, o acceder a través de la tabla de entidades.

Al ingresar en una zona, por ejemplo, “Almacenamiento y rebombeo 1”, se visualizará, según lo indicado en la Figura 36, una tabla con las entidades de dispositivos. Esta tabla incluirá información relacionada con el estado de conexión, la fecha y hora de la última actividad, así como la cantidad de bombas y tanques vinculados. Además, se presentará un gráfico general de la zona que muestra el porcentaje de llenado de todos los tanques (es importante recordar que cada zona tiene tanques con ID diferentes).

La ventana de alarmas muestra el estado de las mismas para todos los dispositivos de la zona en cuestión.

Figura 35: Visualización de ventana pop-up al hacer clic en las zonas del mapa.

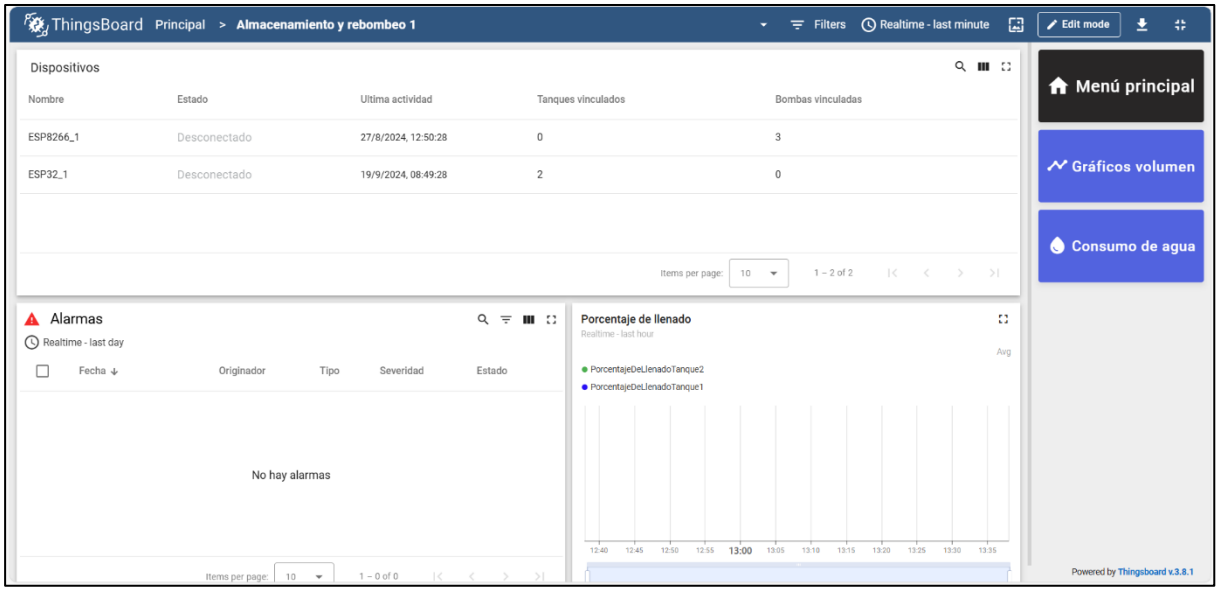


Figura 36: Pantalla de visualización a nivel de zona en ThingsBoard.

En el lado derecho, se encuentran los botones de navegación, que permiten regresar al menú principal o visualizar gráficos de volumen y consumo de agua. Los primeros permiten ver tanto el volumen histórico como el volumen en tiempo real de cada tanque de la zona, mientras que los segundos indican el cambio de volumen (delta) entre dos mediciones consecutivas, lo cual se puede modificar en la cadena de reglas (rule chain).

En las Figuras 37 y 38 se pueden visualizar los gráficos de volumen y consumo de agua. Existe un gráfico en tiempo real con una ventana de 2 horas y un gráfico histórico que muestra los datos de cada tanque en un lapso de 1 semana.

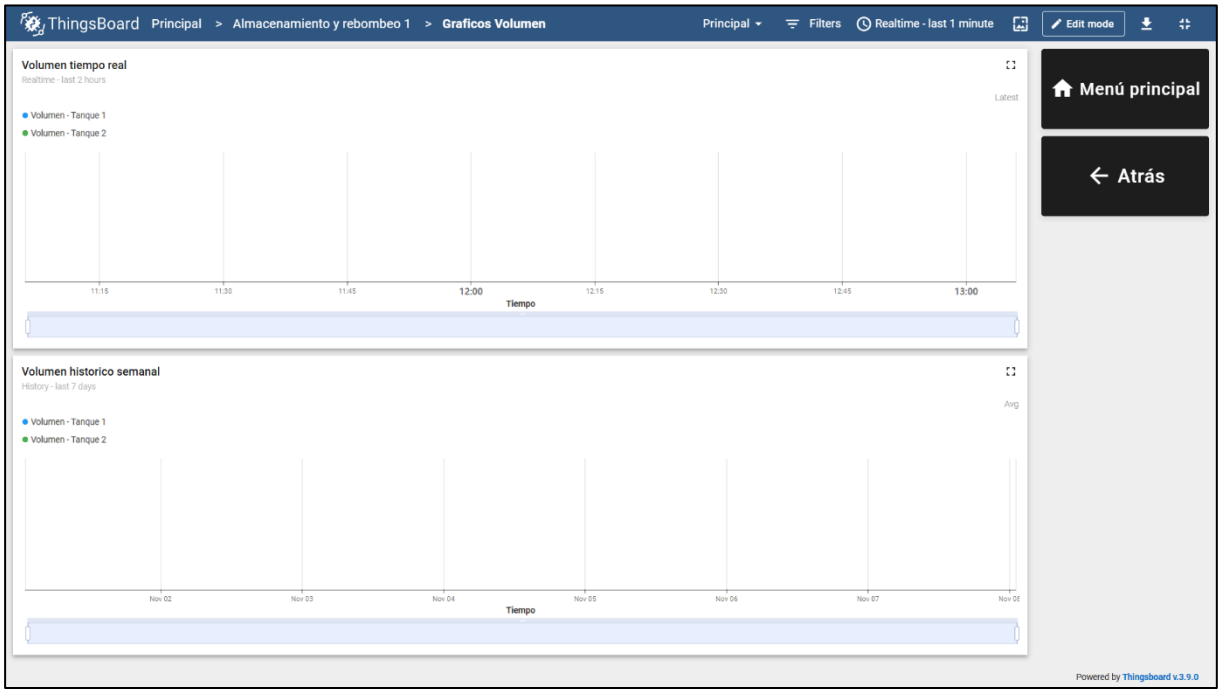


Figura 37: Gráficos de volumen de los tanques en la zona.

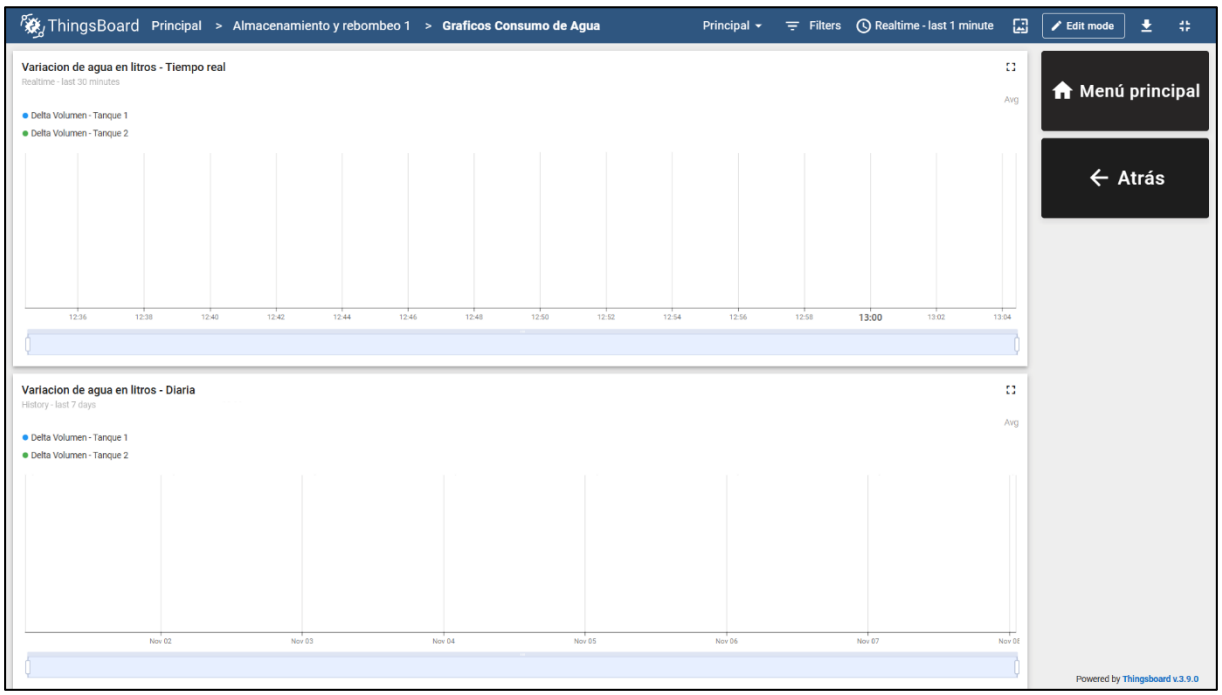


Figura 38: Gráficos de consumo de agua según cada tanque de la zona.

Estos gráficos pueden observarse en períodos de tiempo diferentes a los mostrados en las imágenes. ThingsBoard permite visualizar los datos en escalas de tiempo configurables. Esto se puede ver en la Figura 39, donde para el gráfico en tiempo real se puede definir el período

de tiempo, que varía desde 1 segundo hasta 12 horas, y luego en intervalos de días, hasta 30 días. También es posible establecer un período personalizado.

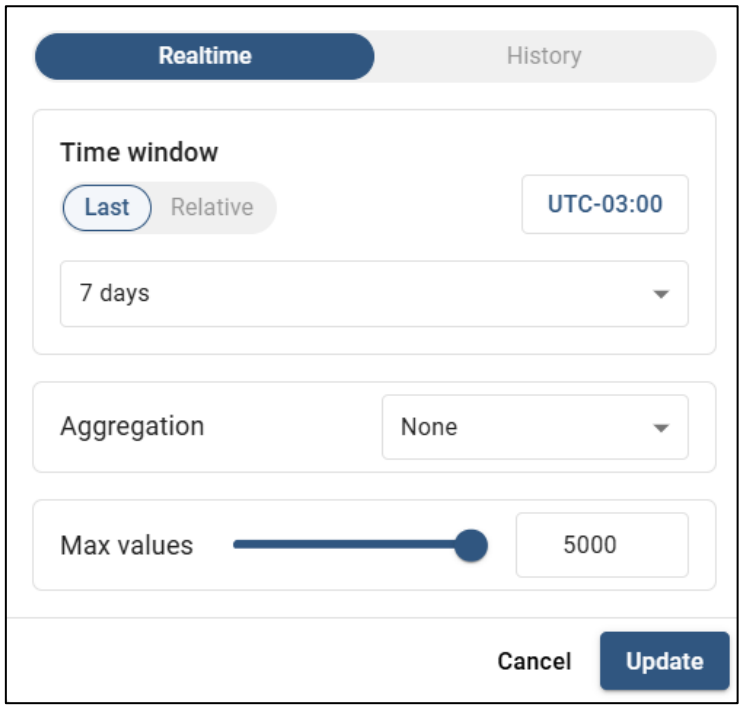


Figura 39: Modificación del eje temporal y agregación de información en los gráficos.

Otra función interesante es la opción de "agregar" información, si se desea. De esta manera, es posible agrupar las muestras en un período de tiempo específico y, a partir de ello, obtener valores como el mínimo, el máximo, el promedio, la suma total o el conteo.

El último nivel de especificidad corresponde a los dispositivos, al que se puede acceder desde la tabla de entidades de cada zona.

La pantalla de visualización para este nivel de dispositivo se muestra en la Figura 40. En ella, se podrá controlar el estado de conexión del dispositivo, la hora del último estado, las alarmas asociadas, el porcentaje de llenado de los tanques correspondientes, así como la posibilidad de acceder a los menús de los tanques y las bombas vinculadas.

El acceso a estos menús depende de si el dispositivo está vinculado a tanques y/o bombas. En caso de que el dispositivo esté vinculado, aparecerá un "ojo" visible como botón pulsador. Si el dispositivo no está vinculado a tanques o bombas, el "ojo" no será visible.

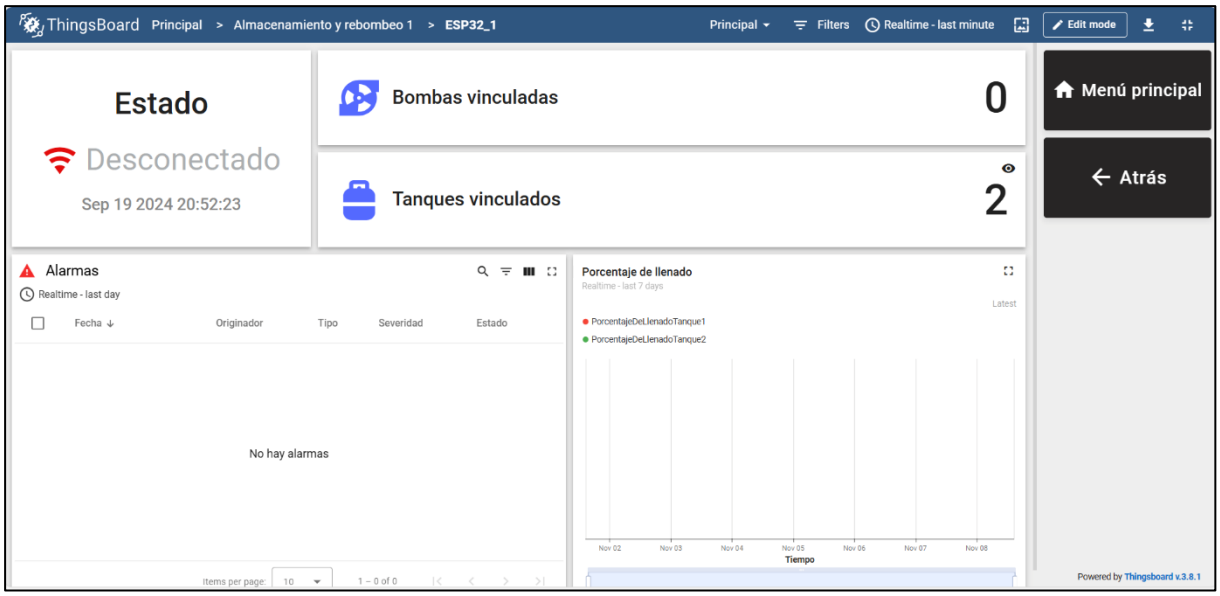


Figura 40: Pantalla de visualización a nivel del dispositivo en ThingsBoard.

Al ingresar en la sección de tanques, se podrá visualizar, tal como se muestra en la Figura 41, el porcentaje del nivel de agua de cada uno. En caso de que existan más tanques de los que tiene vinculados el dispositivo, aparecerá el mensaje “No hay información” y, por lo tanto, solo se mostrará la información de los tanques disponibles. Como máximo, es probable que un dispositivo tenga vinculado hasta 4 tanques, lo que implicaría que en el menú se visualizarían un máximo de 4 tanques.

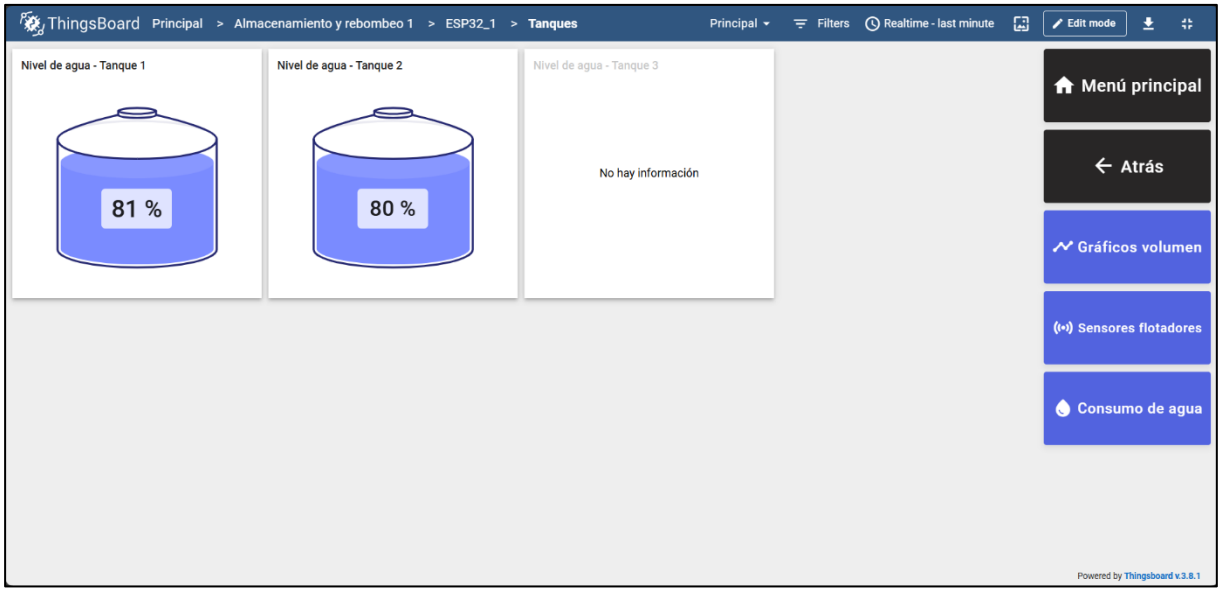


Figura 41: Pantalla de visualización de los tanques para un dispositivo en ThingsBoard.

Los gráficos de volumen y consumo de agua, en este caso, mostrarán únicamente el estado de los tanques vinculados al dispositivo. Por su parte, el menú de los sensores flotadores

mostrará el estado encendido/apagado (nivel normal/nivel crítico) de los sensores de nivel de cada tanque vinculado al dispositivo.

En el caso de los dispositivos que tengan bombas vinculadas, se podrá visualizar un menú similar al presentado en la Figura 42. En este menú, se mostrará el estado de cada bomba (encendida/apagada) y un LED que se ilumina en verde al reconocer el encendido de una bomba.

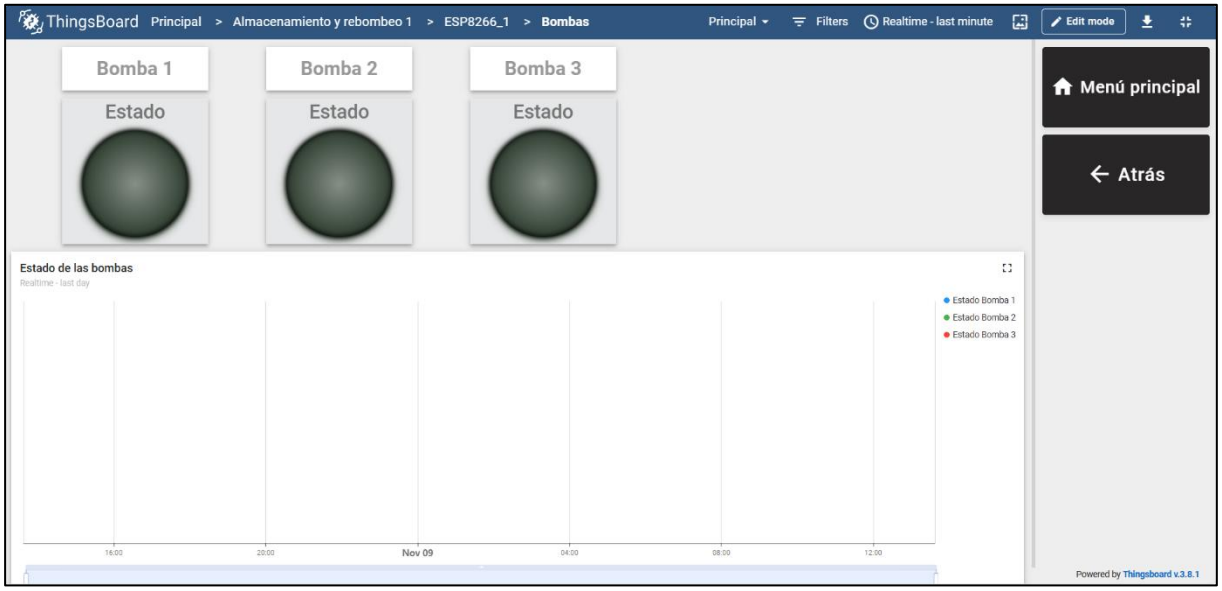


Figura 42: Estado de las bombas vinculadas a un dispositivo en ThingsBoard.

Debido a una limitación del widget, solo se puede definir el color del LED cuando está encendido, y no se reconoce un estado de "no hay información", incluso si se establece un par clave:valor que no exista para el dispositivo. Por lo tanto, la visualización de los LEDs debe hacerse con cuidado, ya que cuando el LED no está iluminado en una bomba, puede deberse a que la bomba está apagada o a que la bomba no está vinculada al dispositivo.

Por último, mencionar que para la asignación de “datasource” a los widgets en muchos casos fue necesario crear los llamados “Entity Aliases”. Estos, son una referencia a una o a un conjunto de entidades utilizadas en la visualización de los widgets. En la Figura 43 se puede apreciar, los que fueron creados y utilizados en este proyecto en widgets como, la tabla de entidades de zonas, el panel de alarmas globales (todos los dispositivos de todas las zonas), widgets gráficos, los de los tanques animados, entre otros.

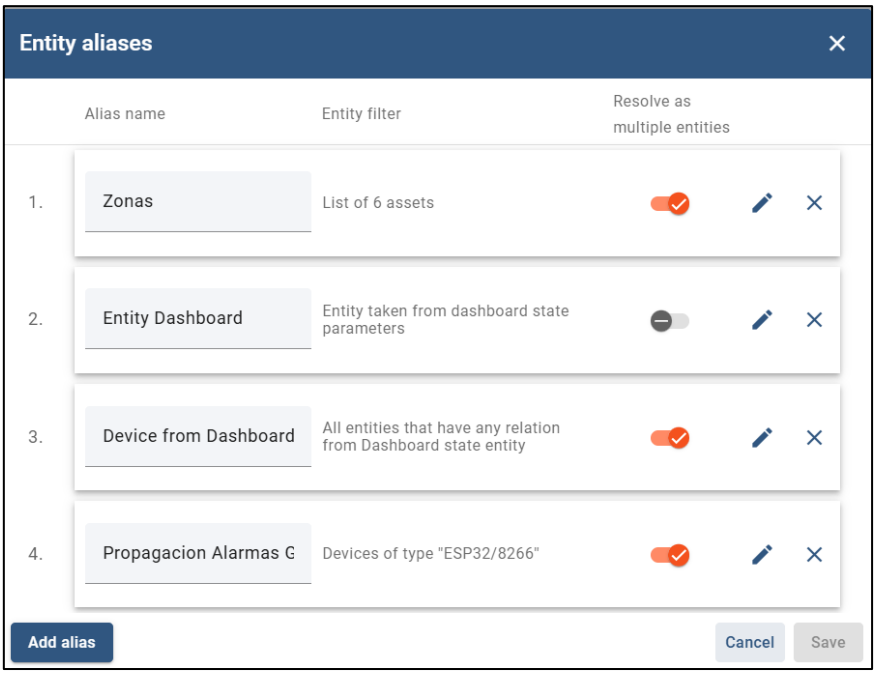


Figura 43: Menú de creación de "Entity Aliases" en ThingsBoard.

- **Rulechain:**

Como se mencionó anteriormente, ThingsBoard utiliza un motor de reglas (rule engine) que, mediante cadenas de reglas (rule chains), permite la manipulación y el control de los mensajes que entran y salen de la plataforma. Las rule chains son fundamentales en el desarrollo de proyectos IoT dentro de la plataforma, tanto como la creación de dashboards.

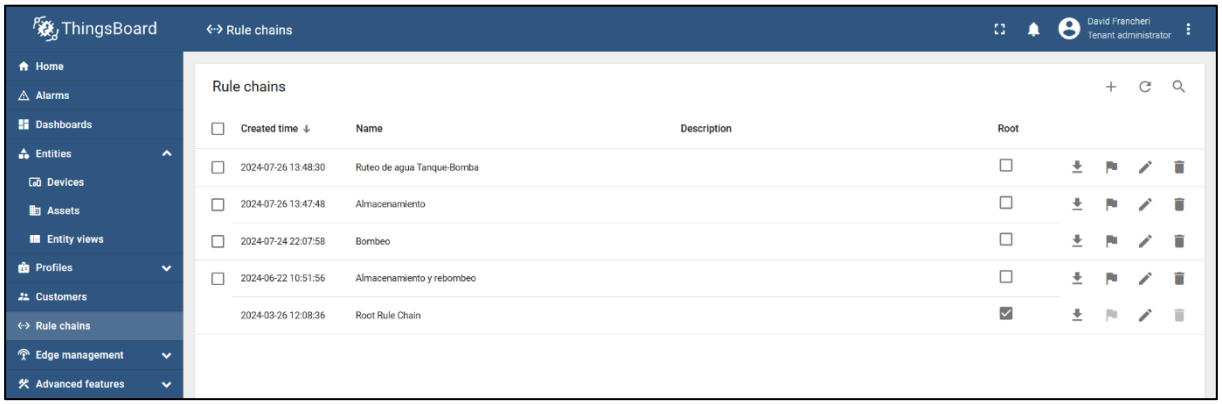


Figura 44: Creación de rule chains en ThingsBoard.

El usuario puede crear tantos rule chains como desee, siempre que no exceda las limitaciones de su plan de suscripción mensual. En la práctica, para este proyecto, el plan más básico debería ser suficiente para cubrir todas las necesidades de Aluminé, incluso considerando futuras expansiones. Sin embargo, una de las limitaciones de este plan es que solo permite la creación de hasta cinco rule chains. Aunque esto podría sortearse, el problema radica en que generaría rule chains sobrecargados, lo que dificultaría el seguimiento, control y edición de los mismos.

La creación de rule chains tiene una interfaz similar a la de otras estructuras ya vistas, y se puede ver en la Figura 44.

La idea es crear los cinco rule chains que se detallan a continuación:

1. Ruteo de agua Tanque-Bomba: Gestiona la activación de bombas remotas, estableciendo la relación entre los tanques que requieren el encendido de una bomba en otra zona, enviando comandos RPC a esos dispositivos.

2. Almacenamiento: Corresponde a las zonas donde solo existen tanques que almacenan agua. En el caso de Aluminé, esto se refiere a la zona más alejada.

3. Bombeo: Corresponde a las dos zonas a orillas del río Aluminé, donde se encuentran principalmente las bombas, que efectúan el primer bombeo hacia las plantas de almacenamiento y rebombeo.

4. Almacenamiento y rebombeo: Se refiere a las tres zonas centrales, donde el agua proveniente del río Aluminé se almacena y luego se rebombea a otras áreas. En algunas de estas zonas también se encuentran plantas potabilizadoras.

5. Root rule chain: El rule chain raíz. Este es el rule chain más relevante, ya que está definido por defecto. Esto significa que los mensajes comienzan en este rule chain. Dependiendo de la clase y tipo de mensajes que lleguen a este punto, se determinará el tratamiento que recibirán y su posterior direccionamiento hacia uno de los cuatro rule chains descriptos anteriormente.

En la Figura 45 se puede visualizar el rule chain principal o “root rule chain”. Este es el rule chain que tiene por defecto todos los mensajes de los dispositivos, ya sean atributos, telemetría, mensajes RPC, de actividad (internos) entre otros.

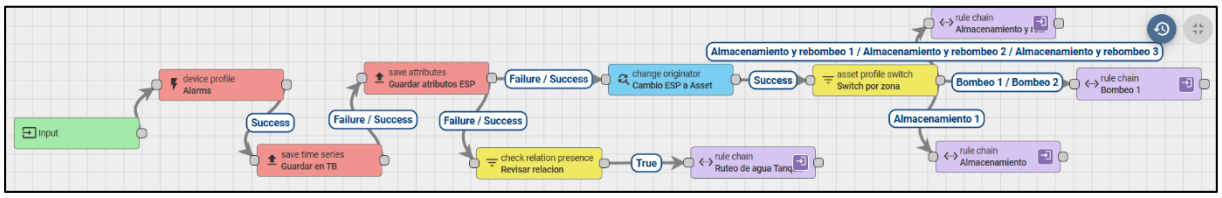


Figura 45: Root rule chain (rule chain raíz) en ThingsBoard.

El objetivo de esta root rule chain es principalmente dirigir los mensajes hacia los distintos rule chains. Primero, los mensajes pasan por el nodo que habilita las alarmas. A continuación, tanto

los mensajes de telemetría como los atributos se almacenan en la base de datos de los dispositivos.

Luego, se realiza una bifurcación:

1. Los mensajes de los dispositivos que tienen tanques con carga de agua mediante bombas remotas se dirigen al rule chain denominado "Ruteo de agua tanque-bomba", a través de un filtro de relación.

2. Los mensajes de todos los dispositivos siguen su curso cambiando de originador, de manera que la entidad emisora se convierte en los assets respectivos de cada dispositivo. Desde allí, se efectúa un switch para derivar los mensajes hacia el rule chain correspondiente a cada zona.

La Figura 46 muestra el rule chain para los mensajes que arriban a las zonas “Almacenamiento y rebombeo”. En este caso, solo se desarrolló la rama para “Almacenamiento y rebombeo 1”, pero el desarrollo para las otras zonas sigue una estructura análoga dentro del mismo rule chain. Además, se consideró la existencia de dos tanques; en caso de haber más, sería necesario extender el desarrollo, añadiendo bloques y modificando los scripts.

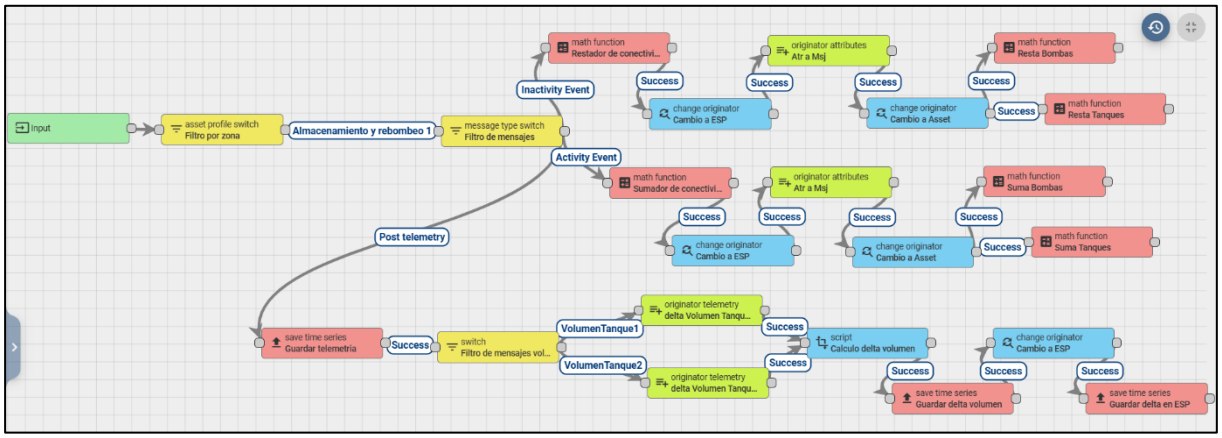


Figura 46: Rule chain para zonas de almacenamiento y rebombeo en ThingsBoard.

El camino de los mensajes comienza al pasar por el filtro de locación, lo cual se realiza mediante el filtro Asset Profile Switch. Luego, los mensajes son filtrados por tipo mediante el filtro Message Type Switch, que clasifica los mensajes en telemetría, actividad e inactividad.

Los mensajes de telemetría se almacenan en la base de datos del asset y, a continuación, pasan por un filtro Switch específico para los mensajes relacionados con el volumen de agua de los tanques. Para cada tanque, se debe generar una conexión de salida; en este caso, se desarrolló para dos tanques.

Los mensajes de volumen llegan a los nodos de enriquecimiento, donde se rastrea la muestra anterior y se copia el volumen en los metadatos del mensaje entrante. Esto permite, en el posterior nodo script de transformación, generar un nuevo mensaje que presenta el resultado de la diferencia entre el volumen de una muestra anterior y el volumen actual. De esta manera, se obtiene el delta de volumen, que luego se guarda en la base de datos tanto del asset como del dispositivo.

Esta rama del rule chain es clave para poder mostrar la información en los gráficos del consumo de agua.

Las otras dos ramas están relacionadas y se refieren a la actividad/inactividad del dispositivo y a la cantidad de bombas y tanques vinculados en línea. Como se mencionó anteriormente, en la plataforma, cuando finaliza el plazo de desconexión de un dispositivo (10 minutos en ThingsBoard Cloud sin actividad), se genera un mensaje de actividad del dispositivo, lo que reinicia el contador de inactivityTimeout hasta el siguiente evento de inactividad. Esto provoca un falso periodo de actividad para el dispositivo.

El inconveniente que surge de esta situación se presenta en el estado de conexión del dispositivo y en la cantidad de bombas y tanques en línea por zona. Durante el periodo de tiempo determinado por el inactivityTimeout, el dispositivo aparecerá como activo, aunque en realidad no esté enviando datos. Además, durante este periodo, también se mostrará que las bombas y los tanques vinculados están en línea en el dashboard general de las zonas. Este comportamiento está programado en la plataforma y la solución para una visualización corregida en los widgets generaría una extensión muy larga dentro del rule chain.

Cuando se envía el primer mensaje desde el dispositivo a la plataforma, se genera un evento de actividad que es dirigido por el rule chain hacia un nodo de función matemática, donde se incrementa unitariamente la variable “connectedDevices” del asset. De manera análoga, cuando el dispositivo queda inactivo (es decir, cuando finaliza el tiempo asignado en inactivityTimeout), el evento de inactividad llega a otro nodo de función matemática, donde se decrementa unitariamente la misma variable.

Para computar (agregar o restar) la cantidad de bombas y tanques vinculados a cada dispositivo de la zona en el asset, primero, después del nodo de función matemática, se realiza un cambio de originador al dispositivo. Luego, mediante un nodo de enriquecimiento, se buscan los atributos “bombasVinculadas” y “tanquesVinculados” enviados por el programa del ESP32,

y se agregan al cuerpo del mensaje. A continuación, se vuelve a cambiar el originador al asset y, para finalizar, se suman (agregan) o restan (dependiendo de si es un evento de actividad o inactividad) los valores de estos atributos con los valores de los atributos del servidor “Bombas” y “Tanques” del asset.

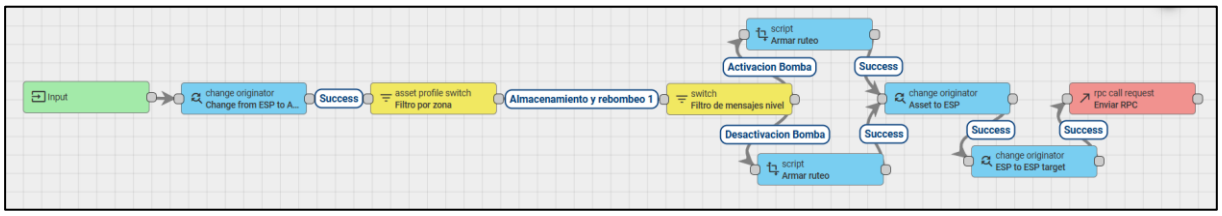


Figura 47: Ruteo tanque-bomba para la activación remota en ThingsBoard.

El otro rule chain desarrollado en este PIP es el presentado en la Figura 47, que corresponde al ruteo entre tanques y bombas ubicados en distintas zonas geográficas, con vinculación a dispositivos diferentes. Este rule chain es crucial, ya que representa probablemente la condición más común en Aluminé, y permite que, al vaciarse o desbordarse un tanque, se active remotamente una bomba que puede estar en un nivel geográfico inferior, vinculada a otro dispositivo.

Antes de seguir el camino de los mensajes que llegan a este rule chain paso a paso, es importante tener en cuenta que, para que una bomba remota se active, se debe enviar el comando RPC desde el mismo dispositivo que la controla. Por lo tanto, el ruteo debe llevar el mensaje desde el dispositivo solicitante hasta el dispositivo demandado dentro de la misma plataforma.

En primer lugar, los mensajes que llegan a este rule chain son aquellos que, previamente, en el root rule chain, pasaron el filtro de relación entre dispositivos. Al llegar, el mensaje cambia su originador al asset correspondiente. Luego, pasa por un filtro por zona y, posteriormente, por un filtro de mensajes de telemetría relacionados con el nivel de agua, es decir, con el estado de los sensores flotadores superiores e inferiores de los tanques. Este filtro, de tipo switch (implementado mediante un script), divide los mensajes según el estado de los sensores flotadores superiores e inferiores. Si se activa el sensor superior, se genera la salida de conexión “Desactivación Bomba”; si es el sensor inferior el que se activa, se genera la salida “Activación Bomba”.

A continuación, el mensaje transita por sendos scripts de transformación, cuyo objetivo es crear un nuevo mensaje con las características de un mensaje RPC. Por ejemplo, en el caso del estado del sensor inferior, el script resultante sería similar al que se muestra en la Figura 48.

Las variables idBomba y estado representarán el ID de la bomba correspondiente a la zona y el estado de la misma (1 para encendido y 0 para apagado). Luego, se inicializa la variable del mensaje y se agregan los metadatos necesarios para configurar características en los mensajes RPC. Esto incluye establecer el tiempo de expiración del mensaje, especificar si es de único sentido o bidireccional (es decir, si debe recibir una respuesta por parte del dispositivo) y determinar si el envío de RPC será persistente en caso de desconexión o expiración del mensaje.

```
var idBomba;
var estado;

// Crear un nuevo mensaje RPC
var msgEnviado = {};
var metadata = {
    deviceName: metadata.deviceName,
    expirationTime: new Date().getTime() + 60000,
    oneway: false,
    persistent: false
};

// Verificar las condiciones basadas en los datos de telemetría y efectuar el ruteo con la idBomba y el estado para con el dispositivo que tiene la bomba

if (msg['EstadoSensorInferiorTanque2'] === true) {
    idBomba = 1; // Asignar el ID de la bomba correspondiente para la zona
    estado = 1; // Encender la bomba
    msgType = 'RPC_CALL_FROM_SERVER_TO_DEVICE';
}
if (msg['EstadoSensorInferiorTanque1'] === true) {
    idBomba = 2; // Asignar el ID de la bomba correspondiente para la zona
    estado = 1; // Encender la bomba
    msgType = 'RPC_CALL_FROM_SERVER_TO_DEVICE';
}
// Asignar el método del RPC y los parámetros correspondientes
msgEnviado.method = 'setEstadoBomba'; // Nombre del método RPC que se invocará en el dispositivo
msgEnviado.params = {
    idBomba: idBomba, // Identificador de la bomba que debe ser controlada
    estado: estado // Estado que debe establecerse (1 = encender, 0 = apagar)
};

// Retornar el nuevo mensaje, los metadatos y el tipo de mensaje
return {
    msg: msgEnviado,
    metadata: metadata,
    msgType: msgType
};
```

Figura 48: Creación del mensaje RPC para el script de transformación en ThingsBoard.

A continuación, se procede a identificar la fuente del mensaje, en función del par clave:valor recibido. En este script, se ha desarrollado la lógica para dos tanques, pero debería revisarse y adaptarse según el número de ID de tanque que requiera la activación de la bomba remota. Luego, dependiendo del tanque solicitante, se debe establecer el idBomba y el estado correspondiente para la zona. Además, es necesario especificar el nuevo tipo de mensaje: “RPC_CALL_FROM_SERVER_TO_DEVICE”.

Por último, se debe construir el cuerpo del mensaje, que incluye un "método" que debe coincidir con la llamada RPC definida en el código del dispositivo a ser llamado, y unos "parámetros", que contienen los datos que se enviarán, en este caso, el ID de la bomba y su estado. Finalmente, el mensaje creado con el cuerpo, los metadatos y el tipo de mensaje fijado sale del nodo script de transformación.

Los últimos pasos en el camino del mensaje RPC son los siguientes: primero, se vuelve a cambiar el originador desde el asset al dispositivo (el vinculado al tanque solicitante). Luego, en función de la relación entre este dispositivo y el dispositivo objetivo (el que tiene la bomba a encender), se cambia nuevamente el originador del mensaje. Finalmente, se ejecuta el nodo de acción para realizar la solicitud RPC desde el servidor. De esta manera, el mensaje RPC se envía desde el propio dispositivo que tiene la bomba vinculada.

Como nota final, los rule chain “Bombeo” y “Almacenamiento” siguen una estrategia de diseño similar a la de “Almacenamiento y rebombeo”, con las simplificaciones al caso de cada una, considerando la ausencia de tanques en “Bombeo” y de bombas en “Almacenamiento”.

3.2.3 Diseño y construcción del prototipo para montaje

En esta sección se describirán las partes y piezas utilizadas para ensamblar el prototipo, que eventualmente podría ser instalado en Aluminé, en las zonas donde existan tanques y/o bombas. Además, se presentarán los elementos utilizados para realizar la simulación y mostrar los resultados.

- **ESP32 y placa de conexiones:**

El ESP32 se montará sobre pines Dupont soldados a una placa experimental PCB. Además, se añadirán borneras para la conexión a los puertos GPIO. Para alimentar la placa, se soldará el módulo MB102 para protoboard visualizado en la Figura 49. Este módulo admite una entrada

de tensión de entre 6.5 V y 12 V, y proporciona una salida de 3.3 V y 5 V (con control independiente mediante jumpers). Además, el conector USB puede usarse tanto como entrada (si no hay adaptador externo) como salida. La placa puede suministrar hasta 700 mA de corriente nominal.

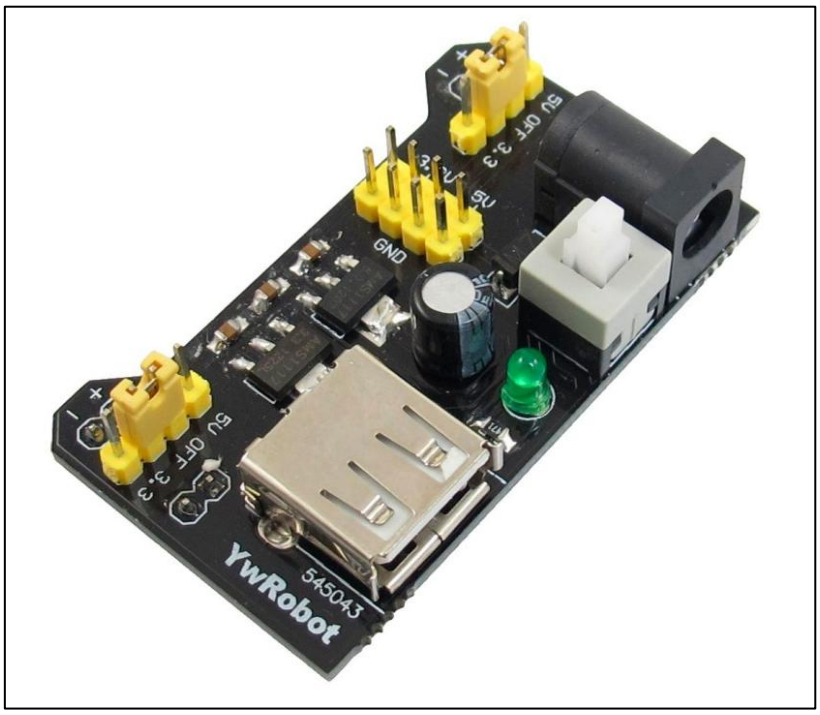


Figura 49: Placa de alimentación con entrada 6.5-12 VDC y salida 3.3 VDC y 5 VDC. (Fuente:Starware)

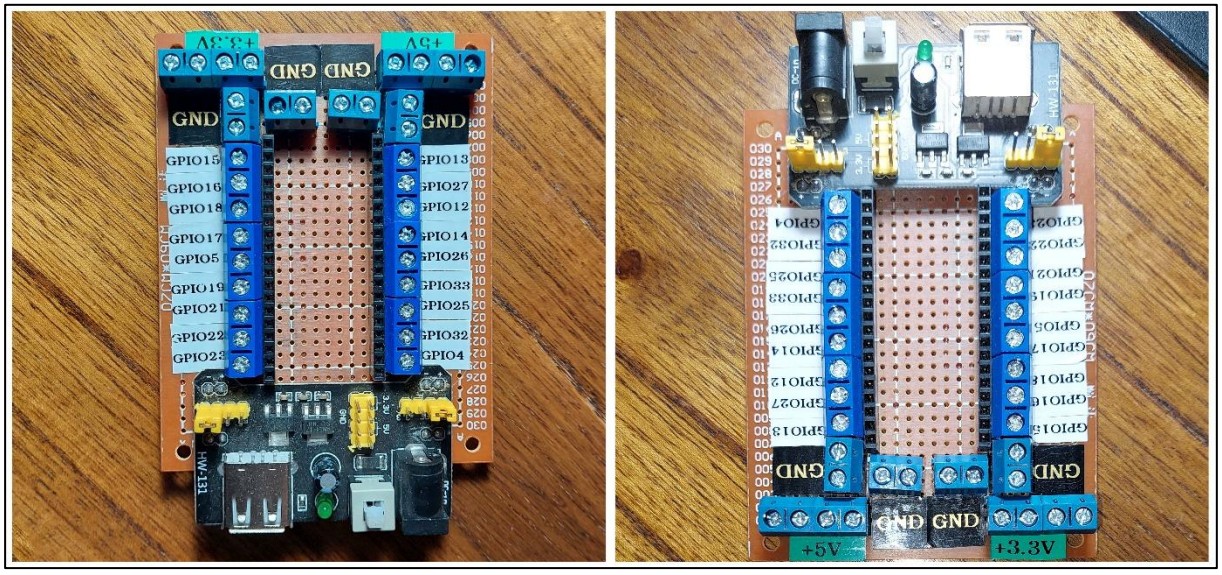


Figura 50: Placa de conexiones donde se monta el ESP32.

En la Figura 50 se puede observar el resultado de la placa prototipo, que se ubicará dentro de la carcasa plástica. Es importante aclarar que el ESP32 debe colocarse con el puerto USB

orientado hacia las borneras de GND, ya que la soldadura se ha realizado de manera que tanto la alimentación como los GPIO coincidan con esa disposición.

Aunque la cantidad de GPIO habilitados en el diseño puede ampliarse (más detalles en la conclusión de este PIP), el número de pines de entrada/salida es suficiente para cubrir dos tanques y tres bombas sin inconvenientes, además de considerar los pines destinados al LCD y al pulsador externo.

- **Carcasa:**

La carcasa donde se ubicará el automatismo será una caja estanca sencilla, como la que se muestra en la Figura 51. Sobre esta se realizarán un par de perforaciones y modificaciones. En primer lugar, será necesario hacer un agujero en la parte inferior para el paso de los cables de los sensores flotadores, de nivel de agua y de las bombas. Además, en el canto izquierdo se deberá realizar un pequeño orificio para la alimentación que energiza el automatismo. Finalmente, en la cara frontal se hará un orificio para ubicar el pulsador externo, así como un rectángulo con las dimensiones del display LCD.

Aunque esta carcasa requiere trabajo adicional, la ventaja es que todo es relativamente económico y fácil de ejecutar. Otra opción más prolija sería encargar las cajas con los orificios y el rectángulo para el LCD a medida, utilizando un servicio de impresión 3D. Aunque esta alternativa es la mejor opción en cuanto a acabado, su costo es superior.

Figura 51: Carcasa externa para alojar el ESP32. (Fuente: gadeelectricidad.com.ar)

La caja adquirida tiene unas dimensiones de 10x15 centímetros. Luego de trabajar sobre la misma, el resultado final es el visualizado en las Figuras 52 y 53.

Figura 52: Vista frontal (arriba) y lateral (abajo) de la carcasa externa modificada para alojar el ESP32.

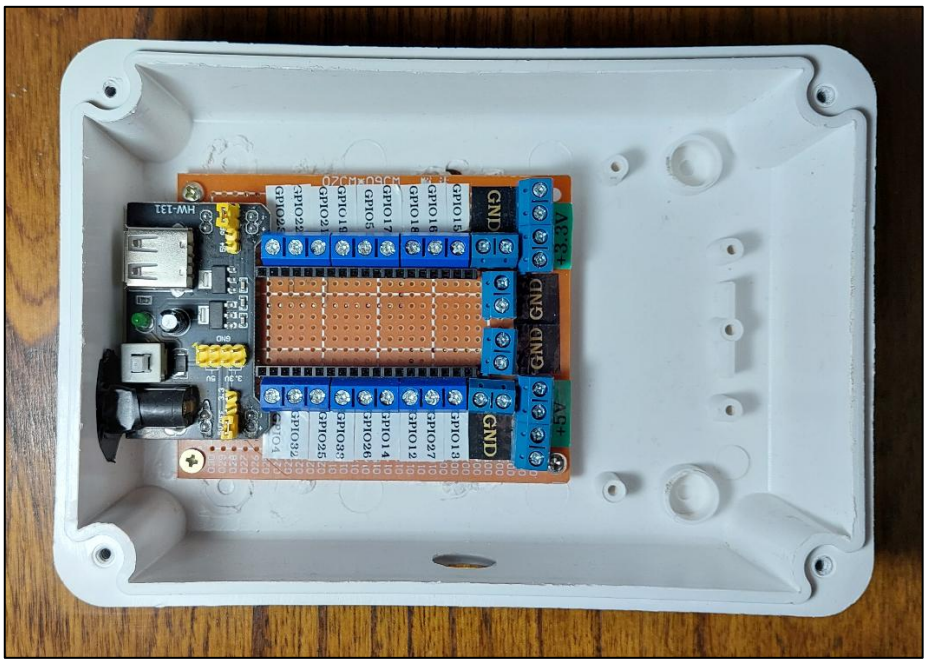


Figura 53: Vista interna de la ubicación de la placa de conexiones dentro de la carcasa.

- **Maqueta para realizar la simulación:**

Para realizar la simulación de funcionamiento y obtener los resultados, se utilizará un recipiente de pintura para simular un tanque, sobre el cual se colocarán los sensores flotadores, tal como se observa en la Figura 54. En la parte superior, se montará el sensor ultrasónico que mide el nivel, utilizando un soporte. Este sensor será soldado sobre una placa, como se ilustra en la Figura 55, tiene una reducción de tensión a 3.3 V sobre el pin "Echo" ya que es una entrada al ESP32 y es la tensión admisible por el dispositivo.

Figura 54: Recipiente de pintura para la simulación del tanque con los sensores flotadores ubicados.

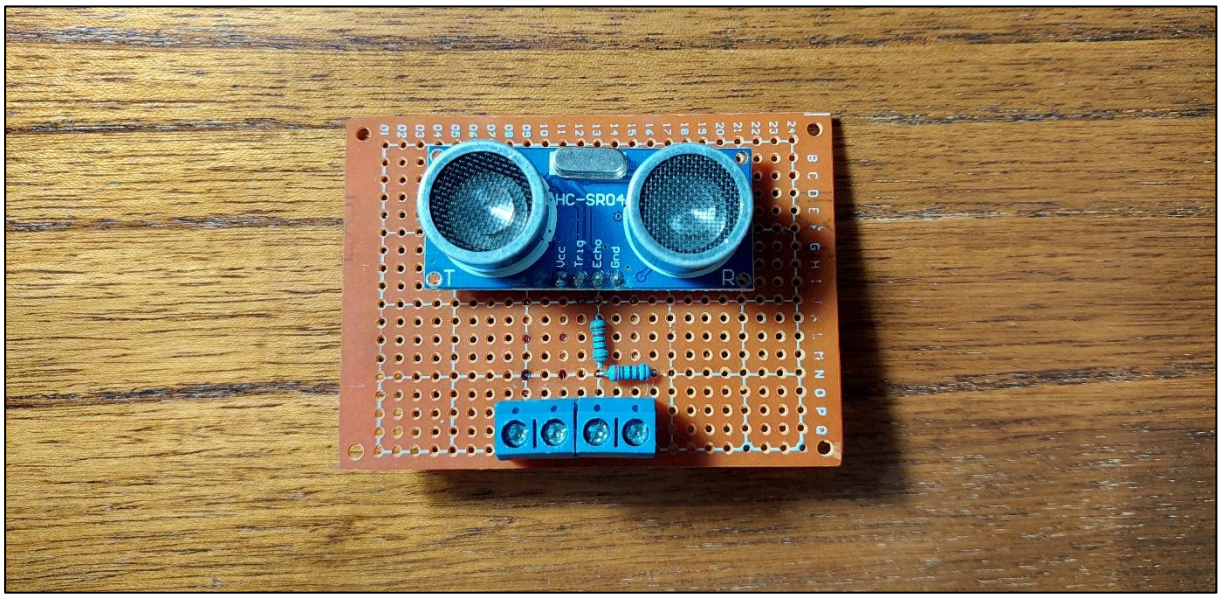


Figura 55: Sensor ultrasónico HC-SR04 soldado en una placa para fácil conexión.

Los datos de las dimensiones del tanque se cargarán en el ESP32. Además, se empleará un ESP8266 para controlar una bomba de 5 V visualizada en la Figura 56, replicando así el funcionamiento que podría ocurrir en Aluminé con el accionamiento de una bomba a través de la nube. En este caso, el ESP32 estará ubicado en la zona “Almacenamiento y rebombeo 1”, mientras que el ESP8266 que controla la bomba se situará en la zona “Bombeo 1”. Además, para simplificar la simulación, se consideró especificar en el ESP32 la vinculación con un único tanque sin bombas y para el ESP8266 la vinculación con una única bomba. En los resultados de la simulación, podrá verse en algunos widgets la existencia de tags “Tanque 2” o “Bomba 2”, esto es debido a la creación de esos tags en los dispositivos en el pasado y quedaron en la base de datos de los assets. Por último, la bomba de agua se vinculó mediante un microtubo hasta el tanque y se la ubico en otro recipiente con agua para permitir la carga.

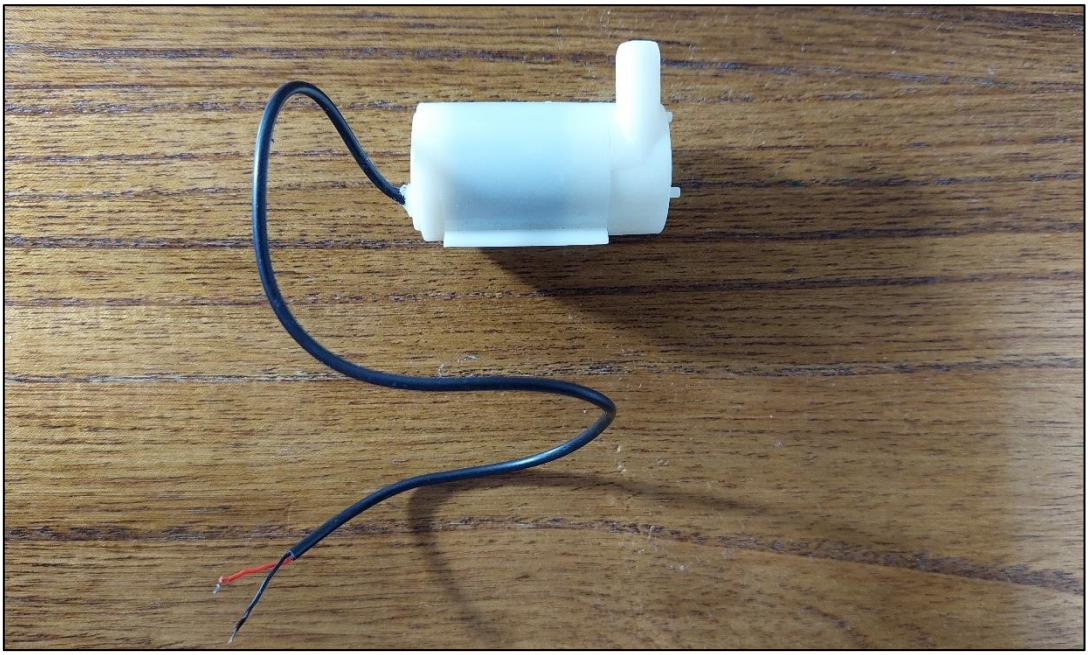


Figura 56: Bomba de agua utilizada para la prueba de funcionamiento.

3.3 Costos estimados para la eventual implementación en Aluminé

En esta última sección previa a visualizar los resultados de la simulación, se considerarán los costos globales que implicaría este proyecto para su eventual implementación en la localidad de Aluminé. Cabe remarcar que los costos son estimativos y podrían variar según la situación económica del país y el tipo de hardware adquirido.

En términos generales, para cada tanque deben contemplarse dos sensores flotadores y un sensor ultrasónico (con hardware adicional para garantizar inmunidad al ruido debido a la extensión en el terreno), mientras que para cada bomba se requiere una entrada de relé.

Para cada dispositivo, se debe considerar una caja estanca que incluya una pantalla LCD, un pulsador, una placa PCB, un módulo de alimentación, un cargador, pines Dupont, borneras y cables de conexión.

Además, es necesario contar con un módem o router 4G, cables de extensión en el terreno, el plan básico de ThingsBoard, y los planes de telefonía o del proveedor de servicios de Internet (por ejemplo, Starlink) para cada una de las seis zonas.

En Aluminé, la disposición de tanques y bombas es aproximadamente la observada en la Tabla 2:

Zona	Cantidad de tanques	Cantidad de bombas
Bombeo 1	-	4
Bombeo 2	-	3
Almacenamiento y rebombeo 1	8	3
Almacenamiento y rebombeo 2	2	2
Almacenamiento y rebombeo 3	2	1
Almacenamiento 1	1	-
Totales	13	13

Tabla 2: Disposición aproximada de bombas y tanques en Aluminé.

En la Tabla 3 se presenta la inversión inicial estimada necesaria para la implementación del sistema en Aluminé. Cabe mencionar que algunos costos asociados a las comunicaciones deben abonarse mensualmente para mantener los servicios; tal es el caso de los seis chips 4G de Movistar y del plan “Maker” del servicio ThingsBoard Cloud.

ITEM	CANTIDAD	DESCRIPCIÓN/FUNCIÓN	COSTO
Sensor flotador (acero inoxidable)	26	Censar cambios en niveles críticos de agua	$ 292.816,00
Sensor ultrasónico	13	Censar el porcentaje de llenado de los tanques	$ 32.500,00
Caja sensor ultrasónico (3D)	13	Carcasa protectora para el sensor ultrasónico	$ 114.400,00
Pack borneras 2 vías (10)	3	Borneras para sensor ultrasónico	$ 9.000,00
Placa PCB experimental 5x5 Cm	13	Placa para soldar borneras y ultrasónico	$ 41.600,00
NodeMCU ESP 8266	13	Dispositivo con interfaz UART para transmitir desde el sensor ultrasónico al ESP32 mediante RS485	$ 74.100,00
Modulo MAX 485	26	Permite conversión RS485-UART para transmitir datos a larga distancia	$ 69.420,00
SUBTOTAL TANQUES			**$ 633.836,00**

Relés SSR	13	Comandar la bobina del contactor de bomba correspondiente	$ 91.000,00
SUBTOTAL BOMBAS			**$ 91.000,00**
NodeMCU ESP32	10	Dispositivo de monitorización y control del sistema de distribución de agua potable	$ 110.000,00
Caja estanca 10x15x8Cm	10	Carcasa donde alojar el dispositivo con visualización local de información	$ 41.800,00
Placa PCB experimental 7x9 Cm	10	Placa donde se suelda el dispositivo y borneras para fácil conexión	$ 30.000,00
(5) Kits Pines hembra Dupont (1x40)	2	Pines para soldar en PCB y colocar el ESP32	$ 8.000,00
Pack cables macho-hembra Dupont (40)	1	Cables para conectar la pantalla LCD al ESP32	$ 4.300,00
Pack borneras 3 vías (30)	2	Borneras para simplicidad de conexión	$ 15.000,00
Pack borneras 2 vías (10)	8	Borneras para simplicidad de conexión	$ 24.000,00
Módulo de alimentación 6.5 V – 12 V	10	Módulo de adaptación para energizar el dispositivo y hardware necesario	$ 32.000,00
Cargador 12 V 2A	10	Fuente de energía para alimentar la caja de control y sensores	$ 56.000,00
Pantalla LCD	10	Pantalla de visualización local de información	$ 74.000,00
Pulsador externo	10	Pulsador para pasar por los menús en la visualización local	$ 50.000,00
Cable 18 AWG (100 metros)	15 (*)	Cable para extensión en terreno	$ 930.990,00
Cable UTP Cat 5e (60 metros)	2 (*)	Cable para comunicación por RS-485	$ 65.800,00
Caño corrugado 3/4″ (25 metros)	20 (*)	Protector para pasar el cableado por el terreno	$ 242.000,00
SUBTOTAL CAJA DE CONTROL			**$ 1.683.890,00**
Router/Modem 4G	6	Router para formar una red local en cada zona y habilitar conexión a internet	$ 813.516,00
Plan prepago Movistar (mensual)	6	Vínculo con la internet y la plataforma IoT	$ 30.000,00
Plan "Maker" plataforma IoT ThingsBoard (mensual)	1	Acceso a la monitorización completa del sistema y control remoto de bombas	$ 16.576,00
SUBTOTAL COMUNICACIONES			**$ 860.092,00**
TOTAL (Costo de inversión inicial)			**$ 3.268.818,00**
(*) La extensión de cable en el terreno puede ser inexacta.			

Tabla 3: Costos estimados para la implementación en Aluminé.

El uso de chips 4G representa la opción más económica, ya que los servicios de los proveedores de internet tradicionales suelen tener un costo más elevado. Si bien existen planes de bajo costo de Starlink, como el de $12.000 mensuales por 10GB de datos (luego del primer

mes), estos continúan siendo más costosos que el mantenimiento de un chip 4G prepago. Además, debe considerarse el costo adicional del hardware necesario para operar con Starlink.

El mantenimiento mensual de las comunicaciones asciende a $46.576,00, lo cual resulta muy accesible. No obstante, podrían surgir erogaciones puntuales en caso de fallas o desperfectos en sensores u otros dispositivos del sistema. Estos costos expresados en pesos argentinos fueron relevados al día 23/07/25.

4 Resultados

En la siguiente sección se mostrarán imágenes con los resultados observados localmente en la pantalla LCD, así como también a través de la plataforma IoT ThingsBoard durante las pruebas de funcionamiento.

4.1 Resultados locales

Como se mencionó anteriormente, el prototipo cuenta con un botón que permite desplazarse por los distintos menús en el LCD y visualizar en tiempo real la información que recaba el dispositivo. En este caso, en la carcasa se encuentra el ESP32, el cual está vinculado a un tanque y no tiene bombas asociadas.

Figura 57: Visualización del nivel de agua del tanque 1 en proceso de carga.

Figura 58: Visualización del estado de conexión exitoso al WiFi de la zona.

Las Figuras 57, 58 y 59 permiten visualizar el nivel de agua del tanque 1, así como el estado de conexión a la red WiFi y a la plataforma IoT. En el “Anexo B” se presentan otros menús que muestran distintas condiciones de funcionamiento en la pantalla LCD, como por ejemplo, cuando hay más de un tanque o cuando hay bombas vinculadas.

Figura 59: Visualización del estado de conexión exitoso a la plataforma IoT.

4.2 Resultados en la plataforma IoT

A continuación, se presentará los datos recabados en la plataforma IoT a partir del envió de información mediante WiFi.

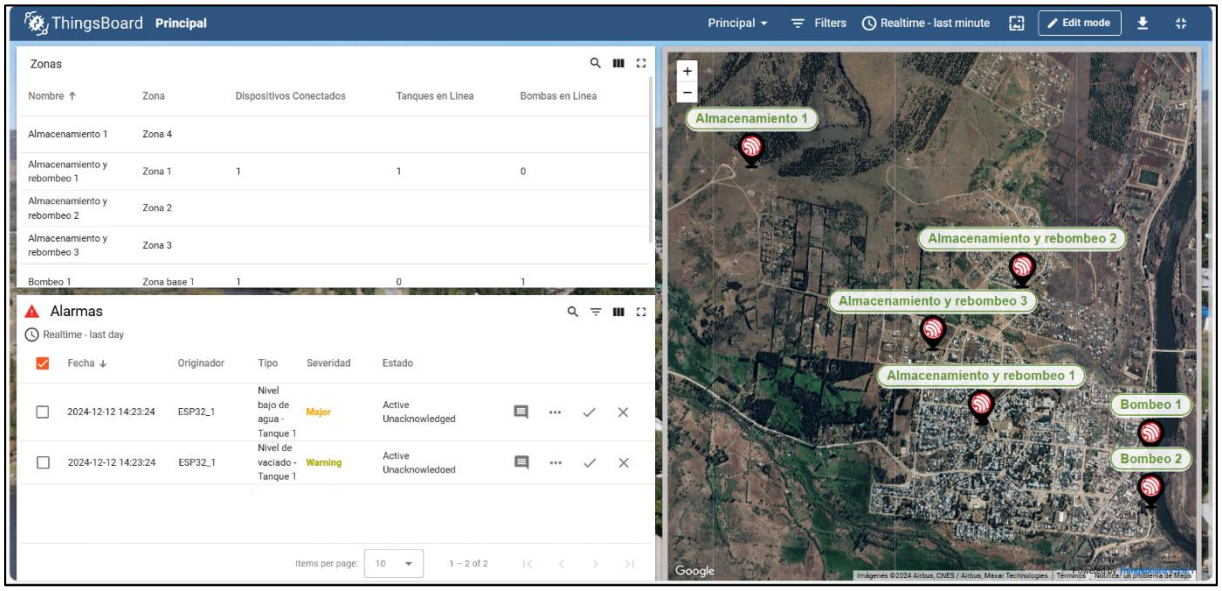


Figura 60: Menú principal del panel de control en el inicio de la simulación.

La Figura 60 muestra los dispositivos conectados ubicados en las zonas mencionadas con anterioridad. Además, puede observarse que el ESP32 ubicado en “Almacenamiento y rebombeo 1” tiene un tanque en línea y el ESP8266 ubicado en “Bombeo 1” tiene una bomba en línea. Tal número de bombas y tanques en línea se incrementaría en caso de existir otros dispositivos en dicha zona con otros tanques y bombas vinculados.

Bajo la tabla de entidades, se encuentra el cuadro de alarmas, en la cual se puede visualizar algunas de las alarmas que fueron creadas para el tanque 1. En especial, se muestra que el tanque está vacío (la bomba no comenzó la carga o el nivel está por debajo del sensor flotador inferior), el nivel de gravedad es warning porque paso menos de 1 minuto desde el recibo de telemetría. La otra alerta indica que el nivel de agua está bajo con severidad “mayor”, esto es porque el porcentaje de llenado está por debajo del 5%.

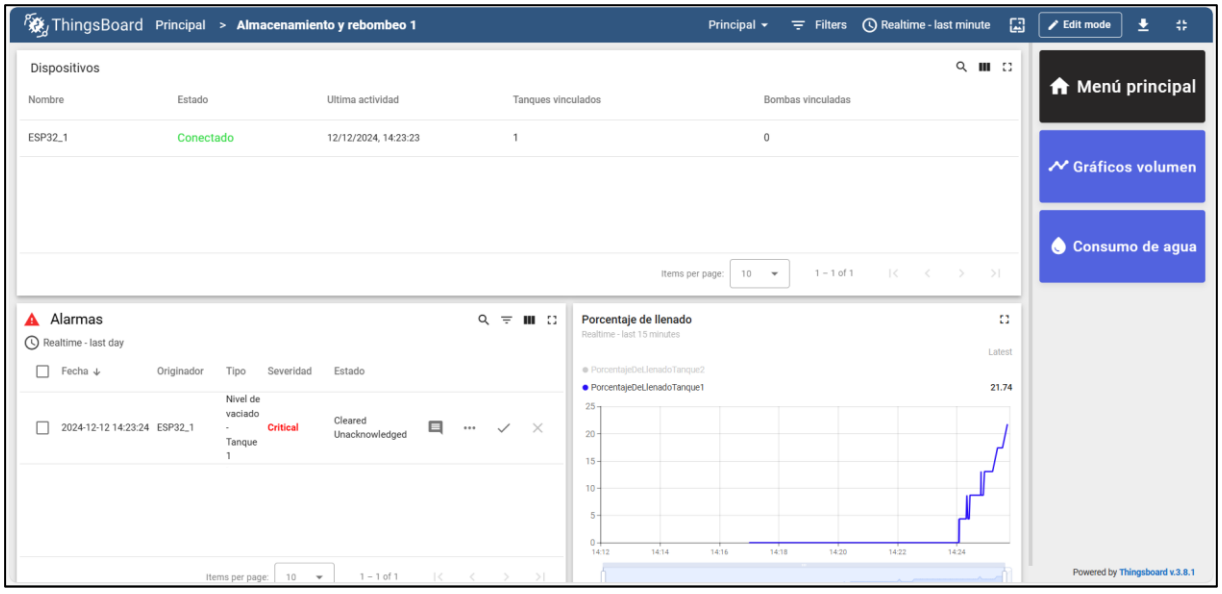


Figura 61: Estado de la zona "Almacenamiento y rebombeo 1" en el comienzo de la simulación.

En la Figura 61 se puede ver el estado de la zona de almacenamiento y rebombeo 1, indicando además del estado de los dispositivos en la tabla de entidades, el porcentaje de llenado de los tanques totales en la zona. Se observa que la alarma de vaciado paso a crítico, pero luego de que el nivel de agua supero el sensor inferior por el lapso de un minuto, paso a estado “cleared”, solo que no está reconocida y por eso todavía se puede observar en las alarmas.

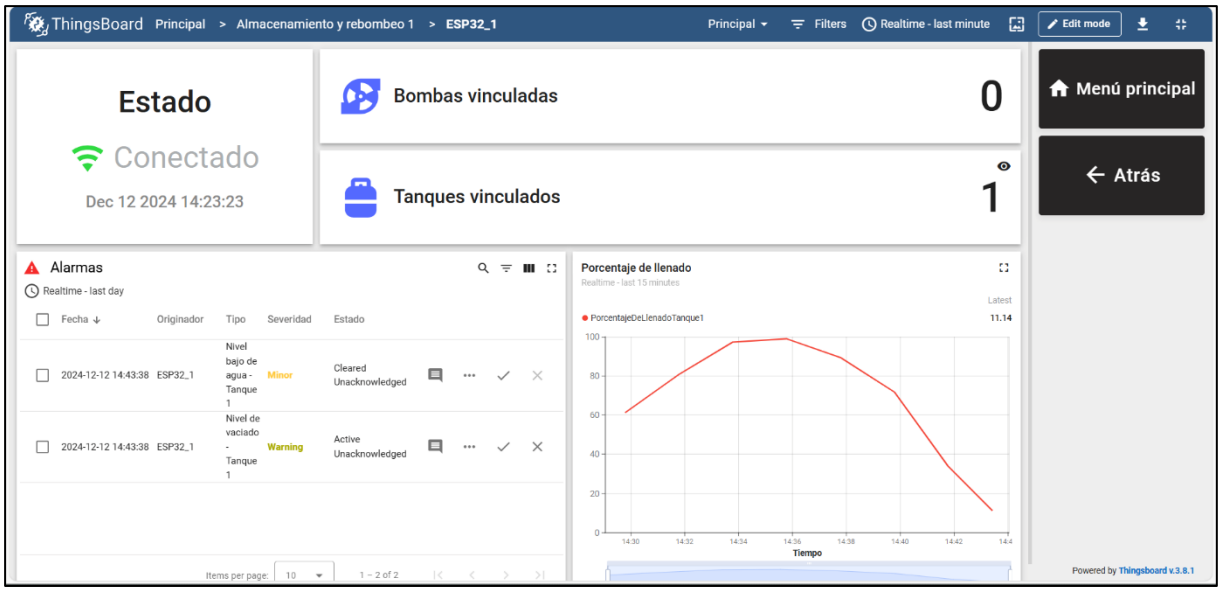


Figura 62: Cuadro de estado del ESP32 durante la simulación.

Las Figuras 62 y 63 presentan el estado de los dispositivos ubicados en “Almacenamiento y rebombeo 1” y en “Bombeo 1”. En cada caso se puede observar que están conectados y muestran características propias de cada caso. En el ESP32, se observa la curva que representa el porcentaje de llenado en tiempo real, mientras que en el ESP8266 no hay información dado que no hay tanques y además por ende tampoco hay alarmas.

Otro punto a observar es el “ojo” visible en cada caso al lado de “Bombas vinculadas” y “Tanques vinculados”. Como se mencionó anteriormente, este ojo solo es visible en el menú correspondiente solo si el dispositivo tiene vinculados tanques y/o bombas.

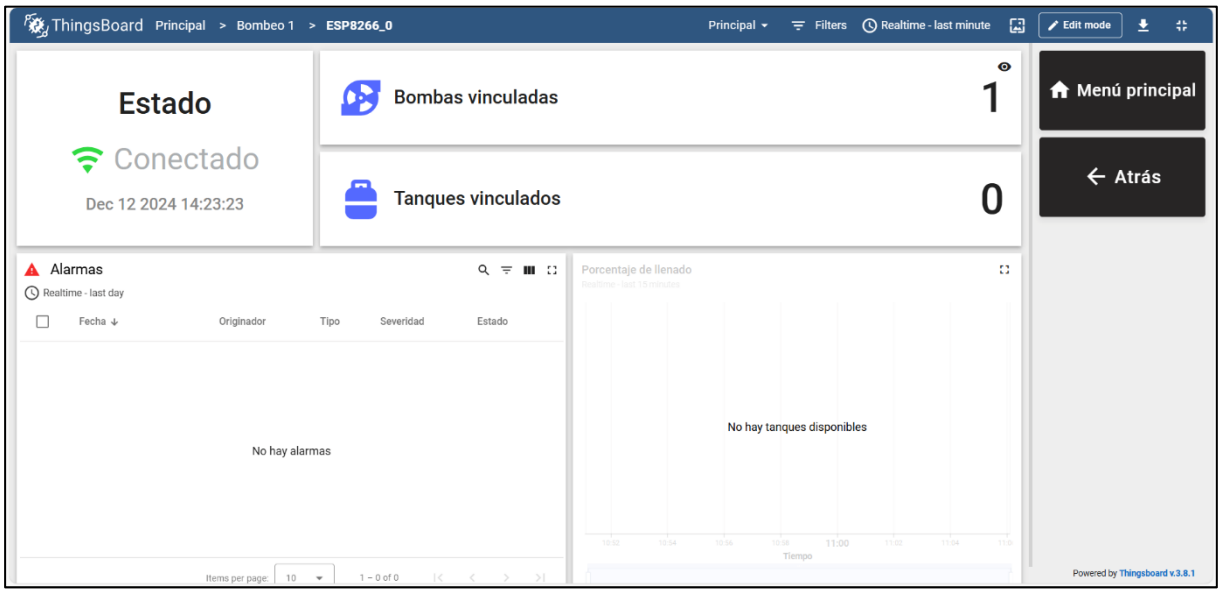


Figura 63: Cuadro de estado del ESP8266 al comienzo de la simulación.

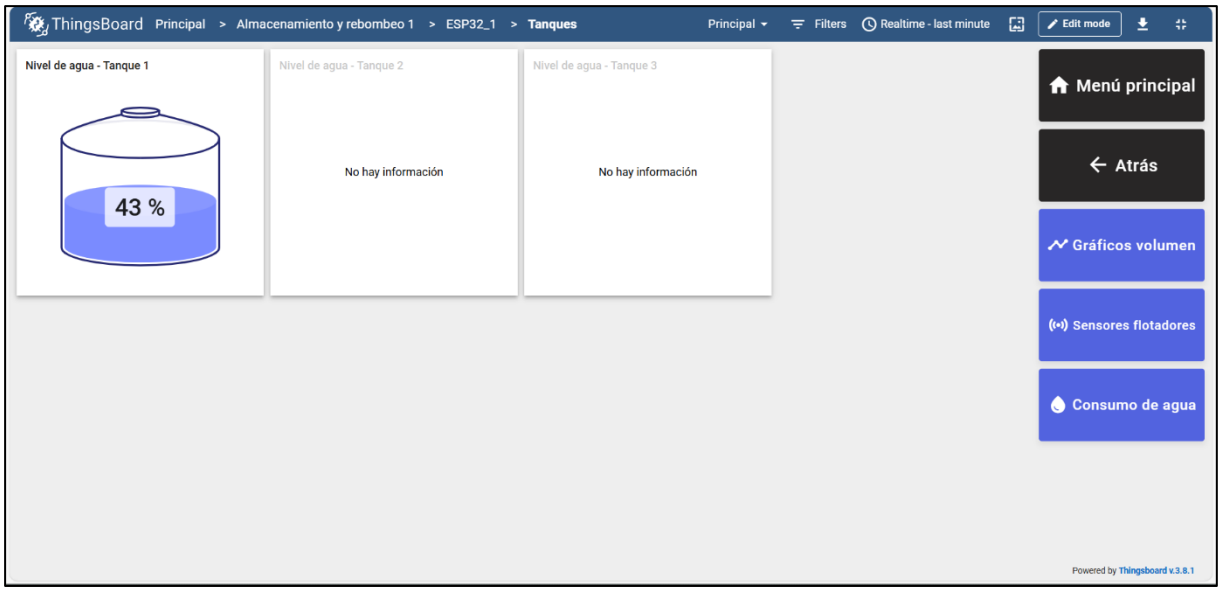


Figura 64: Cuadro de estado para los tanques vinculados al ESP32 de la zona "Almacenamiento y rebombeo 1".

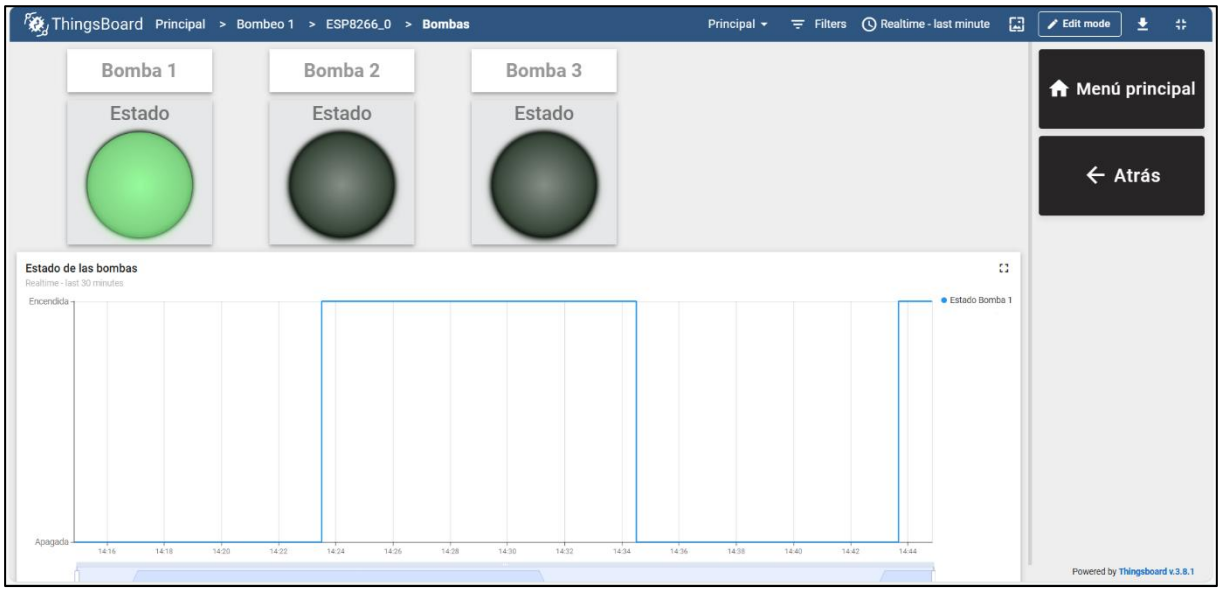


Figura 65: Cuadro de estado para las bombas vinculadas al ESP8266 de la zona "Bombeo 1".

Las Figuras 64 y 65 muestran los estados de los tanques y de las bombas para los respectivos dispositivos accediendo a través del "ojo". Como se puede ver en la Figura 64, solo se muestran los tanques que están vinculados al dispositivo en cuestión, en los no vinculados figura la leyenda "No hay información". Se debe recordar que los tanques no vinculados corresponden a los demás tanques que se encuentran en la zona pero que estarían vinculados a otro dispositivo.

De manera semejante ocurriría con el cuadro de estado de la bomba, con la desventaja (señalada anteriormente) que, en este caso, en los widgets de los leds no figurara "No hay

información". Para saber que bombas están vinculadas al dispositivo en cuestión, se deberá mirar el grafico del estado de abajo. En este caso, como el ESP8266 solo tiene una bomba, entonces, solo hay que observar el estado del primer led.

A continuación, la Figura 66 indica el estado de los sensores flotadores en el tanque, durante la prueba. Se ve con simpleza si el sensor flotador superior (grafico superior) está en situación crítica, lo que implica posibilidad de desborde y desactiva la bomba. Mientras que el sensor flotador inferior de manera análoga, se encuentra en estado normal cuando hay agua sobre él, es decir, se encuentra en situación crítica de activado de bomba cuando el nivel está por debajo del mismo.

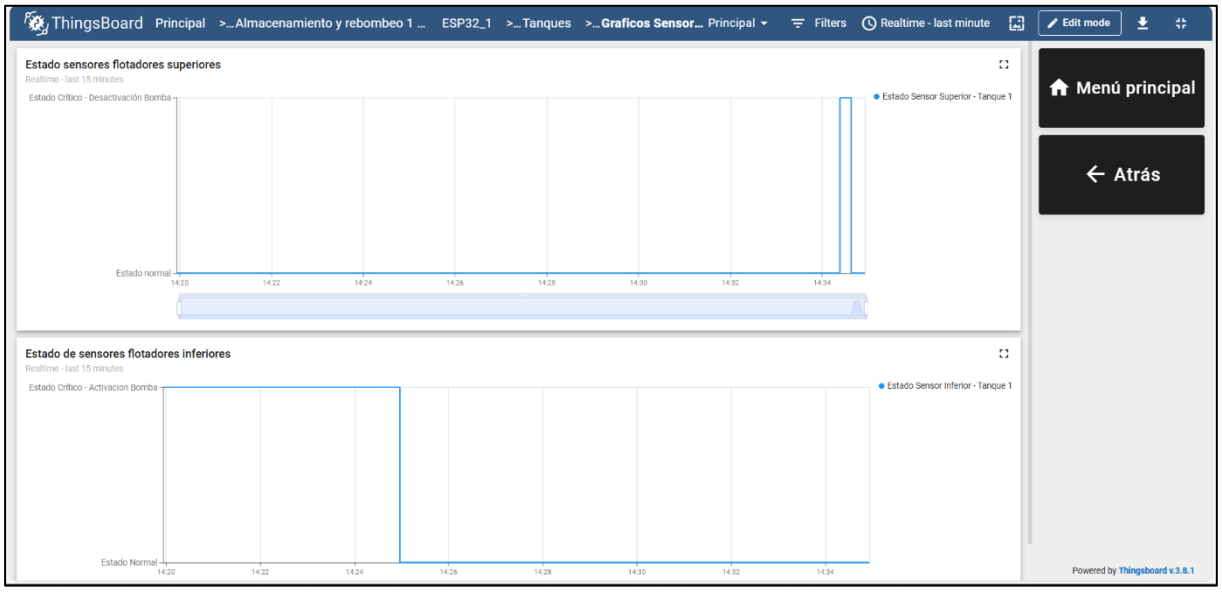


Figura 66: Estado de los sensores flotadores en el tanque durante la simulación.

Las Figuras 67 y 68 muestran los gráficos de consumo de agua y de volumen respectivamente.

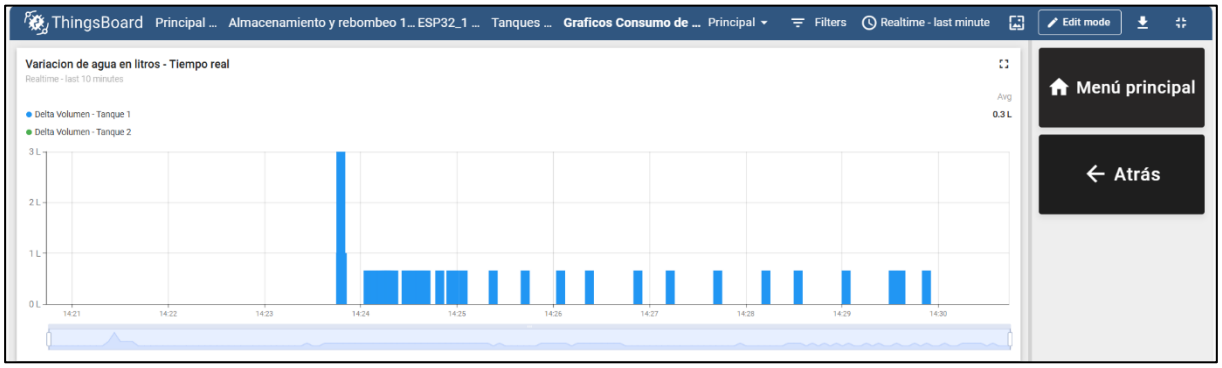


Figura 67: Variación de agua en litros entre muestras durante la simulación.

En la Figura 67 las primeras mediciones pueden ser anómalas por tapar parcialmente el sensor ultrasónico, luego, sin embargo, tiende a mantenerse constante el delta entre mediciones,

el cual es de 0.66 litros, el cual es el ritmo de carga dada por la bomba de 5 V. Se debe tener en cuenta que cada medición se envía cada 10 segundos, sin embargo, existen algunas inconsistencias en términos temporales que se mencionaran en la sección de las conclusiones.

La Figura 68, presenta la variación volumétrica de agua en el tanque útil (entre los sensores flotadores), puede observarse la variación del nivel en forma progresiva, teniendo en cuenta que, para la descarga, se hizo a "mano" por lo que hay variaciones más abruptas. Se puede ver que la bomba comienza a actuar antes de alcanzar los 0 litros, esto puede deberse a una medición incorrecta de la distancia desde el fondo del bote de pintura al punto de accionamiento del sensor flotador (del orden del centímetro), también puede contribuir el pequeño oleaje del agua al descargar el tanque, modificando parcialmente el flotador del sensor inferior.

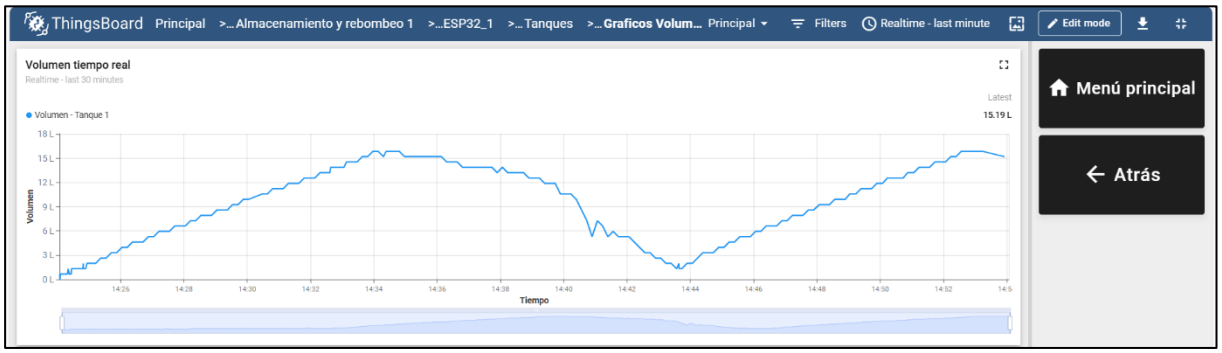


Figura 68: Variación volumétrica de agua del tanque durante la simulación.

5 Conclusiones

5.1 Conclusiones

En esta penúltima sección se presentarán las conclusiones del trabajo, así como las futuras mejoras y el trabajo pendiente para materializarlo en una solución práctica que beneficie a la población de Aluminé.

El trabajo realizado mejora sustancialmente la operación del sistema de abastecimiento de agua potable en Aluminé, al optimizar el funcionamiento de las bombas mediante el automatismo desarrollado. Gracias a este sistema, las bombas operan únicamente durante los períodos de tiempo asignados, lo que no solo permite un ahorro energético para Aluminé, sino que también contribuye a aumentar la vida útil de las bombas. A su vez, se pondría fin al uso inapropiado del agua del río Aluminé, evitando su vaciamiento y promoviendo la sostenibilidad ambiental. La opción de automatismo propuesta en este proyecto no solo es viable, como se demostró a lo largo del desarrollo, sino que también resulta considerablemente más económica en comparación con otras alternativas, como el uso de un PLC, que es común en entornos industriales. No obstante, es importante tener en cuenta que no se trata de un dispositivo "industrial", por lo que aspectos como la humedad, el polvo y las vibraciones externas podrían afectarlo. Además, se recomienda ubicarlo en lugares con temperaturas controladas, ya que, a largo plazo, se han reportado fallas debido a sobrecalentamiento. Dado que en Aluminé la temperatura suele ser baja durante la mayor parte del año, salvo en excepciones durante el verano, este factor no debería representar un problema significativo.

El prototipo montado en la caja estanca fue suficiente para este trabajo, aunque se deberían considerar algunas modificaciones que se mencionarán en la siguiente subsección. La maqueta utilizada para simular el tanque y la bomba fue útil para mostrar el funcionamiento y permitió obtener datos valiosos para el análisis. Sin embargo, se presentaron problemas relacionados con los gráficos de consumo de agua, particularmente en lo que respecta a los deltas. Si bien la variación del volumen de agua siempre se mantuvo constante (lo cual era esperado, ya que solo existe la carga de la bomba), el intervalo entre muestras no era constante. Tras un análisis más detallado, se observó que, en cada muestra (configurada para el envío de datos cada 10 segundos), siempre se registraba un valor de delta en el gráfico (como correspondía). El problema surgió cuando, en ocasiones, el delta era cero, mientras que en otras era de 0,66 L. En los momentos en los que el delta era cero, los valores del volumen del tanque en muestras consecutivas (que se utilizan para calcular el delta) eran iguales. Por lo tanto, el problema no

estaba en el cálculo dentro del "rule chain" de la plataforma, sino en el envío de la información hacia la misma. En general, el valor de delta actualizado a 0,66 L se observaba cada dos muestras consecutivas, es decir, cada 20 segundos, y con menor frecuencia cada 30 segundos. Es probable que estas variaciones se deban a limitaciones inherentes a la resolución del sensor ultrasónico. Cabe señalar que, en teoría, la capacidad del sensor para detectar cambios en función de las variaciones en el nivel de agua es del orden de las tres décimas de centímetro; sin embargo, en las prácticas efectuadas, la resolución observada se aproxima más al centímetro. Dado que el diámetro del recipiente de pintura es de 29 cm, el delta de 0,66 L se lograría al elevarse el nivel de agua en 1 cm. Además, según el ritmo de llenado observado en los gráficos, se estima que los 0,66 L se obtienen aproximadamente después de 25 segundos. Esto podría explicar la falta de detección de variaciones en el volumen de agua del tanque entre muestras consecutivas.

Otro aspecto relevante es la omisión del modo "sleep" en la programación del ESP32. Inicialmente, el proyecto contemplaba la posibilidad de ubicar el dispositivo en zonas muy remotas, donde no hay acceso a energía eléctrica, y operarlo mediante baterías. La idea era que el dispositivo permaneciera en modo de hibernación la mayor parte del tiempo y solo "despertara" para enviar datos a la plataforma, con intervalos que podrían ser más largos que los de otros dispositivos. Sin embargo, este modo fue desactivado en la programación por varias razones. En primer lugar, causaba problemas con el envío de datos a la plataforma. Existen funciones dentro del bucle principal del programa que mantienen la conexión con el servidor de ThingsBoard, pero, al despertar, la conexión no siempre se restablecía adecuadamente. En segundo lugar, en las zonas donde actualmente se encuentran los tanques y/o bombas hay acceso a energía eléctrica, lo que hacía innecesario el uso de esta función. En tercer lugar, un prototipo con la configuración actual (incluyendo los sensores láser que podrían colocarse en los tanques para medir el nivel de agua) y la posibilidad de enviar datos mediante GSM o 4G genera un consumo energético muy elevado para una batería. En este escenario, sería necesario recortar drásticamente las funcionalidades actuales del prototipo, como la visualización local, la frecuencia de envío de datos y la tecnología de comunicaciones utilizada para transmitir la información.

En resumen, el trabajo desarrollado ha logrado cumplir con los objetivos propuestos en el plan de trabajo. Este proyecto sienta las bases para la automatización del sistema de abastecimiento y distribución de agua potable en la localidad de Aluminé mediante una solución basada en IoT. Permite fomentar un uso más eficiente de la energía eléctrica, utilizando las

bombas lo justo y necesario, promoviendo la eficiencia energética. Asimismo, esta automatización mejora significativamente la gestión del recurso hídrico, evitando derrames innecesarios y favoreciendo condiciones para una mayor sostenibilidad ambiental. Finalmente, se concretó la construcción de una maqueta a escala que modela el sistema diseñado, junto con el desarrollo del panel de control en la plataforma IoT, el cual permite la monitorización integral del sistema. Esto representa un paso previo fundamental para su eventual implementación en las locaciones reales de Aluminé.

Como nota final, se debe mencionar que durante la elaboración de este informe se utilizó de manera responsable inteligencia artificial únicamente como herramienta de corrección de textos en términos de gramática, sintaxis, semántica y fluidez.

5.2 Modificaciones y trabajos futuros

Existen diversas mejoras que se pueden implementar en este proyecto. En cuanto a la plataforma, sería posible elaborar un widget para obtener información más detallada sobre el estado de las bombas, específicamente sobre el LED que indica su funcionamiento. Esto requeriría conocimientos de programación, pero la plataforma cuenta con una sección que permite editar o crear nuevos widgets, lo que lo convierte en una opción viable.

Otro aspecto a considerar es que, para implementar el prototipo en las zonas de Aluminé, es necesario utilizar los sensores láser mencionados en este informe si se desea obtener mayor precisión en las mediciones, especialmente en tanques cuya altura supere los 5 metros. Esto implicaría considerar la diferencia en el consumo de corriente entre los dos tipos de sensores (20 mA frente a 180–500 mA), por lo cual se debe planificar adecuadamente su alimentación.

Los sensores LiDAR TF Mini Plus vienen con cables de 24 AWG, una posible solución es extender corriente alterna (CA) hasta cada tanque y utilizar un cargador dedicado por tanque para alimentar cada sensor. Otra opción es utilizar un conjunto de convertidores: un step-up a la salida de la caja estanca para elevar la tensión y un step-down en cada tanque para reducirla al nivel requerido por el sensor, compensando así la caída de tensión por distancia.

Si se opta por la segunda opción, se debe garantizar que la fuente de alimentación proporcione suficiente corriente para todos los dispositivos. Con la configuración actual, esto no sería posible, ya que la placa de alimentación utilizada solo puede suministrar un máximo de 700 mA. Además, hay que considerar el consumo del ESP32 (picos de 380 mA), así como

el de posibles bobinas de electroimanes para los relés (si se vinculan bombas y no se usan SSR). Esto implicaría realizar otra perforación en la caja estanca para colocar otro cargador que alimente los sensores láser con 5 V y suficiente amperaje (aproximadamente 2 A para 4 sensores). O bien, reemplazar la placa de alimentación utilizada en este proyecto por una única fuente de 5 V con una capacidad de entre 2 A y 3 A (según la cantidad de sensores), incorporando además un circuito que regule la tensión a 3.3 V, necesario para el pulsador y los sensores conectados a las entradas del ESP32. En cualquier caso, posteriormente, se deberá colocar un conversor step-up para elevar la tensión y transmitir corriente continua (CC) hasta el otro extremo donde un convertidor step-down la reduciría al nivel adecuado para cada sensor.

Si se opta por usar los sensores ultrasónicos empleados en las pruebas de este trabajo, no habría mayores inconvenientes para extender cables en un terreno de 30 a 50 metros (distancia estimada desde la fuente de energía hasta el tanque más alejado) para la alimentación de los sensores. Se debe mencionar que el sensor HC-SR04, utilizado en este trabajo, funcionó correctamente en pruebas a corta distancia, e incluso se hizo una prueba en un corto intervalo de tiempo con una extensión de 27 metros utilizando cable UTP categoría 6, con resultados favorables. No obstante, en pruebas reales en campo, aunque su bajo consumo energético (15 mA) no representa un gran problema, el ruido electromagnético puede degradar la calidad de la señal, generando perdida de datos o mediciones erróneas. Para aumentar la confiabilidad, pueden considerarse dos soluciones. La primera consiste en utilizar un microcontrolador de prestaciones básicas con interfaz UART (una opción posible es el ESP8266), ubicado junto al sensor para recibir directamente los datos. Luego, mediante una placa transceptora UART-RS485, se transmite la señal al ESP32, donde otra placa transceptora la convierte nuevamente a UART. La señal UART en nivel TTL puede ser conectada a una de las tres interfaces UART por hardware independientes que posee el ESP32. La segunda opción es adquirir un sensor ultrasónico diferente, como el JSN-SR04T, el cual permite transmitir directamente por UART en algunos de sus modos. En este caso, no sería necesario contar con un microcontrolador junto al sensor, y bastaría con las placas transceptoras UART-RS485. Además, este sensor está preparado para la intemperie, tiene un mayor alcance (hasta 600 cm), distintos modos de operación y mantiene un consumo de corriente reducido; sin embargo, su costo es aproximadamente diez veces superior al del sensor ultrasónico HC-SR04 utilizado en este PIP.

En cualquier caso, ya sea con sensores ultrasónicos o con sensores láser, será necesario adquirir un par de placas transceptoras UART-RS485 (por ejemplo, modelo MAX485) por cada

sensor, con el fin de transmitir los datos a larga distancia mediante RS-485 y minimizar el ruido electromagnético inducido.

Asimismo, podría contemplarse una solución mixta que combine ambos modelos de sensores ultrasónicos, así como sensores láser, según las necesidades derivadas de las dimensiones físicas de cada tanque.

Para integrar estos circuitos y facilitar las conexiones dentro de la caja estanca, sería recomendable utilizar una caja estanca más grande y soldar la nueva placa de conexiones en una PCB de mayor tamaño. Esto también permitiría soldar más pines, en caso de ser necesario.

Por último, si la temperatura fuera un problema, y se deseara añadir un ventilador a la caja para actuar como extractor de aire caliente, se podría considerar uno de 5 V (como los que se usan en impresoras). Si el ventilador es de 12 V, se debería prever que la entrada a la caja estanca sea de 12 V, utilizando dos circuitos reguladores de tensión: uno para 5 V y otro para 3.3 V. Este ventilador podría ser controlado por el ESP32, de manera que funcione solo cuando la temperatura de la caja supere un umbral específico, utilizando una sonda de temperatura.

6 Bibliografía

Congresos, conferencias y seminarios:

[1] Romeo, M. (2017). Webinar IOT.

[2] Caballero, M.A. (2021). Taller IOT | Lora LoraWan, ESP32 y MQTT.

Tesis o disertaciones:

[3] Morcillo, L. (2021). Sistema domótico hogareño: Tomacorriente inteligente para la gestión del consumo energético, Tesis de grado, Universidad Nacional del Comahue.

Informes, reportes y documentos técnicos:

[4] Grupo Comunicaciones y Eficiencia Energética. Universidad nacional del Comahue. (2020). Implementación de una red de Internet de las Cosas en la ciudad de Neuquén.

[5] Grupo Comunicaciones y Eficiencia Energética. Universidad nacional del Comahue. (2023). PROGRAMA DE EFICIENCIA ENERGÉTICA EN GOBIERNOS LOCALES - ETAPA 1.

[6] Elemon. (2018). Presentación EESA-IOT 5.0.

[7] Microchip. (2019). LoRa basics.

Fuentes de internet:

[8] https://iwaponline.com/wst/article/82/12/2691/76088/Making-urban-water-smart-the-SMART-WATER-solution

[9] https://www.mdpi.com/2073-4441/13/13/1729

[10] https://electropeak.com/learn/full-guide-to-esp32-pinout-reference-what-gpio-pins-should-we-use

[11] https://docs.arduino.cc/language-reference

[12] https://docs.espressif.com/projects/arduino-esp32/en/latest/index.html

[13] https://www.espressif.com/en/products/modules/esp32

[14] https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[15] https://www.luisllamas.es/que-es-soc-y-som

[16] https://programarfacil.com/esp8266/esp32

[17] https://joy-it.net/files/files/Produkte/SBC-NodeMCU-ESP32/SBC-NodeMCU-ESP32_Datasheet_2023-09-13.pdf

[18] https://aprendiendoarduino.wordpress.com/category/conectividad-iot

[19] https://freeeway.com/es/que-es-una-plataforma-iot-cual-es-la-arquitectura-de-una-plataforma-iot

[20] https://thingsboard.io/docs

[21] https://thingsboard.io/docs/reference

[22] https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04

[23] https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[24] https://www.machinemfg.com/es/working-principle-of-laser-displacement-sensor

[25] https://www.tejjy.com/lidar-vs-laser

[26] https://es.wikipedia.org/wiki/LiDAR

[27] https://www.dfrobot.com/product-1915.html

[28] https://www.makerguides.com/tfmini-plus-distance-sensor-uart-with-arduino

[29] https://www.burak.in/blog/hydrostatic-level-measurement-types-advantages-applications

[30] https://www.bloginstrumentacion.com/productos/caudal/qu-cmo-funciona-interruptor-flotador

[31] https://www.mokolora.com/lora-and-wireless-technologies

[32] https://www.nfc-tag-shop.de/info/en/knowledge/nfc-in-smartphones/does-nfc-drain-battery/

[33] https://novelbits.io/ble-power-consumption-optimization/

[34] https://www.researchgate.net/publication/318054538_Comparative_Study_of_Communication_Interfaces_for_Sensors_and_Actuators_in_the_Cloud_of_Internet_of_Things

[35] https://trolinkiot.com/blog/2556.html

[36] https://deepbluembedded.com/esp32-sleep-modes-power-consumption/#esp32-active-mode

[37] https://www.emnify.com/blog/iot-connectivity

[38] https://geekflare.com/es/mqtt-vs-coap-vs-http

[39] https://www.dusuniot.com/es/blog/iot-based-water-quality-monitoring

[40] https://www.catsensors.com/es/catsensors/catnews/control-del-nivel-de-rios-y-rieras-con-sensores-ultrasonicos-lorawan

[41] https://flujometros-caudalimetros.com/caudalimetro/transductor-de-caudal

[42] https://www.captahydro.com/blog/monitoreo-de-aguas-y-telemetria-en-tiempo-real-soluciones-para-un-control-eficiente

7 Anexos

7.1 Anexo A: Hojas de especificación de dispositivos

7.1.1 NODE MCU ESP32:

NODE MCU ESP32

Microcontroller Development Board

With the NodeMCU-ESP32, comfortable prototyping is possible with simple programming via Lua script or the Arduino IDE and the breadboard-compatible design. This board has 2.4 GHz dual-mode Wifi and a BT wireless connection. In addition, a 512 KB SRAM and a 4MB memory are integrated on the microcontroller development board. The board has 21 pins for interface connection, including I2C, SPI, UART, DAC and ADC.

MAIN FEATURES	
Model	NodeMCU ESP32
Special features	Wifi and BT
Processor	Tensilica LX6 Dual-Core
Clock frequency	240 MHz
SRAM	512 kB
Memory	4 MB
Wifi standard	802.11 b/g/n - 2.4 GHz
Operating voltage	3.3 V; 5 V via micro-USB
Logic level	3.3 V
Max. current draw per GPIO	40 mA
Interfaces	I2C, SPI, UART, DAC, ADC

FURTHER SPECIFICATIONS	
Operating temperature	-40° C to 125° C
Dimensions	48 x 26 x 11.5 mm
Items delivered	NodeMCU ESP32

FURTHER DETAILS	
Article No.	SBC-NodeMCU-ESP32
EAN:	4250236816104
Customs Tariff No.	8473302000

Published: 13.09.2023

www.joy-it.net
SIMAC Electronics GmbH,
Pascalstr. 8, 47506 Neukirchen-Vluyn

Figura 69: Caratula principal de la hoja de especificaciones resumida para el ESP32.

7.1.2 MB102 Fuente de alimentación para protoboard 3.3V/5V

Handson Technology

Datasheet

MB102 Breadboard 3.3V/5V Power Supply

A 3.3V and 5V Breadboard Power Supply Module with series diode, polarity reversal protection. The module can take 6.5V to 12V input and can produce 3.3V and +5V. This is a must have power supply module for experimenters who have to test/prototype electronic circuits on breadboard or perforated/veroboards.

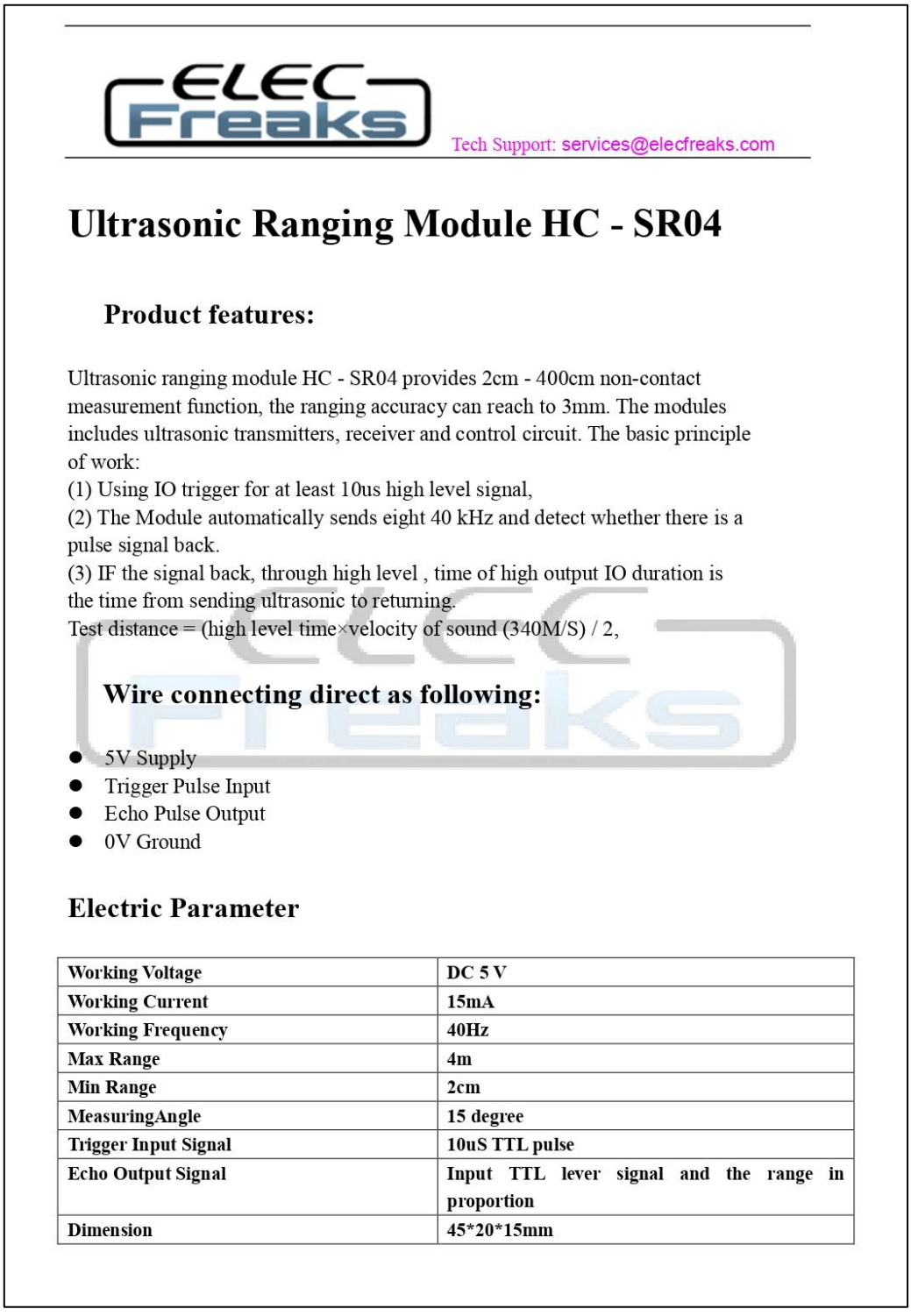
Brief Data:

- Plug directly to MB102 Standard breadboard.
- Input voltage: 6.5-12 V (DC) or 5V USB power supply.
- Output voltage: 3.3V and 5V can switch over.
- Maximum output current: <700 mA.
- External Input voltage ON/OFF switch.
- Independent control of upper and Lower Bread Board Power Rails. Can switch over to 0V, 3.3V, 5V using jumpers on any rail.
- On-board two groups of 3.3V, 5V DC output plug pin, convenient external lead use.
- USB device connector onboard for power output to external device.
- Size: 5.3cm x 3.5cm.

1 www.handsontec.com

Figura 70: Caratula principal de la hoja de especificaciones para el módulo de alimentación MB102.

7.1.3 Sensor ultrasónico HC-SR04



ELEC Freaks

Tech Support: services@elecfreaks.com

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

Electric Parameter

Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Figura 71: Caratula principal de la hoja de especificaciones para el sensor ultrasónico HC-SR04.

7.2 Anexo B: Capturas extra de visualización local del proyecto

Figura 72: Visualización local del nivel de agua en el tanque 1.

Figura 73: Visualización local del nivel de agua en el tanque 2.

Figura 74: Visualización local del estado de la bomba 1.

Figura 75: Visualización local del estado de la bomba 2.

Figura 76: Visualización del estado de conexión al WiFi de la zona

Figura 77: Visualización del estado de conexión sin éxito al WiFi de la zona.

Figura 78: Visualización del estado de conexión a la nube IoT.

Figura 79: Visualización del estado de conexión sin éxito a la nube IoT.

7.3 Anexo C: Funciones completas del programa cargado en el ESP32.

7.3.1 Función: interrupcionPulsador

```
115  // Funcion interrupcion para el pulsador externo
116  void IRAM_ATTR interrupcionPulsador() {
117   eliminarRebotePulsador();
118  if (lectura == LOW) {
119      return;
120    }
121    if (Tanque >= numeroMaxTanques) {
122      if (Bomba >= numeroMaxBombas) {
123        if (finBomba == 1) {banderaInterrupcionWifi = 1;}
124        finBomba = 1;
125      } else {
126        Bomba++;
127        tiempo = millis();
128      }
129    } else {
130      Tanque++;
131      tiempo = millis();
132    }
133    banderaInterrupcion = 1;
134  }
```

Figura 80: Código completo para la función de interrupción por pulsador externo.

7.3.2 Función: interrupcionSensor

```
136  #if numeromaxtanques > 0
137    // Funcion interrupcion para los sensores de nivel flotador
138    void IRAM_ATTR interrupcionSensor() {
139      for (int j = 1; j <= numeroMaxTanques; j++) {
140        if (digitalRead(sensorNivelBajo[j - 1]) == HIGH) {
141          #if numeromaxbombas > 0
142            encendidoLocalBombas(j);
143          #endif
144          checkTimerDatosOk = true;
145          return;
146        }
147        if (digitalRead(sensorNivelAlto[j - 1]) == HIGH) {
148          #if numeromaxbombas > 0
149            apagadoLocalBombas(j);
150          #endif
151          checkTimerDatosOk = true;
152          return;
153        }
154      }
155    }
156  #endif
```

Figura 81: Código completo para la función interrupción por los sensores flotadores.

7.3.3 Función: eliminarRebotePulsador

```
158 // Funcion para evitar el bouncing (rebote indeterminado entre 0 y 1) del
    pulsador externo
159 void eliminarRebotePulsador() {
160   int retardoAntirebote = 40;
161   int ultimoEstadoPulsador = LOW;
162   while (1) {
163     lectura = digitalRead(pinPulsadorExt);
164     if (lectura != ultimoEstadoPulsador) {
165       tiempoUltimoRebote = millis();
166     }
167     if ((millis() - tiempoUltimoRebote) > retardoAntirebote) {
168       if (lectura == HIGH) {
169         pulsador = 1;
170         return;
171       } else {
172         return;
173       }
174     }
175     ultimoEstadoPulsador = lectura;
176   }
177 }
```

Figura 82: Código completo para la función que evita el efecto rebote en el pulsador externo.

7.3.4 Función: encendidoLocalBombas

```
179 #if numeromaxbombas > 0
180 // Funcion encendido de bombas locales
181 void encendidoLocalBombas(int j) {
182   if (j <= numeroMaxBombas - cantidadBombasIndependientes) {
183     digitalWrite(pinRele[j-1], LOW);
184     cargaBombas[j - 1] = true;
185   }
186 }
```

Figura 83: Código completo para la función que enciende las bombas locales al dispositivo.

7.3.5 Función: apagadoLocalBombas

```
188 // Funcion apagado de bombas locales
189 void apagadoLocalBombas(int j) {
190   if (j <= numeroMaxBombas - cantidadBombasIndependientes) {
191     digitalWrite(pinRele[j-1], HIGH);
192     cargaBombas[j - 1] = false;
193   }
194 }
195 #endif
```

Figura 84: Código completo para la función que apaga las bombas locales al dispositivo.

7.3.6 Función: retardoDeTiempo

```
197  // Funcion Delay (retardo) en milisegundos
198  void retardoDeTiempo(int tiempoEsperadoMs) {
199  unsigned long Tiempo = 0;
200    Tiempo = millis();
201    while (millis() - Tiempo < tiempoEsperadoMs) {
202      if (banderaInterrupcion == 1) {
203        banderaInterrupcion = 0;
204        return;
205      }
206    }
207  }
```

Figura 85: Código completo de la función retardo de tiempo (milisegundos).

7.3.7 Función: retornoUltrasonico

```
209  // Funcion retorno en CM de distancia a objeto (por ultrasonico)
210  int retornoUltrasonico(int Trigger, int Echo) {
211    int duracion;
212    digitalWrite(Trigger, LOW);
213    retardoDeTiempo(1);
214    digitalWrite(Trigger, HIGH);
215    retardoDeTiempo(1);
216    digitalWrite(Trigger, LOW);
217    duracion = pulseIn(Echo, HIGH);
218    distanciaCm = round(duracion / conversionVelocidadSonido);
219    return distanciaCm;
220  }
```

Figura 86: Código completo de la función para retornar la distancia en centímetros desde el sensor ultrasónico.

7.3.8 Función: calculoNivelTanque

```
#if numeromaxtanques > 0
  // Funcion de calculo del porcentaje y del volumen de agua del tanque
  void calculoNivelTanque(int Tanque) {
    if (numeroMaxTanques == 0) { return; }
    float distanciaEfectiva = alturaMaximaTanques[Tanque - 1] -
    distanciaSensorInferiorDPiso[Tanque - 1] -
    distanciaSensorSuperiorDTecho[Tanque - 1];
    retornoUltrasonico(pinTrigTanques[Tanque - 1], pinEchoTanques[Tanque -
    1]);
    float distanciaLlenado = alturaMaximaTanques[Tanque - 1] - distanciaCm
    - distanciaSensorInferiorDPiso[Tanque - 1];
    porcentaje = (distanciaLlenado / distanciaEfectiva) * 100;
    if (porcentaje <= 0) { porcentaje = 0; }
    if (porcentaje >= 100) { porcentaje = 100; }
    volumen = (distanciaLlenado * diametroTanques[Tanque - 1] *
    diametroTanques[Tanque - 1] * PI * 0.25) * volLitros;
    if (porcentaje >= 100) {
      longitudPorcentaje = 3;
    } else if (porcentaje >= 10) {
      longitudPorcentaje = 2;
    } else {
      longitudPorcentaje = 1;
    }
  }
#endif
```

Figura 87: Código completo de la función que calcula el volumen y el porcentaje de agua en el tanque.

7.3.9 Función: infoBombasLCD

```
// Funcion informacion de estado de bombas en display LCD
void infoBombasLCD(int Bomba, int bombaInicialZona) {
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Bomba ");
  lcd.setCursor(6, 0);
  lcd.print(bombaInicialZona - 1 + Bomba);
  lcd.setCursor(0, 1);
  lcd.print("Estado: ");
  lcd.setCursor(8, 1);
  if (cargaBombas[Bomba - 1] == true) {
    lcd.print("Activada");
  } else {
    lcd.print("Apagada");
  }
}
```

Figura 88: Código completo de la función que muestra el estado de las bombas en el LCD.

7.3.10 Función: infoTanquesLCD

```
// Funcion informacion de nivel de tanques en display LCD
void infoTanquesLCD(int Tanque, int tanqueInicial, float porcentaje, int longitudPorcentaje) {
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Tanque ");
  lcd.setCursor(7, 0);
  lcd.print(tanqueInicial - 1 + Tanque);
  lcd.setCursor(0, 1);
  lcd.print("Nivel: ");
  lcd.setCursor(7, 1);
  if (porcentaje <= 0) {
    lcd.print("Vacio!");
  } else if (porcentaje >= 100) {
    lcd.print("Lleno!");
  } else {
    lcd.print(porcentaje);
    lcd.setCursor((10 + longitudPorcentaje), 1);
    lcd.print("%");
  }
}
```

Figura 89: Código completo para la función que muestra el estado de los tanques en la pantalla LCD.

7.3.11 Función: procesarSetEstadoBomba

```
#if numeromaxbombas > 0
  //Funcion RPC de biblioteca ThingsBoard para procesar las solicitudes RPC referidas a la activacion de las bombas independientes
  void procesarSetEstadoBomba(const JsonVariantConst &data, JsonDocument &response) {
    int bombaID = (data["idBomba"].as<int>()) - numeroBombaInicial;
    bool estadoBomba = data["estado"].as<bool>();

    if (bombaID <= numeroMaxBombas && bombaID >= numeroMaxBombas - cantidadBombasIndependientes) {
      digitalWrite(pinRele[bombaID], !estadoBomba);
      cargaBombas[bombaID] = estadoBomba;
    }
    response["nuevoEstadoBomba"] = estadoBomba ? "Encendida" : "Apagada";
  }
```

Figura 90: Código completo de la función que procesa las solicitudes RPC por la plataforma para el encendido y apagado de bombas.

7.3.12 Manejo de Callback: setEstadoBomba

```
405    // Define el array de callbacks RPC que apunta a la funcion procesarSe-
       tEstadoBomba cuando se invoca "setEstadoBomba"
406    const std::array<RPC_Callback, 1U> callbacks = {
407      RPC_Callback{ "setEstadoBomba", procesarSetEstadoBomba }
408    };
409  #endif
```

Figura 91: Código completo para el manejo de Callbacks que permite direccionar a "procesarSetEstadoBomba" cuando se invoca el método "setEstadoBomba".

7.3.13 Función: conexionWiFi

```
281  // Funcion de informacion de estado de conexion WiFi en display LCD
282  void conexionWiFi(int tiempoPanelEnS) {
283    tiempo = millis();
284    while (millis() - tiempo <= 1000 * tiempoPanelEnS) {
285      if (WiFi.status() == WL_CONNECTED) {
286        lcd.setCursor(0, 0);
287        lcd.print("Conexion");
288        lcd.setCursor(0, 1);
289        lcd.print("exitosa a WiFi");
290        intentos = 0;
291      } else if (WiFi.status() != WL_CONNECTED && intentos <= maxIntentos) {
292        WiFi.begin(ssid, contrasenaWiFi);
293        lcd.setCursor(0, 0);
294        lcd.print("Conectando a");
295        lcd.setCursor(0, 1);
296        lcd.print("WiFi...");
297        intentos++;
298        estadoDeConexionIoT = 0;
299      } else {
300        lcd.setCursor(0, 0);
301        lcd.print("Fallo de");
302        lcd.setCursor(0, 1);
303        lcd.print("conexion a WiFi");
304        estadoDeConexionIoT = 1;
305      }
306      if (banderaInterrupcionWifi == 1) {
307        lcd.clear();
308        banderaInterrupcionWifi = 0;
309        return;
310      }
311      retardoDeTiempo(1000);
312      lcd.clear();
313    }
314  }
```

Figura 92: Código completo para la función que muestra el estado de la conexión WIFI en el LCD.

7.3.14 Función: checkConexionTB

```
316  //Funcion para revisar la conexion a ThingsBoard en el menu del LCD
317  void checkConexionTB(int tiempoPanelEnS) {
318    tiempo = millis();
319    while (millis() - tiempo <= 1000 * tiempoPanelEnS) {
320      if (estadoDeConexionIoT == 1) {
321        lcd.setCursor(0, 0);
322        lcd.print("Conexion fallida");
323        lcd.setCursor(0, 1);
324        lcd.print("a nube IoT");
325      } else if (estadoDeConexionIoT == 2) {
326        lcd.setCursor(0, 0);
327        lcd.print("Conexion exitosa");
328        lcd.setCursor(0, 1);
329        lcd.print("a nube IoT");
330      } else {
331        lcd.setCursor(0, 0);
332        lcd.print("Conectando");
333        lcd.setCursor(0, 1);
334        lcd.print("a nube IoT...");
335      }
336      if (banderaInterrupcionWifi == 1) {
337        lcd.clear();
338        banderaInterrupcionWifi = 0;
339        return;
340      }
341      retardoDeTiempo(1000);
342      lcd.clear();
343    }
344  }
```

Figura 93: Código completo para la función que muestra el estado de la conexión a la plataforma IoT en el LCD.

7.3.15 Función: envioDatosThingsBoard

```
346 //Funcion para enviar datos a ThingsBoard referentes a los tanques y bombas
    locales
347 void envioDatosThingsBoard() {
348   //Envio de atributos locales por unica vez a ThingsBoard (numero maximo
      de tanques y numero maximo de bombas vinculadas al dispositivo)
349   if (enviarUnaVez != true) {
350     StaticJsonDocument<100> JSONObjAtr;
351     JSONObjAtr["tanquesVinculados"] = numeroMaxTanques;
352     JSONObjAtr["bombasVinculadas"] = numeroMaxBombas;
353     String jsonAtr;
354     serializeJson(JSONObjAtr, jsonAtr);
355     //Envio de atributos a ThingsBoard
356     tb.sendAttributeJson(jsonAtr.c_str());
357     enviarUnaVez = true;
358   }
359   #if numeromaxtanques > 0
360     //Envio Info de los tanques
361     for (int j = 1; j <= numeroMaxTanques; j++) {
362       // Determinar el estado de las variables a enviar
363       bool lecturaNivelInferior = digitalRead(sensorNivelBajo[j - 1]);
364       bool lecturaNivelSuperior = digitalRead(sensorNivelAlto[j - 1]);
365       calculoNivelTanque(j);
366       // Armar objeto JSON
367       StaticJsonDocument<100> JSONObjTan;
368       JSONObjTan["EstadoSensorInferiorTanque" + String(numeroTanqueInicial
        - 1 + j)] = lecturaNivelInferior;
369       JSONObjTan["EstadoSensorSuperiorTanque" + String(numeroTanqueInicial
        - 1 + j)] = lecturaNivelSuperior;
370       JSONObjTan["PorcentajeDeLlenadoTanque" + String(numeroTanqueInicial -
        1 + j)] = porcentaje;
371       JSONObjTan["VolumenTanque" + String(numeroTanqueInicial - 1 + j)] =
        volumen;
372       // Convertir objeto JSON a String
373       String jsonDataTan;
374       serializeJson(JSONObjTan, jsonDataTan);
375       //Envio de informacion a Things Board
376       tb.sendTelemetryJson(jsonDataTan.c_str());
377       retardoDeTiempo(100);
378     }
379   #endif
380   //Envio Info de las bombas
381   for (int i = 1; i <= numeroMaxBombas; i++) {
382     StaticJsonDocument<100> JSONObjBom;
383     JSONObjBom["EstadoBomba" + String(numeroBombaInicial - 1 + i)] =
      cargaBombas[i - 1];
```

```
384 String jsonDataBom;
385     serializeJson(JSONObjBom, jsonDataBom);
386     //Envio de informacion a Things Board
387     tb.sendTelemetryJson(jsonDataBom.c_str());
388     retardoDeTiempo(100);
389   }
390 }
```

Figura 94: Código completo para la función que permite enviar datos a la plataforma IoT ThingsBoard.

7.3.16 Función: timerEnvioDatos

```
411 // Establece la funcion que se ejecuta cuando desborda el timer
412 void IRAM_ATTR timerEnvioDatos() {
413   portENTER_CRITICAL_ISR(&timerMux);
414   if (WiFi.status() == WL_CONNECTED && pulsador == 0) {
415     checkTimerDatosOk = true;
416   }
417   checkReconexionWifi++;
418   portEXIT_CRITICAL_ISR(&timerMux);
419 }
```

Figura 95: Código completo para la función que se ejecuta cuando el timer desborda.

7.3.17 Función: setup

```
421 void setup() {
422   pinMode(pinPulsadorExt, INPUT_PULLDOWN);
423
424   #if numeromaxtanques > 0
425     for (int i = 1; i <= numeroMaxTanques; i++) {
426       pinMode(sensorNivelAlto[i - 1], INPUT_PULLDOWN);
427       pinMode(sensorNivelBajo[i - 1], INPUT_PULLUP);
428       pinMode(pinTrigTanques[i - 1], OUTPUT);
429       pinMode(pinEchoTanques[i - 1], INPUT);
430       attachInterrupt(digitalPinToInterrupt(sensorNivelAlto[i - 1]),
          interrupcionSensor, RISING);
431       attachInterrupt(digitalPinToInterrupt(sensorNivelBajo[i - 1]),
          interrupcionSensor, RISING);
432       if (digitalRead(sensorNivelBajo[i - 1]) == HIGH) {
433         #if numeromaxbombas > 0
434           encendidoLocalBombas(i);
435         #endif
436         checkTimerDatosOk = true;
437       }
438     }
439   #endif
440
```

```
441    #if numeromaxbombas > 0
442        for(int B=1; B <= numeroMaxBombas; B++){
443          digitalWrite(pinRele[B-1], HIGH);
444          pinMode(pinRele[B-1], OUTPUT);
445        }
446    #endif
447
448    lcd.init();  // Inicializar el LCD
449     attachInterrupt(digitalPinToInterrupt(pinPulsadorExt),
        interrupcionPulsador, RISING);
450
451    timer = timerBegin(1000000);
452    timerAttachInterrupt(timer, &timerEnvioDatos);
453    timerAlarm(timer, tiempoEnvioDatos * 1000000, true, 0);
454  }
```

Figura 96: Código completo para la función setup.

7.3.18 Función: loop

```
456 void loop() {
457   retardoDeTiempo(10);
458   while (pulsador == 1) {
459     timerStop(timer);
460     if (Tanque < numeroMaxTanques && millis() - tiempo >= (1000 *
      tiempoMenuLCD)) {
461       Tanque++;
462       tiempo = millis();
463     }
464     if (Tanque >= numeroMaxTanques && millis() - tiempo >= (1000 *
      tiempoMenuLCD) || numeroMaxTanques == 0 || Tanque >= numeroMaxTanques
      && Bomba == 1 || finBomba == 1) {
465       tiempo = millis();
466       Bomba = 1;
467       while (pulsador == 1) {
468         if (Bomba < numeroMaxBombas && millis() - tiempo >= (1000 *
          tiempoMenuLCD)) {
469           Bomba++;
470           tiempo = millis();
471         }
472         if (Bomba >= numeroMaxBombas && millis() - tiempo >= (1000 *
          tiempoMenuLCD) || finBomba == 1 || numeroMaxBombas == 0) {
473           lcd.clear();
474           conexionWiFi(tiempoMenuWifi);
475           checkConexionTB(tiempoMenuWifi);
476           lcd.noBacklight();
477           Tanque = 0;
478           Bomba = 0;
479           pulsador = 0;
480           finBomba = 0;
481           timerStart(timer);
482           return;
483         }
484         infoBombasLCD(Bomba, numeroBombaInicial);
485         retardoDeTiempo(1000);
486         lcd.clear();
487       }
488     }
489     #if numeromaxtanques > 0
490       calculoNivelTanque(Tanque);
491     #endif
492     infoTanquesLCD(Tanque, numeroTanqueInicial, porcentaje,
      longitudPorcentaje);
493     retardoDeTiempo(1000);
494     lcd.clear();
495   }
```

```
496      if (checkReconexionWifi >= tiempoCheckeoWifi / tiempoEnvioDatos) {
497       intentos = 0;
498       checkReconexionWifi = 0;
499       estadoDeConexionIoT = 0;
500     }
501     if (WiFi.status() != WL_CONNECTED && intentos <= maxIntentos) {
502       estadoDeConexionIoT = 0;
503       WiFi.begin(ssid, contrasenaWiFi);
504       retardoDeTiempo(1000);
505       if(intentos == maxIntentos){
506         estadoDeConexionIoT = 1;
507       }
508       intentos++;
509     }
510     if (WiFi.status() == WL_CONNECTED) {
511       if (!tb.connected()) {
512         while (!tb.connect(TB_SERVER, TOKEN)) {
513           // Espera a que la conexión sea exitosa
514           if (checkReconexionWifi >= tiempoCheckeoWifi / tiempoEnvioDatos) {
515             intentos = 0;
516             checkReconexionWifi = 0;
517             }
518           estadoDeConexionIoT = 1;
519           enviarUnaVez = false;
520           if (pulsador == 1) { return; }
521         }
522         #if numeromaxbombas > 0
523           if (!tb.RPC_Subscribe(callbacks.cbegin(), callbacks.cend())) {
524             return;
525           }
526         #endif
527         estadoDeConexionIoT = 2;
528       }
529       else {
530         estadoDeConexionIoT = 2;
531       }
532       if (checkTimerDatosOk == true) {
533         checkTimerDatosOk = false;
534         envioDatosThingsBoard();
535       }
536     }
537     tb.loop();
538   }
```

Figura 97: Código completo de la función principal loop.